Scalable Text Mining with Sparse Generative Models

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Doctor of Philosophy

at the

University of Waikato

by

Antti Puurula

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2015

Abstract

The information age has brought a deluge of data. Much of this is in text form, insurmountable in scope for humans and incomprehensible in structure for computers. Text mining is an expanding field of research that seeks to utilize the information contained in vast document collections. General data mining methods based on machine learning face challenges with the scale of text data, posing a need for scalable text mining methods.

This thesis proposes a solution to scalable text mining: generative models combined with sparse computation. A unifying formalization for generative text models is defined, bringing together research traditions that have used formally equivalent models, but ignored parallel developments. This framework allows the use of methods developed in different processing tasks such as retrieval and classification, yielding effective solutions across different text mining tasks. Sparse computation using inverted indices is proposed for inference on probabilistic models. This reduces the computational complexity of the common text mining operations according to sparsity, yielding probabilistic models with the scalability of modern search engines.

The proposed combination provides sparse generative models: a solution for text mining that is general, effective, and scalable. Extensive experimentation on text classification and ranked retrieval datasets are conducted, showing that the proposed solution matches or outperforms the leading task-specific methods in effectiveness, with a order of magnitude decrease in classification times for Wikipedia article categorization with a million classes. The developed methods were further applied in two 2014 Kaggle data mining prize competitions with over a hundred competing teams, earning first and second places.

Contents

Abstract iii

List of Abbreviations and Acronyms viii

Notation and Nomenclature ix

List of Notations x

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Thesis Statement . . . 2
- 1.3 Contributions of the Thesis . . . 3
- 1.4 Published Work . . . 5
- 1.5 Structure of the Thesis . . . 7

2 Text Mining and Scalability 9
- 2.1 Introduction to Text Mining . . . 9
  - 2.1.1 Defining Text Mining . . . 9
  - 2.1.2 Related Fields . . . 12
  - 2.1.3 Application Domains . . . 15
- 2.2 Text Mining Methodology . . . 18
  - 2.2.1 Text Documents as Multiply Structured Data . . . 18
  - 2.2.2 Structured Representations for Text . . . 22
  - 2.2.3 Text Mining Applications as Machine Learning Tasks . . . 26
  - 2.2.4 Linear Models as Methods for Text Mining . . . 29
  - 2.2.5 Text Mining Architectures as KDD Processes . . . 34
- 2.3 The Scalability Problem . . . 36
  - 2.3.1 Scale of Text Data . . . 36
  - 2.3.2 Views on Scalability . . . 39
  - 2.3.3 Approaches to Scalable Text Mining . . . 41

3 Multinomial Naive Bayes for Text Mining 47
- 3.1 Multinomial Naive Bayes . . . 47
  - 3.1.1 Introduction . . . 47

3.1.2 Definition . . . 50
3.1.3 Estimation . . . 53
3.2 Generative Models Extending MNB . . . 55
3.2.1 Mixture Models . . . 55
3.2.2 N-grams and Hidden Markov Models . . . 59
3.2.3 Directed Graphical Models . . . 62
3.2.4 Factor Graphs and Gates . . . 66
3.2.5 Inference and Estimation with Directed Generative Models 68
3.2.5.1 Overview of Algorithms . . . 68
3.2.5.2 Dynamic Programming . . . 70
3.2.5.3 Expectation Maximization . . . 72

4 Reformalizing Multinomial Naive Bayes 75
4.1 Formalizing Smoothing . . . 75
4.1.1 Smoothing Methods for Multinomials . . . 75
4.1.2 Formalizing Smoothing with Two-State Hidden Markov Models . . . 80
4.2 Extending MNB for Fractional Counts . . . 86
4.2.1 TF-IDF and Feature Transforms with MNB . . . 86
4.2.2 Methods for Fractional Counts with Multinomial Models 87
4.2.3 Formalizing Feature Transforms and Fractional Counts with Probabilistic Data . . . 89
4.3 Formalizing MNB as a Generative Directed Graphical Model . . 92
4.4 Extending MNB with Prior Scaling and Document Length Modeling . . . 96

5 Sparse Inference 99
5.1 Basic Case: Sparse Posterior Inference . . . 99
5.2 Extension to Joint Inference on Hierarchically Smoothed Sequence Models . . . 102
5.3 Extension to Joint Inference on Mixtures of Sequence Models . . 106
5.4 Further Specialized Efficiency Improvements for Sparse Inference . . . 107
5.5 Tied Document Mixture: A Sparse Generative Model . . . 109

6 Experiments 115
6.1 Methodology . . . 115
6.1.1 Experimental Framework . . . 115
6.1.2 Performance Measures . . . 117
6.1.3 Baseline Methods . . . 120
6.1.4 Parameter Optimization . . . 122
6.1.5 Significance Tests . . . 125

6.2 Datasets . . . 126
6.2.1 Dataset Overview . . . 126
6.2.2 Preprocessing . . . 127
6.2.3 Segmentation . . . 129
6.2.4 Dataset Statistics . . . 130
6.3 Experiments and Results . . . 133
6.3.1 Evaluated Linear Model Modifications . . . 133
6.3.2 Smoothing Methods . . . 134
6.3.3 Feature Weighting and the Extended MNB . . . 136
6.3.4 Tied Document Mixture . . . 137
6.3.5 Comparison with Strong Linear Model Baselines . . . 137
6.3.6 Scalability and Efficiency . . . 144

7 Conclusion **151**
7.1 Summary of Results . . . 151
7.2 Implications of Findings . . . 152
7.3 Revisiting the Thesis Statement . . . 153
7.4 Limitations of the Thesis . . . 154
7.5 Future Work . . . 155

Appendix A Tables of Results **159**

Appendix B Kaggle LSHTC4 Winning Solution **181**

List of Abbreviations and Acronyms

BM25	Best Match 25
BNB	Bernoulli Naive Bayes
DBN	Dynamic Bayes Network
DM	data mining
EM	expectation maximization
HMM	Hidden Markov Model
IDF	inverse document frequency
IE	information extraction
IID	independent and identically distributed
IR	information retrieval
KDD	knowledge discovery in databases
KDT	knowledge discovery in textual databases
LM	language model
LR	Logistic Regression
LSHTC	large-scale hierarchical text classification
MAP	mean average precision
Micro-F1	micro-averaged F1-score
ML	machine learning
MNB	Multinomial Naive Bayes
NB	Naive Bayes
NDCG	normalized discounted cumulative gain
NLP	natural language processing
SVM	Support Vector Machine
TDM	Tied Document Mixture
TF	term frequency
TF-IDF	term frequency-inverse document frequency
TM	text mining
VSM	Vector Space Model

Notation and Nomenclature

The notation used in the thesis follows closely the linear algebra notation used in statistical natural language processing and machine learning, graphical models literature in particular [Manning and Schütze, 1999, Bishop, 2006]. Counts of words in a document are represented using a word vector $\boldsymbol{w}$, and a sequence of words is represented using a word sequence $\underline{\boldsymbol{w}}$. Mixture model notation is used to denote models conditional on variables, e.g. $p_l(n) = p(n|l)$ and $p_{li\underline{k}_j}(\underline{w}_j) = p(\underline{w}_j|l, i, \underline{k}_j)$. Apostrophe is used to reduce notation, by indicating derived functions and variables explained in the context, e.g. $\sum n'$ indicates a sum over the same variable type as n and $p'(n)$ indicates a function related to $p(n)$. Information retrieval and natural language processing terminology is reduced, e.g. "word" is used ambiguously to refer to word types and tokens.

List of Notations

$\boldsymbol{w}$	vector of word counts $\boldsymbol{w} = [w_1, ..., w_n, ..., w_N]$, ordinarily integer counts $w_n \in \mathbb{N}^0$
n	word variable, word vector index $1 \leq n \leq N$
N	number of distinct words, dimension of a word vector $N = \lvert\boldsymbol{w}\rvert$
$\underline{\boldsymbol{w}}$	sequence of words $\underline{\boldsymbol{w}} = [\underline{w}_1, ..., \underline{w}_j, ..., \underline{w}_J]$
j	word sequence index $1 \leq j \leq J$, $1 \leq \underline{w}_j \leq N$
J	length of document $J = \lvert\boldsymbol{w}\rvert_1 = \lvert\underline{\boldsymbol{w}}\rvert$
l	label index variable $1 \leq l \leq L$
$\boldsymbol{c}$	label vector variable $\boldsymbol{c} = [c_1, ..., c_l, ..., c_L]$, $c_l \in \{0, 1\}$
L	number of distinct labels in a collection
D	collection, a dataset of text documents
i	document index variable $1 \leq i \leq I$, $D^{(i)}$
I	number of distinct documents
$D^{(i)}$	document i of dataset D, including possible meta-data, $D^{(i)} = \boldsymbol{w}^{(i)}$, $D^{(i)} = (\underline{\boldsymbol{w}}^{(i)}, l^{(i)})$
m	mixture component index $1 \leq m \leq M$, α_m, $p(m)$, $p_m(n)$
M	number of mixture components, maximum n-gram order
$\underline{\boldsymbol{k}}$	sequence of mixture assignment indicators $\underline{\boldsymbol{k}} = [\underline{k}_0, , \underline{k}_j, , \underline{k}_J]$, $1 \leq \underline{k}_j \leq M$
$\boldsymbol{\theta}$	vector of model parameters, for a multi-class linear model $y(\boldsymbol{\theta}_l, \boldsymbol{w}) = \theta_{l0} + \sum_{n=1}^{N} \theta_{ln} w_n$, with bias θ_{l0}
$C(l, n)$	count of joint occurrences of variables in collection, $C(l, n) = \sum_{i:l^{(i)}=l} w_n^{(i)}$
$D(l, n)$	discount applied to a count $C(l, n)$
$p_l^u(n)$	unsmoothed multinomial $p_l^u(n) = \frac{C(l,n)-D(l,n)}{\sum_n' C(l,n')-D(l,n')}$
α	back-off weight, determined by the smoothing method
$\underline{\boldsymbol{r}}$	sequence of word weights, interpreted as probabilities of words occurring, $\boldsymbol{r} = [r_1, ..., r_j, ..., r_J]$, $r_j \in [0, 1]$

Chapter 1

Introduction

This chapter introduces the topic of the thesis and motivation for research. It presents a thesis statement based on the results of the research, lists the contributions of the thesis compared to the existing literature, references publications by the author related to the thesis, and describes the structure of the thesis.

1.1 Motivation

The information age has brought an information overflow. The Internet provides a highway where vast amounts of data can be instantly searched and retrieved. This data presents a source that can be mined for knowledge about the world and to improve decision making. A considerable, and perhaps the most valuable, portion of this data is in text form, such as newspapers, web pages, books, emails, blogs and chat messages.

The field of artificial intelligence known as *text mining* has become an intersection between data mining, information retrieval, machine learning and natural language processing. Since its birth in the mid 90's, it has shown consistent growth in research publications, and has been applied in numerous ways in both industry and academia. Companies use text mining to monitor opinions related to their brands, while traditionally qualitative sciences such as the humanities use it as an empirical methodology for research. However, the fragmentation of text mining research has resulted in a variety of tools specialized for different applications.

Text mining applications are mapped into general statistical and machine learning tasks, such as classification, ranking, regression and clustering. The increase of data and computing power enables performing previously impos-

sible tasks using statistical models. E-mail spam filters can be trained with trillions of text documents, and cover billions of words. Documents can be classified into a Wikipedia article hierarchy with millions of categories. A major limit on the possibilities of text mining is the *scalability* in the dimensions of data. General data mining models have not been developed for the sparse and increasingly high dimensional data encountered in text processing tasks. Data mining models that can easily operate on thousands of documents and words can be unusable on datasets with millions of documents and words. The main solution to scalability is to redesign algorithms that have no more than linear computational complexities in terms of documents, words, and class variables.

Given these problems of fragmentation and scalability, it would be useful to have models that are both *versatile* and *scalable*. A versatile model for text mining would have to applicable to different task types with high performance. A scalable model for text mining would have to scale in all of the relevant dimensions of the applied task.

Following an overview of multinomial generative models of text, the thesis proposes extensions of the common Multinomial Naive Bayes (MNB) model as a family of versatile models for text mining. It is shown that when MNB is modified to correct its modeling assumptions, it forms a versatile solution to text mining that is far from the "punching bag of machine learning" that basic Naive Bayes models have been called [Lewis, 1998, Rennie et al., 2003]. By using inverted index data structures, it is shown that many of the processing operations with MNB and its graphical model extensions can be solved as a function of sparsity, turning the "curse of dimensionality" [Bellman, 1952, Feldman and Sanger, 2006] with sparse text data into a blessing. *Sparse generative models* combining generative models with sparse inference offer a versatile and scalable solution to text mining.

1.2 Thesis Statement

The thesis statement is as follows:

Generative models of text combined with inference using inverted indices provide sparse generative models for text mining that are both versatile and scalable, providing state-of-the-art effectiveness and high scalability for various text mining tasks.

1.3 Contributions of the Thesis

The thesis presents a synthesis of the fragmented text mining literature, and describes the generative multinomial models of text used across various text mining tasks. Based on the literature, the thesis proposes a solution to text mining that is both scalable and applicable to diverse tasks. It is shown that many of the multinomial models of text can be formalized as instances and extensions of the MNB model, including the n-gram language models that are widely applied across a variety of text processing applications. This common formalization enables the transfer of modeling innovations across different tasks and research literatures, and provides a high-performing baseline across different task types. Connections to linear and graphical models are shown, and extensions within these frameworks offer MNB even greater flexibility.

Lack of a proper formalization of a computational method means that the behaviour of a method is not explained by any existing theory, and it can behave erratically. This thesis formalizes modifications to MNB such as smoothing, feature weighting and structural extensions, showing that for most uses the modified models are not heuristic, but well-defined graphical models that extend the basic MNB model, in some cases using approximate maximum likelihood estimates. Formalizing these modifications means that the models are well-defined in a probabilistic sense, and their behaviour can be understood from the underlying probability theory.

The proposed sparse inference builds on the use of inverted indices for information retrieval with multinomial models of documents. It is shown that this type of inference is not limited to computing ranking scores for document retrieval, but can be used to compute exact posterior probabilities from any linear model for various uses, such as ranking, classification and clustering. Furthermore, sparse inference is extended to structural models, yielding the same improvements in scalability and computational complexity for many practical cases.

Finally, the thesis presents an extensive empirical evaluation of the proposed models and inference over tens of standard datasets used for text classification and ranked retrieval. The experiments give strong evidence in support of the thesis statement. In both types of tasks, modifications of MNB outperform or rival strong baseline methods commonly used for these tasks in effectiveness. Scalability experiments are conducted on a large Wikipedia article classification task, showing that the proposed sparse inference improves scaling of the models into tasks with a million words, documents, and classes.

An extensive literature review has been conducted to verify the originality of the contributions. As expected, some of the contributions have prior and concurrent work. The following list highlights the most fundamental contributions that are not found in the prior literature, to the best of the author's knowledge:

Synthesis of text mining methodology Surveys of text mining have limited their perspectives to mostly a few of its influences, such as data mining [Witten, 2004, Hotho et al., 2005, Feldman and Sanger, 2006, Stavrianou et al., 2007, Weiss et al., 2012], machine learning [Weiss et al., 2012, Aggarwal and Zhai, 2012], information retrieval [Aggarwal and Zhai, 2012], and natural language processing [Hearst, 1999, Black, 2006]. Chapter 2 gives a concise synthesis of current text mining methodology, incorporating the perspectives of various practitioners in a coherent framework. Text data is shown to be multiply structured data from a linguistic point of view, many of the core algorithms used for text mining are shown to be cases of linear models, and possible views and solutions to the scalability problem are discussed.

Formalization of smoothing with two-state Hidden Markov Models Early, but discontinued research showed that the linearly smoothed n-gram language models used for information retrieval could be formalized with two-state Hidden Markov Models (HMM) [Miller et al., 1999, Xu and Weischedel, 2000, Hiemstra, 2001]. Chapter 4 shows that all of the smoothing methods for multinomial generative models can be expressed as exact inference on a two-state HMM. The formalization further shows that most of the parameter estimates for the smoothed multinomials derive from expected log-likelihood parameter estimation on a two-state HMM. Heuristically defined backoff models for Kneser-Ney [Kneser and Ney, 1995] and power-law [Huang and Renals, 2010] discounting are shown to derive from constraints applied to the two-state HMM.

Formalization of feature weighting with probabilistic data Concurrent research has derived the expected log-likelihood estimation of n-gram language models from probabilistically weighted data [Zhang and Chiang, 2014]. Chapter 4 extends this probabilistic data view to inference, showing that both estimation and inference is well defined for probabilistically weighted word sequences and non-negative fractional word counts. This greatly extends the versatility of generative models of text, as data can be weighted to correct modeling assumptions, without losing the probabilistic formalization.

Sparse inference Earlier research has applied inverted indices for reducing the classification times for K-nearest Neighbours [Yang, 1994] and Centroid [Shanks et al., 2003]. The same reductions are gained for computing posterior probabilities for linearly interpolated language models in information retrieval [Hiemstra, 1998, Zhai and Lafferty, 2001b]. Chapter 5 shows that inverted indices can be used to reduce the inference complexity according to sparsity for a variety of processing tasks, including linear models and structural extensions of linear models. Applied to Wikipedia classification with a million possible categories, an order of magnitude reduction of classification times is obtained for Tied Document Mixture, a novel structural extension of MNB.

Evaluation of modified generative models Evaluation of text mining methods has mostly been limited to experiments in one task type with a few possible methods [Sebastiani, 2002, Huston and Croft, 2014]. Chapter 6 presents a consistent framework for evaluation of text mining models applied for both text classification and ranked retrieval tasks. Model parameters are searched using Gaussian random search optimization, avoiding the problem of local optima. Statistical testing is conducted between datasets, measuring the strength of the discovered effects across text collections. Experiments are conducted over tens of text classification and retrieval datasets, comparing a large set of modified MNB variants with strong baseline methods for the tasks.

1.4 Published Work

The thesis includes and expands on earlier work by the author, most published in peer-reviewed conferences:

[**Puurula, 2011a**] *Mixture Models for Multi-label Text Classification*, New Zealand Computer Science Research Student Conference in Palmerston North, New Zealand, 2010. An analysis of generative mixture models for text modeling

[**Puurula, 2011b**] *Large Scale Text Classification with Multi-label Naive Bayes*, Second Smart Information Technology Applications Conference in Seoul, South Korea, 2011. A generative multi-label mixture model for modeling text

[**Puurula, 2012a**] *Scalable Text Classification with Sparse Generative Modeling*, Pacific Rim International Conference on Artificial Intelligence in Kuching, Malaysia, 2012. Earliest form of sparse posterior inference for MNB and a multi-label mixture model extension

[**Puurula and Bifet, 2012**] *Ensembles of Sparse Multinomial Classifiers for Scalable Text Classification*, ECML/PKDD - PASCAL Workshop on Large-Scale Hierarchical Classification in Bristol, United Kingdom, 2012. An ensemble of sparse generative models used successfully in a machine learning competition

[**Puurula, 2012b**] *Combining Modifications to Multinomial Naive Bayes for Text Classification*, Asian Information Retrieval Symposium in Tianjin, China, 2012. Combinations of modifications to Multinomial Naive Bayes examined

[**Puurula and Myaeng, 2013**] *Integrated Instance- and Class-based Generative Modeling for Text Classification*, Australasian Document Computing Symposium in Brisbane, Australia, 2013. Early version of the Tied Document Mixture model presented with sparse posterior inference

[**Puurula, 2013**] *Cumulative Progress in Language Models for Information Retrieval*, Australasian Language Technology Association Workshop in Brisbane, Australia, 2013. Combinations of modifications to information retrieval language models examined

[**Puurula et al., 2014**] *Kaggle LSHTC4 Winning Solution*, Arxiv.org preprint, 2014. A description of the winning solution to the Kaggle Large Scale Hierarchical Text Classification competition, with an ensemble of sparse generative models

[**Tsoumakas et al.**] *WISE 2014 Challenge: Multi-label Classification of Print Media Articles to Topics*, Web Information Systems Engineering in Thessaloniki, Greece, 2014. A report on the Kaggle WISE competition, where an ensemble using sparse generative models as components came second

[**Trotman et al., 2014**] *Improvements to BM25 and Language Models Examined*, Australasian Document Computing Symposium in Melbourne, Australia, 2014. Exploration of recent information retrieval ranking functions, including generative language models discussed in the thesis. *Best paper award*

Open source code related to the thesis is distributed online, making the methods presented here available for wider use. The SGMWeka[1] open source toolkit for sparse generative modeling is available through SourceForge.net, as well as dataset preprocessing scripts required to reproduce the results shown in

[1] http://sourceforge.net/projects/sgmweka/

the thesis. The Kaggle LSHTC4 winning solution[2] is available via the Kaggle website, making it possible to replicate the winning methods. The competition description of the LSHTC4 solution is included in Appendix B of the thesis.

1.5 Structure of the Thesis

The thesis is written to be accessible to readers of differing backgrounds. The introductory chapters, as well as the experiments and conclusion are intended to be readable by most. The chapters introducing novel mathematical ideas require extensive background knowledge in probability theory and statistical mathematics, and are recommended mainly for researchers. The rest of the thesis is structured as follows:

Chapter 2 introduces the topic of text mining, covering the terminology and methodology of text mining that will be used in the following chapters. The emerging field of text mining is highly fragmented, and the used terms and methods differ widely. The chapter includes an extensive literature review of the topic, and presents the many facets of text mining in an integrated framework that is accessible to readers without extensive mathematical background.

Chapter 3 introduces the Multinomial Naive Bayes model for text mining and its extensions with generative graphical models. This chapter establishes the mathematical notation that will be used throughout the rest of the thesis, and defines concepts such as graphical models and dynamic programming. This chapter is written to be accessible to readers with basic understanding of probability theory.

Chapter 4 presents a more detailed analysis of the MNB model for text mining. It is shown that all of the commonly used smoothing methods for correcting data sparsity with multinomial text models can be formalized as approximate maximum likelihood estimation on a constrained Hidden Markov Model. It is shown that feature weighting can be equally formalized for MNB models and its extensions. Furthermore, practical graphical model extensions of MNB are proposed that maintain the efficiency of the model, while providing greater effectiveness and modeling flexibility. This chapter is accessible to readers with experience in deriva-

[2]http://www.kaggle.com/c/lshtc/forums/t/7980/winning-solution-description

tions for graphical models.

Chapter 5 presents the idea of sparse inference for MNB, and more generally for linear models and structured extensions of linear models. The complexity of inference is reduced as a function of sparsity, by using inverted index representation of model parameters. This chapter is the most technically demanding in the thesis, and contains novel algorithms and derivations.

Chapter 6 presents an extensive empirical evaluation of the modeling ideas presented in Chapters 4 and 5 in the context of text classification and retrieval. In terms of effectiveness, it is demonstrated that the proposed extensions of MNB models greatly improve on the commonly used generative models for these tasks, providing results competitive with strong baseline methods for both tasks. In terms of scalability, it is shown that MNB with sparse inference easily scales to classification with a million features, documents and classes. Sparse inference on structured extensions of MNB scale with a similarly reduced time complexity, reducing inference times by an order of magnitude in the highest-dimensional cases examined.

Chapter 7 concludes the thesis with a summary of a thesis, revisits the thesis statement, and discusses the implications of the findings, limitations of the thesis, and possible future work.

Chapter 2

Text Mining and Scalability

This chapter presents a brief introduction to text mining, followed by a comprehensive overview of text mining methodology, and a discussion on the scalability problem in text mining. The introduction discusses the variety of definitions for text mining, related fields preceding text mining, and domains that apply text mining. The overview of text mining methodology provides a synthesis of viewpoints on text mining, starting from the linguistic properties and representation of text data, followed by mapping of text mining problems into machine learning tasks, and finally comparing text mining architectures to knowledge discovery processes. The discussion on scalability describes the scalability problem in text mining with examples, implicit views on scalability taken by researchers and practitioners, and existing approaches to scalability.

2.1 Introduction to Text Mining

2.1.1 Defining Text Mining

Progress in information technology has brought an information overflow, with transformative societal implications that affect all aspects of human life. A considerable and possibly the most significant portion of this information is in the form of text data, such as books, news articles, microblogs and instant messages. These vast quantities of text data can only be accessed and utilized using computers, but the automated processing of text is only possible using technology specialized for human language. Text mining (TM) in a broad sense refers to technology that allows the utilization of large quantities of text data. In the following, this working definition will be amended by a more concise one.

Text mining originates from several earlier research fields, such as data mining, machine learning, information retrieval, natural language processing. Like

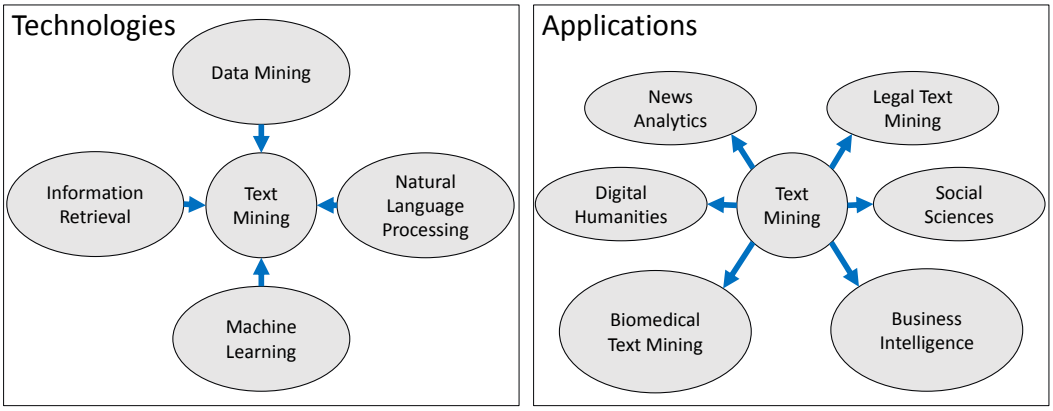


Figure 2.1: Relationship of TM to the major related fields (technologies) and application domains (applications)

these fields, TM has a foundation in computer science, with considerable influence from applied artificial intelligence [Fayyad et al., 1996, Witten, 2004]. It is highly related and sometimes used interchangeably with terms such as information extraction, opinion mining and text analytics. TM is used in a variety of application domains, such as biomedical TM and business intelligence. The related fields have influenced TM in terminology and methodology, whereas the application domains have been influenced by TM. Figure 2.1 illustrates the relationship of TM to the major related fields and application domains. These relationships will be discussed in detail in Sections 2.1.2 and 2.1.3.

The term "text mining" originated in the data mining publications of mid 1990's. Feldman et al. wrote a series of publications starting from 1995 under the term "knowledge discovery in textual databases" (KDT) [Feldman and Dagan, 1995, Feldman et al., 1997, 1998, Feldman and Sanger, 2006], and by 1997 a number of authors used the term text mining [Ahonen et al., 1997a, Rajman et al., 1997, Feldman et al., 1997, Tkach, 1997]. The early proponents of TM considered it to be an application of data mining to text data, and saw text data as "unstructured data" that needs to be structured for use in data mining: *"before we can perform any kind of knowledge discovery in texts we must extract some structured information from them"* [Feldman and Dagan, 1995]. This KDT definition viewed TM as data mining, with natural language processing and information retrieval as preprocessing and indexing steps in the mining process [Feldman and Dagan, 1995, Feldman et al., 1997, 1998, Ahonen et al., 1997a,b, Albrecht et al., 1998, Dörre et al., 1999, Tkach, 1997, Liddy, 2000]. A slight variation of this KDT definition was TM as a different type of process from general data mining [Rajman et al., 1997, Rajman and Besanon, 1998, Merkl, 1998, Witten et al., 1999, Tan, 1999, Witten, 2000a,b].

Within a few years TM started to interest other research communities. Natural language processing and computational linguistics researchers saw TM as a potential application [Hearst, 1999]. Many of the leading machine learning methods of the next decade were developed and popularized in the context of modeling text [Elkan, 1997, Joachims, 1998, Lewis, 1998, Lafferty et al., 2001, Hofmann, 1999, Blei et al., 2003], providing TM practitioners an advanced toolkit. Information retrieval had extended earlier into information extraction, and its similarities to TM were discovered at this time as well [Nahm, 2004, Mooney and Bunescu, 2005]. The definitions of TM gradually diversified away from Feldman's KDT definition of TM, as the term slowly started to be used in exceptionally diverse contexts in both academia and business. Witten [2004] in his review discusses the problems of defining TM, and explicitly avoids providing a concise definition. Both Black [2006] and Hotho et al. [2005] give a definition of TM close to the KDT definition, while noting the diversity of definitions. Cohen and Hersh [2005] in their survey of biomedical TM avoid providing an explicit definition. Stavrianou et al. [2007] in their survey give the KDT definition, while Weiss et al. [2012] appears to implicitly use the KDT definition.

Although the KDT definition is a very simple characterization of TM, it is not very descriptive in practice. Perhaps the biggest problem with the definition is that it does not capture what is unique about TM. TM overlaps many fields to the extent that any of its applications could equally be considered as problems of the related fields. What makes TM is unique is not the tasks and problems it shares with the related fields, but the *interdisciplinarity* and *integration of methods* for solving the problems.

An example of a TM problem could be a web monitoring system for analyzing sentiment related to a brand. A system of this type would require methods from information retrieval to search text data related to the brand, natural language processing and information extraction for extracting parts of the text that refer to the brand, and machine learning for predicting the sentiment. The system would further need text visualization and statistical tests for confirming the reliability of the predictions. An integrated architecture for constructing such a system would most accurately be called a TM solution.

The view of TM as integration of artificial intelligence-based text processing technologies captures the main novelty of TM. Perhaps the first definition of TM from this point of view was reflected in the title of the KDD'2000 work-

shop on TM: *"Text Mining as Integration of Several Related Research Areas"* [Grobelnik et al., 2000]. Feinerer et al. [2008] avoids choosing a definition, but seems to support this view as well: *"In general, text mining is an interdisciplinary field of activity amongst data mining, linguistics, computational statistics, and computer science"*. For the purpose of this thesis, a concise definition is proposed:

Text mining is an interdisciplinary field of research on the automatic processing of large quantities of text data for valuable information.

2.1.2 Related Fields

Text mining originates from earlier and well-established fields grounded in computer science and artificial intelligence, the four major ones being data mining (DM), information retrieval (IR), natural language processing (NLP) and machine learning (ML). All of these fields are interdisciplinary with a considerable amount of participation from a diverse range of academic subjects related to computer science. As information technology is becoming increasingly prevalent in the modern world, much of the research in these fields is becoming distributed and applied across every discipline in the academic world, including the "soft sciences" of the humanities that have previously relied on qualitative methodologies. Outside academia, these fields exist as viable industries, with a global market of start-up and large-cap companies alike. A comparison of TM and related fields is given in the following.

Data mining and knowledge discovery in databases (KDD) [Fayyad et al., 1996, Chen et al., 1996] deal with the discovery of useful patterns in large databases, and have origins in statistics, machine learning and databases. Within KDD, DM constitutes the algorithms used for discovery of patterns, whereas KDD refers to the overall interactive process, where the user explores a dataset [Fayyad et al., 1996]. The term TM originated in DM research, and DM certainly remains one of the major influences in current TM. Many of the definitions used for KDD apply equally to TM today: *"The KDD process can be viewed as a multidisciplinary activity that encompasses techniques beyond the scope of any one particular discipline such as machine learning."* [Fayyad et al., 1996] and *"KDD also emphasizes scaling and robustness properties of modeling algorithms for large noisy data sets."* [Fayyad et al., 1996]. Multidisciplinarity and scalability are equally defining qualities of TM. To some extent, TM shares the idea of *process models* that are applied in different tasks types [Ahonen et al., 1997a,b, Liddy, 2000]. An early view was that TM is simply DM

with a text-specific preprocessing phase and an additional document filtering phase [Ahonen et al., 1997a, Dörre et al., 1999], but current TM systems are better described as architectures than a process [Feldman and Sanger, 2006]. A TM system neither requires a user discovering new patterns: TM can be used to automatically monitor existing well-known patterns in text, such as sentiment and topic. The DM characterization of text as "unstructured data" is also a broad generalization: text has shown to be a unique type of data structured in multiple ways, requiring specialized methods very different from general DM. The goals of TM and DM sometimes differ: the TM output is not necessarily hard facts or quantifiable values, but "soft information" in the form of text. Overall, there is surprisingly little interaction between TM and DM today, although much of TM can be situated in the context of KDD.

Machine learning deals with systems that learn from data, and has origins in statistics, artificial intelligence and computer science. For a given learning task and performance measure, a learning system improves its performance using data [Mitchell, 1997]. This contrasts with statistics, where the emphasis is on finding the correct models for data, and not on directly optimizing performance [Breiman, 2001a]. The division in goals has led to a division between the "two cultures" of traditional statistics and machine learning [Breiman, 2001a]. The success of ML has lead to it being adopted as a general framework in a variety of application domains requiring artificial intelligence. Much of ML has dealt with text data [Joachims, 1998, Sebastiani, 2002, Lafferty et al., 2001, Blei et al., 2003], and much of TM is based on the application of ML methods; text classification in particular. The division of TM into distinct task types [Feldman and Sanger, 2006, Aggarwal and Zhai, 2012] also follows the general ML framework. Like with DM, the main reason for not considering TM as simply an application of ML is the uniqueness of text data. Techniques such as inverted indices have proven crucial for processing text, yet these are virtually unknown in ML. Although the majority of TM methods originate in ML, TM systems also require tools that are specialized for text, originating from a variety of disciplines, some requiring no learning, and some constructed with highly specialized human expertise.

Information retrieval deals with systems for retrieval and ranking of documents for a given information need. The ubiquitous case is web search, where the information need is expressed as a query consisting of words, and the ranked set of documents consists of webpage links. Modern research into IR started in the context of computerized library indexing systems [Maron, 1961, Manning et al., 2008], where the number of paper documents was increasing

rapidly. This parallels the arrival of vast amounts of digital documents and the development of TM half a century later. Although IR is more general and deals with different types of data, text is the main type of information used to index most forms of data, and text retrieval using word vectors and inverted indices constitutes the main methodology of current IR [Manning et al., 2008]. These two text retrieval techniques also constitute key components for TM, since they enable scalable search of documents. IR has become possibly the main influence on TM, as it has expanded into more complex tasks after progress in ad-hoc text retrieval was considered to be stagnant at the end of the millennium [Voorhees and Harman, 1999].

Natural language processing or computational linguistics deals with the processing of text data using algorithms based on linguistic theory, and originated in the considerable efforts to develop machine translation during the early years of the cold war [Pierce, 1966]. Basic NLP tasks are segmenting, parsing, and annotating text according to the underlying linguistic structure of the data [Manning and Schütze, 1999]. Corpus linguistics refers to the goal of producing well-defined computational theories to understand natural language, whereas the term NLP is closer to the goal of developing practical language technology. TM was proposed early as an additional goal for computational linguistics [Hearst, 1999], and NLP has certainly had a large influence on TM. Statistical NLP in particular predated TM by integrating IR, statistics and NLP [Manning and Schütze, 1999]. Research in NLP has relied on annotated digital corpora such as the Brown corpus [Kucera and Francis, 1967]. In contrast, TM often uses unannotated, large scale, and noisy datasets, with the goal of extracting information of value. NLP is not as interdisciplinary as TM; TM research exists across a wide variety of problem domains, whereas NLP research is seldom conducted in other fields.

Related to statistical NLP is another field at the intersection of NLP and IR deserving a mention. Information extraction (IE) research started in the context of the Message Understanding Conference series starting from 1987 [Grishman and Sundheim, 1996, Sarawagi, 2008]. The goal of IE is the extraction of predetermined patterns from text collections, such as names, places and events. IE therefore has a more limited and concise goal than TM, but the two fields overlap to the extent that sometimes little difference is seen between the two [Sebastiani, 2002, Stavrianou et al., 2007]. One viewpoint is considering IE as an intermediate step to TM, where the extracted patterns are used for discovery of more complex information [Nahm, 2004, Mooney and Bunescu, 2005]. Text summarization deals with the extraction of human-readable sum-

maries of text, and is a type of TM task that falls outside the scope of IE [Witten, 2004, Sarawagi, 2008, Das and Martins, 2007].

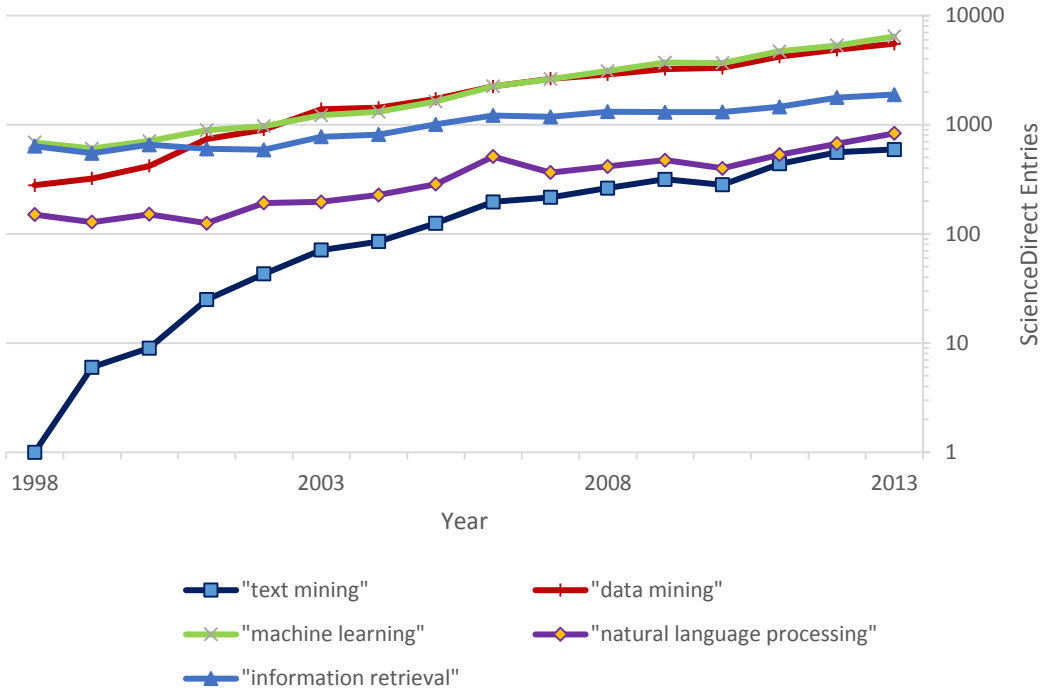
One way to measure the development of TM compared to the related fields is to count the number of publications that use the term over the years. Figure 2.2 shows the number of publications for years 1998-2013 indexed by Google Scholar[1] and ScienceDirect[2] containing the names of the fields. As can be seen, for a decade TM has grown at a steady rate of 800 more publications each year according to Google Scholar, and 50 according to ScienceDirect. Analyzing these graphs relies on a couple of assumptions: the quality of indexed publications is roughly similar, and that terminological ambiguity such as the use of "computational linguistics" for "natural language processing" does not distort the results. Since both indices agree on the main effect compared to the four other fields, this is most likely correct: published TM research has grown to a volume that the related fields had around year 2000.

2.1.3 Application Domains

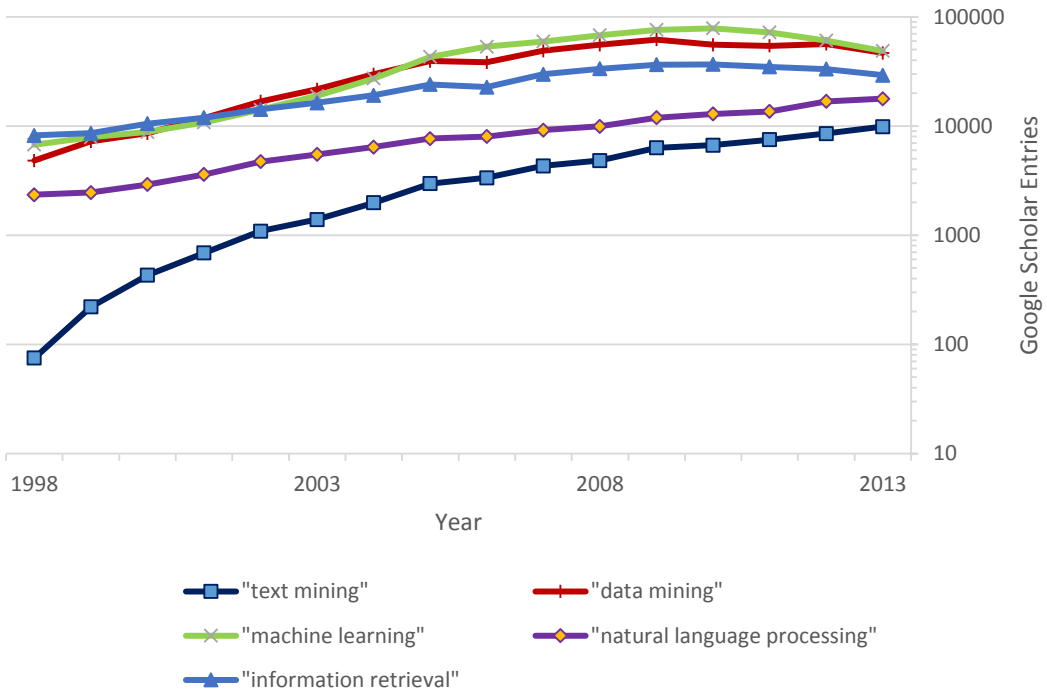
TM has propagated into a wide variety of domains both in academia and business. In academia, TM is enabling new forms of research by providing a methodology for meta-research of large quantities of publications [Jensen et al., 2006], as well as providing computational methods for fields that have lacked quantitative methodologies [Schreibman et al., 2008]. Aside from the sheer volume shown in Figure 2.2, the diversity of TM applications is challenging and lacks a clear-cut categorization. Surveys in TM have suggested different application areas over the years. Black [2006] cites business intelligence and biomedical TM as the main applications. Feldman and Sanger [2006] give corporate finance, patent research, and life sciences as the most successful applications. Fan et al. [2006] categorizes applications into medicine, business, government, and education. Weiss et al. [2012] gives a number of case studies: web market intelligence, digital libraries, help desk applications, news article categorization, email filtering, search engines, named entity extraction, and customized newspapers. Overall, the general categories of biomedical TM and business intelligence capture the two most established application domains of TM.

Biomedical TM was proposed in the context of information retrieval as *"the discovery of hidden connections in the scientific literature"* [Swanson,

[1] scholar.google.com
[2] sciencedirect.com



(a) ScienceDirect



(b) Google Scholar

Figure 2.2: Number of academic publications per year indexed by ScienceDirect and Google Scholar with the phrases “text mining”, “machine learning”, “data mining”, “natural language processing”, and “information retrieval”, retrieved 2.8.2014. Both indices show TM research currently as active as the related fields were around the year 2000

1988, 1991, Hearst, 1999]. Biomedical TM has become an exceedingly popular application, since TM has provided a meta-analysis methodology to discover new facts by combining evidence from the vast biomedical research literature [Cohen and Hersh, 2005, Jensen et al., 2006, Rzhetsky et al., 2008, Zhou et al., 2010, Korhonen et al., 2012, Van Landeghem et al., 2013, Shemilt et al., 2013]. The overall output of scientific publications has increased exponentially for the last century, with some current estimates of annual growth at 4.73% [Larsen and von Ins, 2010] and $8 - 9\%$ [Bornmann and Mutz, 2014]. In 2014, the biomedical article index PubMed[3] contains entries for over 24 million publications, and this exponentially growing literature can only be comprehended using new methods. The popularity of biomedical TM has lead to a common perception of TM as simply biomedical literature mining.

The origins of TM in business intelligence can be traced to a 1958 IBM paper [Luhn, 1958], that proposed an automated system for managing information in documents. Business TM applications are diverse, including financial TM [Kloptchenko et al., 2004, Groß-Klußmann and Hautsch, 2011], marketing [Decker and Trusov, 2010, Pehlivan et al., 2011], and market intelligence [Archak et al., 2007, Godbole and Roy, 2008, Baumgartner et al., 2009, Pehlivan et al., 2011, Ghose and Ipeirotis, 2011, Archak et al., 2011, Netzer et al., 2012]. Perhaps the most common business use of TM is sentiment analysis, and more generally opinion mining [Dave et al., 2003, Pang and Lee, 2008], that seeks to analyze text data in order to monitor opinions related to companies, brands and products. Some business TM publications are starting to use the term text analytics as a synonym for TM [Gruhl et al., 2004, Groß-Klußmann and Hautsch, 2011, Basole et al., 2013], following the trend in the industry where the term "analytics" has become increasingly common over the past decade.

Outside these two major groups of applications, there are application domains that have recently adopted TM methodology. Domains such as law [Coscia and Rios, 2012], political science [Grimmer and Stewart, 2013], humanities [Schreibman et al., 2008], social science [Brier and Hopp, 2011] and intelligence [Maciolek and Dobrowolski, 2013] have combined TM methodology with traditional research methodologies. Many of these domains are applying TM to online data sources such as blogs and micro-blogs, but some use TM on digitized publications. The "soft sciences" such as humanities and social science in particular see TM as providing a new methodology for performing quantitative research on issues that previously relied only on qualitative methods [Schreibman et al., 2008].

[3]http://www.ncbi.nlm.nih.gov/pubmed

The extreme interdisciplinarity and variety makes assessing the overall scope of TM in great detail daunting, due to differences in: 1) terminologies, 2) methodologies, and 3) traditions of publication. Firstly, a large portion of TM research occurs under related terms, such as text analytics, opinion mining, etc. The different fields use their own terminology in addition to TM terms. This makes it difficult to find the relevant publications, and to synthesize a coherent picture from manuscripts written to address different issues using different terms. Secondly, TM is often mixed with the methodologies of the application domain. Comparing TM research often requires expertise of the theory and methodologies of both TM and the application domain. Thirdly, academic communities have varying traditions on publishing: computer scientists publish foremost in conferences, whereas humanities publish in the form of books, while most other disciplines prefer journal publication. A TM book written in the context of digital humanities could prove influential, yet lack the impact factors used in the natural sciences. This complicates assessing the quality of TM publications from external indicators such as citation counts and impact factors. A further complication is grey literature: influential discourse on TM takes place not only in established and peer-reviewed academic contexts, but also in contexts such as non-reviewed articles[4], blogs[5], and white papers[6].

2.2 Text Mining Methodology

2.2.1 Text Documents as Multiply Structured Data

Text data is commonly described as "unstructured data". This phrase originated in the earliest data mining enquiries into text, and has since been used in almost every description of TM. From a data mining point of view, raw text data is not organized into a database of numeric values, hence it can be considered to be unstructured. The purpose of TM was to convert text into a structured form, where data mining could be applied [Ahonen et al., 1997a, Tan, 1999]. The unstructured data description provides a simple introduction to TM from a data mining perspective, but is unfortunately misleading.

Text is more accurately called multiply structured data. The English word "text" comes from the Latin etymology "textus" meaning "woven", related

[4]http://www.nature.com/news/trouble-at-the-text-mine-1.10184
[5]http://breakthroughanalysis.com/
[6]http://hurwitz.com/index.php/component/content/article/394

Figure 2.3: Document structure in an example Wikipedia hypertext document

to the words "textile" and "texture". Not only is language structured, but it occurs in *documents* and *collections* that have additional varying structure. Written text can be understood as sequential statements organized in hierarchies of structures such as sentences, passages, sections, chapters, and so forth. Explicit structure in the form of metadata is virtually always available, whereas the "unstructured data" of human language has implicit structure, arguably among the most complex phenomena to have evolved in nature. The orthographic and typographic structure of written text is further complicated by the structured document mark-ups used in digital text, including hyperlinks that turn text data into hypertextual data better understood in terms of graphs.

A collection of text data consists of documents that can number in millions or more. The collections can be static, with no time component, or dynamic, with documents ordered by time. The collections can be fixed datasets, or streams that are not retained in memory, but processed in an online manner. Collection and document metadata is typically very rich in TM applications, and can include internal and external hyperlink structures of the documents, locations and languages of the documents, author identities, years and dates of authorship, subcategories and ontologies of the documents, etc. The metadata can be unique to the dataset, or highly standardized [Bargmeyer and Gillman, 2000]. Additional explicit metadata can be constructed by applying TM and

ML methods on the dataset [Pierre, 2002]. The document content is organized into fields such as titles, text sections and possibly link sections, and often contains non-text and partly textual media such as figures, illustrations, tables and multi-media. The layout, typography and mark-ups define the visual look of text and hyperlinks connect the text within the document, to other documents, and to resources outside the collection. Figure 2.3 illustrates document structure from the beginning of a Wikipedia document.

A collection used for a specific task has an associated *domain* of background knowledge [Anand et al., 1995, Feldman and Sanger, 2006]. Feldman and Sanger [2006] define domains in TM loosely as: *"a specialized area of interest for which formal ontologies, lexicons, and taxonomies of information may be created"*. The availability and usefulness of domain knowledge is one of the defining properties of TM [Feldman and Sanger, 2006]. The domains depend on the use of the collection, and knowledge from more than one domain can be beneficial. For example, a collection of Twitter microblog messages and newspaper articles used for monitoring a company's public image could benefit from having domain knowledge for spelling correction, normalization, and named entity recognition, for both types of data. The simplest form of domain knowledge is text collections of billions of words in the domain language that can be used to construct models for TM [Napoles et al., 2012, Buck et al., 2014]. Natural language consists of languages, dialects, ethnolects and sociolects. It is common that TM collections and domains only cover a particular subset of one language.

The actual text content of a text document consists of sequential information in the form of natural language. From the last century of linguistics research into the structure of natural language, it has been established that language consists of structures at various levels. Starting from the highest level, these are discourse, pragmatics, semantics, syntax, morphology, phonology and phonetics [Jurafsky and Martin, 2008]. Linguistics itself has divided into subfields that each specialize in one of these levels. Discourse deals with topics and discussions, pragmatics with contextual meanings and interpretations, and semantics with the meanings of linguistic constructions. Syntax deals with the generation of sentences from words, and morphology with the generation of words from morphemes. Phonology and phonetics deal with phonemes and phones, the atomic units of speech. Figure 2.4 shows the structure at the syntactic level for the definition of TM used in this thesis, as provided by the NCLT wide-coverage parser[7] [Cahill et al., 2004].

[7]http://lfg-demo.computing.dcu.ie/lfgparser.html

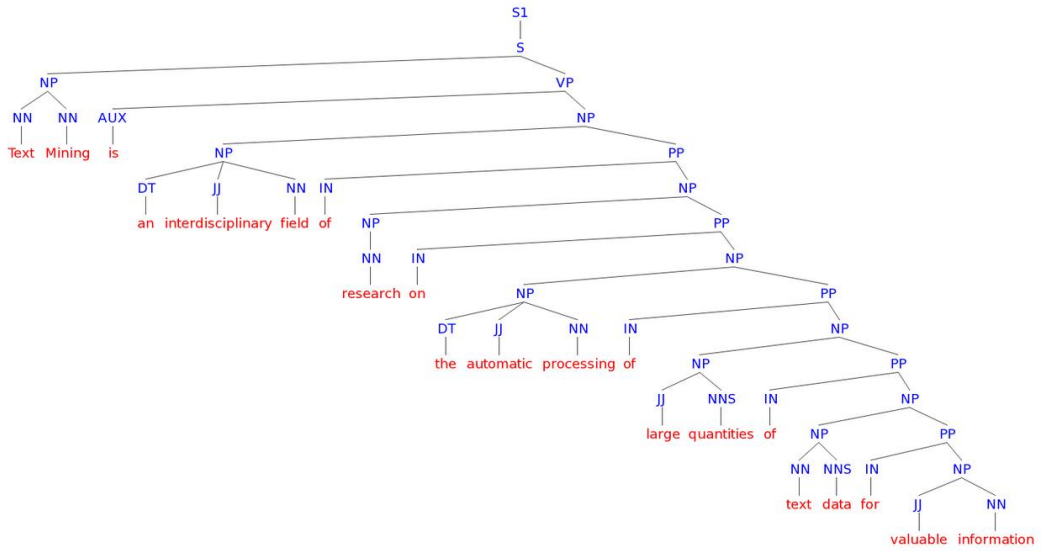


Figure 2.4: Linguistic structure at the syntactic level for the TM definition used in this thesis, according to the NCLT wide-coverage parser [Cahill et al., 2004]

The lowest levels of language are easiest to describe formally, and syntax and the higher levels were considered too complex for formal descriptions until the advent of computational linguistics. While syntax and the lower levels have clearly defined elementary units such as words and morphemes, no consensus exists for elementary units in the higher levels of language. The levels are not strictly separated: morphosyntax, morphophonology, and morphosemantics study some of the interactions between these levels. The levels are neither strictly hierarchical, but parallel. For example, within a word the morphological and syllabic segments commonly overlap: the word "rated" has the morphological boundaries "rate + d", and the syllable boundaries "ra - ted". Text and speech analysis thus commonly annotates linguistic data with overlapping description tiers.

A further complication in natural language is ambiguity in the various levels. Ambiguous structures are very common in natural language. A common example of ambiguity is the sentence *"Time flies like an arrow; fruit flies like a banana"* [Burck, 1965]. The words "flies" and "like" are both used ambiguously, "flies" as a verb in the first clause and as a noun the second, "like" as an adverb in the first and as a verb in the second. The sentence is also a "garden path sentence", since reading it forces the reader to disambiguate the word "flies" in the first clause, only to realize that "time flies" is not used as a noun

phrase, and that the two clauses must be non-related. Resolving ambiguities requires understanding of the larger context, but generally provides higher efficiency to language as a form of communication [Piantadosi et al., 2012].

2.2.2 Structured Representations for Text

The view of text as unstructured data is misleading, but it considerably simplifies the overwhelming complexity of processing text documents. While human readers can easily understand most types of text documents, the simplest types are undecipherable for general computer algorithms. It is therefore necessary to use simplified representations of text to perform any processing of text that is natural for humans. TM uses different representations of text that depend on the use, also called intermediate forms [Tan, 1999] or representational models [Feldman and Sanger, 2006]. In the following discussion, formal notation is introduced that will be extensively used in the later chapters of the thesis.

NLP commonly uses text processed into some type of *normalized form*. This is a form of text with all non-linguistic elements removed or separated from the text content, and the text is encoded using a standard such as ASCII or Unicode. The text can then be further normalized to remove unwanted variance [Zhu et al., 2007, Demuynck et al., 2009, Yang and Eisenstein, 2013]. A common normalization is expansion of abbreviations and number words. Another common normalization is spelling correction. The normalized form depends on the intended use. For modeling text as word sequences using n-gram models, modeling effort can be reduced by removing sentence-initial capitalization and punctuation, and placing sentence boundary tokens. Alternatively, the text can be normalized to recover the capitalization and punctuation instead, if the original text is missing these. A *word sequence* variable representing a normalized document of J words can be formally expressed as $\underline{\boldsymbol{w}}$, where each integer variable $\underline{w}_j : 1 \leq j \leq J$ in the sequence indexes a word in a dictionary of N possible words. Normally the dictionary size N doesn't need to be defined. The dictionary can be easily updated by maintaining a hash table, and mapping each previously unseen word to the integer value $N + 1$.

The normalized word sequences can be further processed according to the intended use. For uses such as text classification, clustering and retrieval, there exists a set of common normalizations: stemming, stopwording and short word removal. Stemming removes word endings, so that for example the words "connect, connected, connecting, connection, connections" are all mapped to the

Table 2.1: Examples of word sequence and word vector representations for two text snippets from Wikipedia as documents. For illustrating sparsity, zero counts are not shown for word vectors. Compared to word sequences, word vector variables grow in dimensionality and become increasingly sparse with more documents encoded by the dictionary

(a) Normalized text form and corresponding word sequence representations with integer encoding

document 1	$\underline{w}_j^{(1)}$	document 2	$\underline{w}_j^{(2)}$
the	1	the	1
book	2	bachman	11
was	3	books	12
released	4	is	13
in	5	still	14
1985	6	in	5
after	7	print	15
the	1	in	5
publication	8	the	1
of	9	united	16
the	1	kingdom	17
first	10	although	18

(b) Dictionary of integer encodings and word vector representations

n	word	$w_n^{(1)}$	$w_n^{(2)}$
1	the	3	2
2	book	1	
3	was	1	
4	released	1	
5	in	1	1
6	1985	1	
7	after	1	
8	publication	1	
9	of	1	
10	first	1	
11	bachman		1
12	books		1
13	is		1
14	still		1
15	print		1
16	united		1
17	kingdom		1
18	although		1

same word "connect" [Lovins, 1968, Porter, 1980]. This reduces variability by performing a heuristic clustering of the words. Short words of less than three characters are removed, along with words that occur in a stop-word list: a list of usually around 1000 common words that are not useful for the task. These linguistic processing methods depend on the language, and in most languages advanced morphological processing is required [Kurimo et al., 2010, Zhao and Liu, 2010]. The linguistic processing can also enrich the words with tagging information, such as part-of-speech and dependency tags.

The normalized sequence forms used in NLP are insufficient for the common ML and DM methods that rely on data organized into vector forms. The majority of TM applications use a *feature vector* representation of text documents originating from information retrieval [Salton, 1963, Salton et al., 1975]. The most basic type of a feature vector for a document is the "bag-of-words", where a feature vector $\boldsymbol{w}$ consists of the counts of each word in the document w_n, where n indexes the dictionary of N words. The norms of a vector

are used to measure document length. L1-norm is the length of the word sequence: $|\boldsymbol{w}^{(i)}|_1 = J^{(i)} = \sum_n w_n^{(i)}$, whereas the "L0-norm" is the number of non-zero counts: $|\boldsymbol{w}^{(i)}|_0 = \sum_n \min(1, w_n^{(i)})$. Sparsity of the vector equals the proportion of non-zero counts: $|\boldsymbol{w}^{(i)}|_0/N$. More complex feature vectors differ from the bag-of-words in how the features are chosen, and how the counts or weights of each feature are computed. Table 2.1 shows a comparison of word sequences and word vector representations using integer counts.

Earlier text mining research believed that simple weighted words are not easily outperformed for most tasks [Salton and Buckley, 1988, Sebastiani, 2002]. Possible alternative features include word pairs [Lesk, 1969], linguistically tagged words [Dave et al., 2003, Gamon, 2004], factor concepts and topics [Borko and Bernick, 1963, 1964, Deerwester, 1988, Hofmann, 1999, Blei et al., 2003], phrases [Salton, 1988] and parse trees [Chubak and Rafiei, 2012]. Some applications such as authorship detection and essay scoring rely on non-typical features such as document length [Larkey, 1998, Madigan et al., 2005]. More recently, word sequence features[8] have become a crucial part in some text classification tasks [Dave et al., 2003, Gamon, 2004, Xia et al., 2011, Lui, 2012, Tsoumakas et al., 2013]. The recent results finding considerable improvements from combining other features with word vectors are due to the availability of more data, and advanced models for combining the feature sets.

For some uses binary weights $\forall n : w_n \in 0, 1$ are sufficient. For most uses it is beneficial to weight the features so that the words relevant for the modeling purpose are weighted higher. This results in non-negative fractional counts $\forall n : w_n \in \mathbb{R} \wedge w_n \geq 0$. With word features, the weighting functions typically dampen high count values, normalize the counts for varying document lengths, and weight the words according to rarity in the collection. A variety of possible weighting functions exist for choosing the weights, the most common being Term Frequency - Inverse Document Frequency (TF-IDF) [Salton and Buckley, 1988]. When weighting functions are used, the transformed counts can be denoted $\boldsymbol{w}$, whereas the original counts can be denoted $\boldsymbol{w}'$.

A collection of I documents can be formalized as a set D, where the document variable $D^{(i)}$ for each document identifier i consists of the structured variables used to represent the document. With word vectors for representation and no label information or other metadata, the document variable consists of the word vector: $D^{(i)} = (\boldsymbol{w}^{(i)})$. With word sequences and label variables $l^{(i)} : 1 \leq l^{(i)} \leq L$ of L possible labels, the document variables would

[8] called n-grams in publications, but these are word sequence features

word	n	postings list
the	1	(1, 3) (2, 2)
book	2	(1,1)
was	3	(1, 1)
released	4	(1, 1)
in	5	(1, 1) (2, 1)
1985	6	(1, 1)
after	7	(1, 1)
publication	8	(1, 1)
of	9	(1, 1)
first	10	(1, 1)
bachman	11	(2, 1)
books	12	(2, 1)
is	13	(2, 1)
still	14	(2, 1)
print	15	(2, 1)
united	16	(2, 1)
kingdom	17	(2, 1)
although	18	(2, 1)

Table 2.2: Inverted index representation for the documents shown in Table 2.1. The postings lists are non-positional and unweighted, containing only the document identifiers and word counts contained in the document word vectors

be $D^{(i)} = (l^{(i)}, \underline{\boldsymbol{w}}^{(i)})$.

The dictionary size N for word vectors in a collection of millions of documents could typically be in the hundreds of thousands. Out of the possible words, typically only some tens or hundreds of words occur in a document. This means that the word vectors are extremely sparse, and both the dimensionality and sparsity increases as larger collections are processed. If counts are accumulated from all documents corresponding to a label, the label-conditional counts are almost as sparse. Like word vectors, most useful representations of text are *high-dimensional sparse data*.

Representing a collection of high-dimensional sparse data can be done with an inverted index [Zobel and Moffat, 2006], enabling scalable retrieval of documents as well as other types of inference [Yang, 1994, Shanks et al., 2003, Kudo and Matsumoto, 2003, Puurula, 2012a]. The scalability of modern web search engines is largely due to the representation of web pages using inverted indices [Witten et al., 1994, Zobel and Moffat, 2006]. An inverted index stores a document collection as a table of dictionary words or *terms*, and a *postings list* of document occurrences of the term n. Table 2.2.2 illustrates an inverted index representation for the example documents shown in Table 2.1. The in-

verted index representation is highly efficient, since the term occurrences are sparse and zero counts do not need to be considered when constructing, storing or using the index. Normally a posting contains a document identifier and the number of occurrences of the term in the document. Position information is sometimes included in the postings, for ranking functions that benefit from proximity information. The postings lists are commonly compressed for additional storage savings, and methods for further improving the efficiency of indices constitute an extensive literature [Witten et al., 1994, Zobel and Moffat, 2006]. Use of inverted indices can be described as a type of sparse matrix computation applied to text, although this view is not ordinarily taken in IR.

2.2.3 Text Mining Applications as Machine Learning Tasks

TM is applied in numerous ways across application domains. One possible way to categorize the applications is to use ML terminology and consider the underlying learning problem that is solved in each application [Feldman and Sanger, 2006, Aggarwal and Zhai, 2012]. This makes it possible to compare solutions used in different applications, and attempt solutions used for non-text data of the same task type. The basic framework of ML is described next, followed by a mapping of many common TM tasks into ML problems.

A commonly accepted definition of ML is: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."* [Mitchell, 1997]. For an example application of e-mail spam filtering, the experience E could be a training dataset of spam/non-spam e-mail examples, the task T could be the binary classification of new examples into spam/non-spam, and the measure P could be the percentage of correctly classified e-mails.

We can consider document collections D a type of dataset for ML algorithms. The first division in learning problems is between inductive and transductive learning [Gammerman et al., 1998, Joachims, 1999]. Given a training dataset D, inductive learning attempts to learn a function for general prediction, usually for making predictions on unseen data. Transductive learning does not attempt to learn a general function, but rather attempts to transfer properties found in a training dataset D to a test dataset D^*. The transduced properties are normally the label information available for dataset D, but not for D^*. Transduction improves prediction quality, but the solutions will be

optimized only for the used test set.

A second division in learning problems is between supervised, semisupervised and unsupervised learning. In supervised learning, a training dataset is provided with label variables, whereas in unsupervised learning no label variables are provided. Supervised learning is considerably easier than unsupervised learning, as the label variables are usually reliable information for learning the function of interest. Unsupervised learning has to constrain the learning problem to compensate for the lack of label information. For example, in a text clustering task the number of clusters is often fixed and prior distributions can be used to guide the learning towards a more plausible clustering. The corresponding supervised task of text classification would have the label variables provided, so that the number of labels and their assignments to documents would not need to be learned, unlike in the unsupervised case. In semisupervised learning some portion of the label variables are provided. Semisupervised learning is very common in TM tasks, for example in many text classification applications there are a small number of labeled documents compared to large quantities of unlabeled documents.

The type of variables to be predicted divides ML problems further. Classification deals with the prediction of discrete label variables. Ranking deals with the ordering of discrete label variables. Regression deals with prediction of continuous label variables. The predicted variables can be structured. Binary classification deals with binary label variables $l \in 0, 1$, multi-class classification with categorical label variables $l : 1 \leq l \leq L$, multi-label classification with label vectors $\boldsymbol{c} = [c_1, ..., c_L]$ of binary variables [Tsoumakas et al., 2010] and multi-dimensional classification with label vectors of categorical variables [Bielza et al., 2011]. Corresponding divisions exist for ranking and regression problems, as well as multi-output or multi-target prediction problems that contain mixed output variables.

More complex structured prediction problems arise when the input variable is not a simple vector. Sequence labeling classifies sequence variables into corresponding structured variables. For example, syntactic parsers map word sequences into parse trees, and speech recognition maps sequences of acoustic feature vectors into word sequences. Solutions for structurally complex problems often have a number of uses, and produce models that provide information about the learning problem. Multi-task learning attempts to solve different tasks at the same time, taking advantage of the related optimization problems to find better solutions [Caruana, 1993].

Table 2.3: Mappings of TM applications to ML tasks

Application	Task	Publication
sentiment analysis	binary classification	[Pang et al., 2002]
spam filtering	binary classification	[Medlock, 2006]
email categorization	multi-class classification	[Joachims, 1997]
price prediction	multi-class classification	[Lee et al., 2014]
news categorization	multi-label classification	[Lewis et al., 2004]
document retrieval	ranking	[Metzler and Croft, 2007]
sales prediction	regression	[Archak et al., 2007]
essay grading	regression	[Larkey, 1998]
patent mapping	clustering	[Fattori et al., 2003]
event detection	clustering	[Allan et al., 1998]
entity recognition	sequence labeling	[McCallum and Li, 2003]

Table 2.3 shows a sample of common TM applications and their mapping into ML tasks. An application can be mapped into a ML task in several different ways. For example text regression problems [Archak et al., 2007, Joshi et al., 2010, Wang et al., 2010a, Archak et al., 2011, Ghose and Ipeirotis, 2011, Cao et al., 2011, Higgins et al., 2014, Lee et al., 2014] such as stock price prediction can often be solved using regression or classification. Commonly, the main improvements in ML come from defining the task well and choosing the features useful for that task, rather than the choice of learning algorithms. Multiple ways of approaching a problem can work, and the combination of different types of solutions is highly beneficial. Complex learning approaches are not necessary for applying the ML framework: classifiers such as K-nearest Neighbours [Cover and Hart, 1967] and Naive Bayes (NB) [Maron, 1961] can operate on the basis of counted training dataset statistics, without the use of iteratively learned parameters.

The performance measures P depend on the application and task. For general classification tasks, accuracy is defined as the percentage of correctly classified instances for a given test set. This might be ill-suited, if the use of the classification system is to find some relevant documents for each possible label. Extensively studied tasks have highly specialized measures that attempt to quantify the usefulness of the ML system in the applications, such as NDCG that is used to measure ranking performance in web search engines [Järvelin and Kekäläinen, 2002].

The ML framework involves segmentation of datasets into training and test portions, so that the performance is not measured on the same data that is used to learn parameters. The most typical split is between a training portion for

learning parameters, a development portion for calibrating meta-parameters of the algorithms, and a final test portion that is used for evaluation. Alternatively, cross-validation segments a dataset into a small number of exclusive training and development portions, and the performance measure can be averaged across the folds. More complex solutions can have a number of nested dataset segmentations, reserving unused testing data to optimize each layer in a system.

2.2.4 Linear Models as Methods for Text Mining

Mapping TM applications into established tasks enables the use of existing methods for solving problems. Earlier methods for TM relied on linguistic methods for performing text preprocessing, and algebraic methods for performing text retrieval. The more recent methods for TM are algorithmic, often model-based, and predominantly originate in statistics and ML. The new algorithmic methods based on statistics and learning have brought a paradigm shift in the field of artificial intelligence, and there remain few areas of TM where solutions based solely on domain expert-knowledge are preferred. Commonly, domain knowledge such as stemmers and sentiment lexicons are used as additional information for learning algorithms.

Most algorithms on word vectors and related representations are applications of linear models, that perform predictions using linear combinations of feature values weighted by learned parameters. The tasks that are commonly solved using linear models include regression, classification, ranking and clustering. In text regression regularized linear regression can be applied [Archak et al., 2007, Joshi et al., 2010, Higgins et al., 2014]. In text classification classifiers such as Centroid [Rocchio, 1971, Han and Karypis, 2000], Bernoulli Naive Bayes (BNB) [Maron, 1961], Logistic Regression (LR) and Support Vector Machines (SVM) [Joachims, 1998, Fan et al., 2008] are linear models. In text ranking and text retrieval, all of the common scoring functions are linear models, including the Vector Space Model (VSM) [Salton et al., 1975], language models [Kalt, 1996, Hiemstra and Kraaij, 1998], as well as more recent discriminative ranking models [Metzler and Croft, 2007]. Text clustering commonly uses linear models, such as multinomial and Cosine distances [Pavlov et al., 2004, Zhong and Ghosh, 2005, Banerjee et al., 2005, Rigouste et al., 2007]. The following presents a succinct overview of the linear model framework for TM.

A basic type of statistical model for solving modeling problems is the linear

regression model, as used for solving text regression problems. Let us assume word vector features $\boldsymbol{w}$, with a dictionary of N words. Let $\boldsymbol{\theta}$ denote a parameter vector of weights for the regression model, where θ_0 is called the "bias" parameter and θ_n for values $1 \leq n \leq N$ are the regression weights for each word feature n[9]. A linear regression predicting *scores* $y(\boldsymbol{\theta}, \boldsymbol{w})$ of a predicted continuous variable y takes the form:

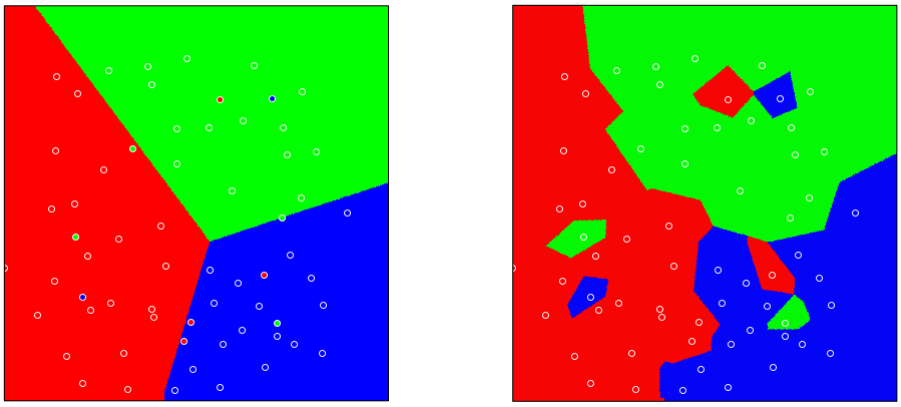
$$y(\boldsymbol{\theta}, \boldsymbol{w}) = \theta_0 + \sum_{n=1}^{N} \theta_n w_n \tag{2.1}$$

The weights θ_n decide how much the predictors w_n explain the observed variations of y seen in a training dataset, where y and $\boldsymbol{w}$ are available, and $\boldsymbol{\theta}$ needs to be estimated. A basic way to estimate the parameters $\boldsymbol{\theta}$ is the method of least squares, so that the sum-of-squares error function over the dataset $E_D(\boldsymbol{\theta}) = 1/2 \sum_i (y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{w}^{(i)})^2$ is minimized. This is equivalent to maximum likelihood estimation assuming that the errors are generated by a Gaussian noise model.

Most applications of linear models use regularization to control overfitting [Frank and Friedman, 1993]. This adds a regularization term $R(\boldsymbol{\theta})$ to the error function, so that the total error function becomes $E'_D(\boldsymbol{\theta}) = E_D(\boldsymbol{\theta}) + \Lambda R(\boldsymbol{\theta})$, where Λ is a weight for the regularization. Regularization often takes the form $1/2 \sum_n |\theta_n|^q$, where q is the L-norm of the regularizer. A common case $q = 1$ is called the Lasso regularizer, and the case $q = 2$ is the ridge or Tikhonov regularization. Regularization causes parameter estimates to shrink towards more conservative values, to zero in the case of the sparsity-inducing Lasso regularizer.

Linear models for classification and ranking apply a further decision rule on the scores $y(\boldsymbol{\theta}, \boldsymbol{w})$ to map the scores into categories and rankings. Binary classification maps the scores based on the sign of the score: with a classification threshold of 0, if $y(\boldsymbol{\theta}, \boldsymbol{w}) \geq 0$, $l = 1$, else $l = 2$. Multi-class classification involves label-dependent parameter vectors $\boldsymbol{\theta}_l$, and maps the scores by maximizing the score: $\text{argmax}_l(y(\boldsymbol{\theta}_l, \boldsymbol{w}))$. Ranking sorts the scores for the parameter vectors, and maps the order of labels into ranks [Metzler and Croft, 2007]. The multi-class linear scoring function can be expressed as:

[9]Using $\boldsymbol{w}$ for the weight vector is the common notation with regression models [Bishop, 2006]. The different notation $\boldsymbol{\theta}$ is used here to keep the notation consistent throughout the thesis.



(a) Linear classifier: L1-regularized Logistic Regression

(b) Non-linear classifier: k=1 Nearest Neighbour

Figure 2.5: Visualization of decision boundaries of a linear and non-linear classifier on 2-dimensional 3-class data. Logistic Regression forms linear decision boundaries, K-Nearest Neighbours forms non-linear boundaries

$$y(\boldsymbol{\theta}_l, \boldsymbol{w}) = \theta_{l0} + \sum_{n=1}^{N} \theta_{ln} w_n \tag{2.2}$$

This elementary function covers a swath of modeling approaches, with highly different semantics for the bias parameters θ_{l0} and the label-dependent parameters θ_{ln}. In classification, models that can be expressed in the form of Equation 2.2 are called linear classifiers, since they form linear decision boundaries as a function of the feature vectors. Figure 2.5 illustrates the decision boundaries of a linear classifier compared to a non-linear classifier. With probabilistic approaches, the scores are related to the posterior probability of the label given the data through a link function. For example, with exponential-family models such as LR, BNB and Multinomial Naive Bayes (MNB), the posterior probabilities are $p(l|\boldsymbol{w}) = \exp(y(\boldsymbol{\theta}_l, \boldsymbol{w}))$. With non-probabilistic approaches such as SVM, the scores are optimized entirely for classification and do not represent probabilities.

Estimating the linear model parameters depends on the models and the strategies used to reduce overfitting. With algebraic methods such as the Centroid Classifier and generative probabilistic models such as MNB, the label-dependent parameters θ_{ln} are estimated for each label independently, while the bias parameters θ_{l0} are assumed uniform or estimated separately. With gen-

Table 2.4: Parameter estimates for the common linear models used in TM. VSM, BMB and MNB are used for ranking, classification, and clustering, BM25 is used for ranking and regularized LR/SVM for classification. BNB and MNB show unsmoothed parameter estimates. For BM25 a unique label exists for each document: $l^{(i)} = i$, and for LR/SVM the labels are binary: $l^{(i)} \in (-1, 1)$. $IDF(n)$ and $LN(i)$ for BM25 refer to the chosen IDF and length normalization functions, respectively. $R(\boldsymbol{\theta})$ and $L(\boldsymbol{\theta}, D^{(i)})$ for LR/SVM refer to the chosen regularization and loss functions, respectively.

Model	Parameters $\boldsymbol{\theta}$	Test w_n
VSM [Rocchio, 1971]	$\theta_{ln} = \frac{\sum_{i:l^i=l} w_n^{(i)}}{\sqrt{\sum_{n'}^{N} (\sum_{i:l^i=l} w_{n'}^{(i)})^2}}$	$\frac{w'_n}{\lvert\boldsymbol{w}'\rvert_2}$
BNB [Maron, 1961]	$\theta_{ln} = \log \sum_{i:l^i=l} \frac{\min(1, w_n^{(i)})}{I}$	$\min(1, w_n'^{(i)})$
MNB [Kalt, 1996]	$\theta_{ln} = \log \frac{\sum_{i:l^i=l} w_n^{(i)}}{\sum_i \sum_n w_n^{(i)}}$	w'_n
BM25 [Manning et al., 2008]	$\theta_{ln} = IDF(n) \frac{(k_1+1)LN(i)}{(k_1)LN(i)}$	$\frac{(k_3+1)w'_n}{k_3+w'_n}$
LR/SVM [Fan et al., 2008]	$\min_{\boldsymbol{\theta}} R(\boldsymbol{\theta}) + C \sum_i L(\boldsymbol{\theta}, D^{(i)})$	w'_n

erative models, $\theta_{ln} = \log(p_l(n))$ are label-conditional log-probabilities, $\theta_{l0} = \log(p(l))$ are label prior log-probabilities, and both types of parameters can be smoothed and scaled to correct for overfitting. With discriminative classifiers such as LR and SVM, the parameters are estimated by minimizing a regularized error function [Fan et al., 2008], similarly to learning the regularized linear regression.

Table 2.4 summarizes the parameter estimates for the commonly used linear models in TM. For BM25, the Croft-Harper IDF function is often used: $IDF(n) = \log \frac{I - I_n + 0.5}{I_n + 0.5}$ [Manning et al., 2008], where I_n is the number of documents where the word n occurs. Another common IDF function with BM25 is $IDF(n) = \log \frac{I+1}{I_n+0.5}$ [Fang et al., 2004]. The soft length normalization for BM25 is given by $LN(i) = w_n^{(i=l)} / (1 - b + b(|\boldsymbol{w}^{(i=l)}|_1 / A))$, where the average document length A is $\sum_i |\boldsymbol{w}^{(i)}|_1 / I$. The loss function $L(\boldsymbol{\theta}, D^{(i)})$ for SVM/LR is $\log(1 + \epsilon^{-l^{(i)} \boldsymbol{\theta}^T \boldsymbol{w}^{(i)}})$ for LR, $\max(0, 1 - l^{(i)} \boldsymbol{\theta}^T \boldsymbol{w}^{(i)})$ for L1-loss SVM and $\max(0, 1 - l^{(i)} \boldsymbol{\theta}^T \boldsymbol{w}^{(i)})^2$ for L2-loss SVM [Fan et al., 2008]. The regularization $R(\boldsymbol{\theta})$ for SVM/LR is $\frac{1}{2} |\boldsymbol{\theta}|_2^2$ for L2 regularization and $|\boldsymbol{\theta}|_1$ for L1 regularization. BM25 requires the meta-parameters k_1, k_3 and b, LR/SVM requires the meta-parameter C for regularization. Use of feature transforms and smoothing for VSM, BNB, and MNB modifies the equations in Table 2.4, and introduces additional meta-parameters.

Inference for different uses with parameters in the form of Equation 2.2 is a trivial summation and maximization for classification, and summation and

sorting for ranking. The algorithms used for estimation differ widely. For the Centroid Classifier and MNB, the estimation is a simple closed-form linear-complexity procedure of summing, normalizing and possibly smoothing the word count statistics. For LR and SVM, the algorithms depend on the error function and regularization [Yuan et al., 2012]. Many of the practical models have the property of a convex error function, so that the estimation can be performed using efficient Gradient Descent algorithms, including the online version Stochastic Gradient Descent [Bottou, 2010, Duchi et al., 2011, Bottou, 2012]. Stochastic Gradient Descent is applied on a large class of models, including L2 and L1-regularized LR [Carpenter, 2008, Tsuruoka et al., 2009], and often outperforms methods tailored for the particular problem.

Linear models can be extended in a number of ways to represent non-linear decision surfaces [Keysers et al., 2003, Bishop, 2006, Chang et al., 2010]. The simplest way is mapping the original feature vectors to transformed ones, examples of which are factor decompositions [Borko and Bernick, 1964, Blei et al., 2003], word pair features [Lesk, 1969], and explicit polynomial mappings [Chang et al., 2010]. Generalized linear models apply an implicit link function to transform different types of prediction tasks into linear regression modeling problems, LR using the logit function being one example. Other types of generalized linear models are not necessarily linear models in the sense of linear classifiers and the definition of Equation 2.2. Replacing the multinomial event model in MNB with a Gaussian model would likewise result in non-linear boundaries.

A second type of extension into non-linear decision boundaries is utilizing information present in individual documents of the collection. K-Nearest Neighbours [Cover and Hart, 1967], Kernel Density Classifiers [Parzen, 1962] and Mixture Models [Li and Yamanishi, 1997] are models that maintain parameters for a set of prototypes for each class, and combine scores for each class from the prototype scores. These models can capture properties of local neighbourhoods in the documents that would be lost with the representation of a single parameter vector. Kernel learning methods [Boser et al., 1992, Joachims, 1998] use feature transformations called kernels into arbitrary spaces that are not explicitly computed, but rather evaluated implicitly by the learning algorithm. This gives kernel learning a great deal of flexibility in learning decision surfaces, but with a computational cost that is not always preferable over a linear kernel maintaining the original feature space [Fan et al., 2008, Yuan et al., 2012].

A third type of non-linear extension is multi-layer methods, such as tree-based learning [Breiman et al., 1984, Quinlan, 1986], neural networks [Rosenblatt, 1958, Widrow, 1960], and ensemble learning [Breiman, 2001b, Friedman, 2002, Sill et al., 2009]. All of these combine layers of elementary base-learner algorithms, often dividing the original documents and feature vectors into different subsets for the base-learners. Tree-based methods combine component learners similar to mixture models, but combine the components using hard decisions based on rules that best segment the data, rather than performing soft combination with fixed mixture weights assigned to each component. Neural networks extend simple base-learners such as LR with hidden layers of learners, with higher modeling flexibility, but also introducing a difficult learning problem that is commonly approached using Stochastic Gradient Descent [Widrow, 1960, Bottou, 2012] combined with heuristics. Ensemble methods combine a set of diverse base-learners to optimize a performance measure, commonly selecting the optimal set of base-learners for the task and learning the best possible combination [Sill et al., 2009, Puurula and Bifet, 2012].

A further extension of linear models in TM is prediction and modeling in tasks that require structured variables, some of which cannot be accurately solved by decomposing the problems into simple linear problems. Examples of such tasks are entity and event detection performed in information extraction, and syntactic tree generation in parsing sentences. However, a majority of the methods used for solving these problems are extensions of basic linear models into structured prediction: Conditional Random Fields [Lafferty et al., 2001] extend LR, Hidden Markov Models [Baum et al., 1970, Kupiec, 1992] extend Naive Bayes, and Max-margin Markov Networks [Taskar et al., 2003] extend SVM. Structured prediction models extending Naive Bayes are described in Chapter 3, and the methods developed in Chapters 4 and 5 can be equally extended into structured modeling.

2.2.5 Text Mining Architectures as KDD Processes

Preprocessing data to structured forms and applying ML to solve tasks forms the basic building blocks of TM. Combining these into complex solutions for TM applications often requires integration of the available components into an *architecture* for the TM application [Feldman and Sanger, 2006, Villalón and Calvo, 2013, Maciolek and Dobrowolski, 2013]. The concept of a TM architecture originates from viewing TM as a case of the KDD process [Feldman and Dagan, 1995, Feldman et al., 1997, Ahonen et al., 1997a].

A basic KDD process is defined as consisting of five steps [Fayyad et al., 1996]: 1) selection, 2) preprocessing, 3) transformation, 4) data mining and 5) interpretation/evaluation. Selection chooses documents and variables of interest for further analysis. Preprocessing consists of modeling noise and missing data. Transformation reduces and transforms the number of considered variables to a form more suited for analysis. Data mining applies algorithms to find interesting patterns. Interpretation/evaluation performs interpretation of the discoveries, possibly visualizing the models or the data using the models. The number of five steps is not fixed, but an example of a possible basic process. All of the steps are interactive, with the user iteratively modifying the steps and cycling through the process to discover more knowledge from the database.

TM was proposed in its earliest forms as KDD with an additional text-specific preprocessing step [Ahonen et al., 1997a, Dörre et al., 1999]. This text preprocessing step consisted of preprocessing each document into a feature vector, and filtering of the feature vector to a form more easily processed by standard DM algorithms. It was also suggested that the filtering step was needed for scalability, as the resulting feature vectors would be exceedingly high dimensional for the usual DM algorithms. Text datasets have since grown thousands of times in all relevant dimensions, and architectural decisions have been suggested to maintain scalability [Villalón and Calvo, 2013, Maciolek and Dobrowolski, 2013].

Current TM architectures can perform the selection step by applying a search engine [Villalón and Calvo, 2013], or applying a web crawler [Maciolek and Dobrowolski, 2013] to retrieve documents related to the TM application. Unlike in typical KDD, the whole collection is therefore not necessarily known or available, but a sample of the vast amount of possible data is gathered in the first step. The preprocessing step can use extensive linguistic processing [Villalón and Calvo, 2013], such as tagging words and phrases according to syntactic roles, identifying named entities and events, and categorizing documents into ontologies. The remaining basic steps of transformation, data mining and interpretation/evaluation largely follow the general KDD process, but with some text specific solutions: transformation can be done with topic modeling [Hofmann, 1999, Blei et al., 2003] instead of general matrix factorization methods, data mining is done with algorithms that operate well on high-dimensional sparse data such as Naive Bayes, and visualization is done with tools such as word clouds [Šilić and Bašić, 2010]. Feldman and Sanger [2006] considers domain knowledge sources as universally important for TM applications and presents extensive use of domain knowledge throughout TM

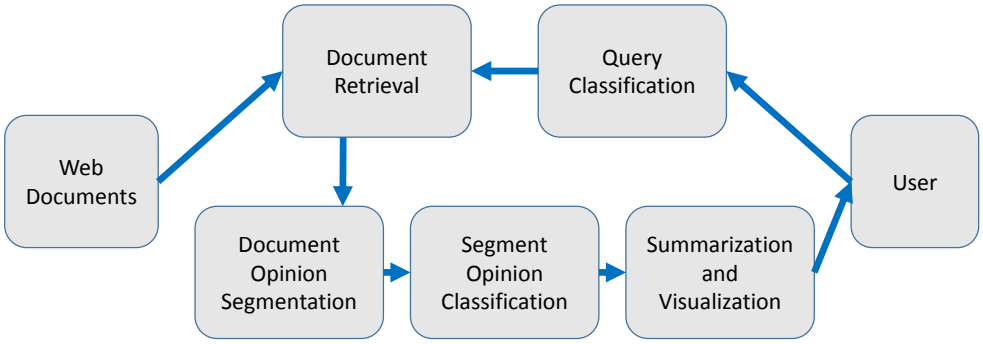


Figure 2.6: Possible TM architecture of an opinion search engine

architectures.

As an example of a TM architecture for an application, a framework presented in a survey of opinion mining [Pang and Lee, 2008] can be illustrated. This divides the construction of an opinion search engine system into four problems: 1) classification of queries into opinion/non-opinion related queries, 2) finding matching documents and segmenting parts of opinionated content, 3) classifying the opinion related to the query in the relevant parts of documents, and 4) presenting the gathered opinion information with summarization and visualization. Figure 2.6 shows a basic TM architecture for this application, decomposed into ML tasks.

The KDD process appears to encompass everything contained in TM, but it can also be too general to describe some TM applications. Many applications of TM involve systems that use the methods of TM such as inverted indices and domain knowledge resources, but do not aim at discovering new types of knowledge, or require an interactive user. Similarly to information extraction, the outcome for TM can be one of known patterns, produced by a fully automatic system. A typical case is text classification applications such as email spam and newspaper topic classification, that produce expected results automatically.

2.3 The Scalability Problem

2.3.1 Scale of Text Data

The information explosion has brought an overwhelming amount of data available to ordinary people and large institutions alike, much of it in the form of

text. A common truism originating from business intelligence research is that 80%-90% of data in corporate databases is in the form of unstructured text [Rajman et al., 1997, Dörre et al., 1999, Godbole and Roy, 2008]. Certainly a large majority of consumed data is in the form of written text such as newspapers, books, blogs, micro-blogs and chat messages. There is more textual data produced electronically in a single day than any individual person can digest in a lifetime, and the rate of production is increasing rapidly.

What has changed is not only the *scale* of data, but its *availability* and *accessability*. For example, the largest library to date is the The British Library, containing 170 million items, closely followed by the Library of Congress with 152 million items[10]. Although considerable in scope, these repositories of data are not so readily accessed as databases existing digitally. The information explosion is not only making vastly larger amounts of data available, but making them rapidly accessible through technologies such as IR and TM.

The available text data is continuously expanding and of vast scale in several ways:

Number of Documents. The number of documents in many datasets and streams is measured in millions, and in some cases billions. The online encyclopedia Wikipedia has 4.5 million articles in English as of 2014[11]. Google Books had digitized 30 million books by 2013[12]. The micro-blog provider Twitter announced in 2013 that its 200 million users were sending 400 million tweets per day[13], more than the global SMS mobile text message traffic combined. The popular social messaging app developer WhatsApp[14] announced in 2014 that its 430 million users send 50 billion messages per day. The scale of data is such that the nascent field of stream mining has emerged as a possible solution, because in many cases merely storing all this data is not practical or even feasible.

Structural Metadata. Aside from the sheer number of documents available in databases and streams, text data in most cases comes with *implicit* and *explicit* metadata. Implicit metadata is unstructured information such as topics, sentiment and named entities that can be discovered using text mining methods and incorporated into the document. Explicit metadata is information such as document hierarchy categorization, link

[10] http://en.wikipedia.org/wiki/List_of_largest_libraries
[11] http://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia
[12] http://en.wikipedia.org/wiki/Google_Books
[13] https://blog.twitter.com/2013/celebrating-twitter7
[14] http://techcrunch.com/2014/01/20/whatsapp-dld/

data and various forms of tags that are attached to the document. For example, a Wikipedia article has category labels and different types of external and internal links. The internal links alone can point to any of the millions of documents in Wikipedia, whereas the number of categories is close to half a million in the English Wikipedia. Various types of metadata are practically always present with text data, and the dimensionality of each type can reach millions or more.

Representation Dimensionality and Sparsity. Once extracted from documents, unstructured text needs to be represented with a structured representation such as the word vector for most types of processing. With most useful structured representations, the dimensionality of the representation grows with the number of documents. Word vectors can grow to millions of words. At the same time, most useful representations are inherently sparse: while word vectors grow to millions, each individual document typically contains only some tens of unique words and the corresponding word vector is almost entirely empty.

Vast datasets have become common in TM research as well. Google N-grams[15] released in 2006 contains 5-gram models estimated from 1 trillion words of web text [Brants et al., 2007], and has been widely used for a great variety of TM tasks. The Annotated Gigaword[16] is a collection of over 4 billion words of English text news, enriched with sentence segmentation, parse trees, dependency trees, named entities and in-document coreference chains [Napoles et al., 2012]. Less annotated gigaword corpora have been produced for Arabic, Mandarin, Spanish and French. The 1B Word Language Modeling Benchmark[17] started at the end of 2013 is a freely available billion word dataset for comparing progress in language modeling [Chelba et al., 2013]. Text classification has started to tackle online ontology classifications such as Wikipedia article categorization[18], where the number of categories reaches hundreds of thousands, and combinations of categories reach millions [Puurula and Bifet, 2012]. Classification and retrieval tasks for web ontologies can have hundreds of millions of documents, accessed via cloud-based storage systems[19] [Gross et al., 2013].

[15] http://catalog.ldc.upenn.edu/LDC2006T13
[16] http://catalog.ldc.upenn.edu/LDC2012T21
[17] http://code.google.com/p/1-billion-word-language-modeling-benchmark/
[18] http://lshtc.iit.demokritos.gr
[19] http://trec-kba.org/

2.3.2 Views on Scalability

By definition TM seeks methods that discover new information by examining large quantities of data, as opposed to small-scale analysis that can be done by trained human specialists. There are three ways to view scalability in TM. Scalability can as an *avoidable* challenge, as a *beneficial* factor in building TM systems, or as *necessary* for many TM tasks. This thesis argues that scalability is both beneficial and necessary. These three views are discussed next.

Scalability can be seen be seen as something to be avoided in the simplest way possible. It affects both system-level architectural and component-level algorithmic design. At the component level, most methods that originate in the related technical fields have problems adapting to the TM domain. IR has mostly worked with scalable methods for accessing text, but most methods originating within NLP, ML and DM are less scalable.

Although text is the natural domain of NLP, most NLP methods have been designed on much smaller corpora and emphasize theory, not practical large scale processing. Some methods such as shallow parsers and taggers can operate with linear complexity, but methods such as deep parsers and discriminative models are mostly non-linear and scale poorly to vast numbers of documents [Lafferty et al., 2001, Blei et al., 2003, Collobert et al., 2011]. Non-NLP ML methods have mostly been defined for tasks on much lower dimensions. SVM excel in text classification effectiveness, and are scalable in both documents and features, but less so in the number of classes. Instance-based learning methods such as K-nearest Neighbours scale well to any amount of training data and classes, but are not generally competitive in effectiveness with very sparse high-dimensional vectors, and can become inefficient in inference with vast numbers of training documents. Decision trees in general do not scale to large numbers of classes or documents with high effectiveness. In brief, the majority of ML and DM methods are ill-suited to the scalability required in the text domain.

Some of the problems with unscalable components can be corrected by well designed system-level architectures. If a simple directional processing pipeline is used, the scalability of the TM system equals the scalability of the weakest component in the pipeline. Good decisions at the system level can reduce the bottlenecks in processing. Architectural design solutions can improve scalability, but designing better architectural solutions can be much harder than using better components. This is evidenced in web-scale search engines, where answers to many fundamental questions in architectural design are still largely

unknown, very difficult to exhaustively answer, and subject to change as the web evolves [Cambazoglu and Baeza-Yates, 2013, Asadi and Lin, 2013]. Before embarking on the difficult task of developing techniques for scalable TM, it is therefore necessary to ask whether scalability is warranted, or whether TM should be confined to smaller problems using less scalable tools from the related technical fields. Certainly for most practitioners experienced in the related fields, the obvious solution for TM would be to work on smaller problems with well-understood and widely adopted tools, coupled with data and feature selection, and the affordable large-scale parallelization enabled by cloud computing.

Scalability can also be seen as desirable. NLP has seen an increasing appreciation of scalable processing methods in the last decade. Motivated by the success of large-scale language models used in speech recognition and machine translation [Buck et al., 2014], the usefulness of large datasets for other NLP tasks has become widely acknowledged in a variety of supervised tasks, including disambiguation [Banko and Brill, 2001], parsing [Pitler et al., 2010], spelling correction [Bergsma et al., 2010], segmentation [Huang et al., 2010] and punctuation recovery [Lui and Wang, 2013]. A general observation has been that effectiveness in tasks improves log-linearly with the amount of data [Pitler et al., 2010], so that each multiplication of the amount of data produces a constant relative improvement. This is in line with the improvement observed in large scale n-gram language models; there seems to be no limit to how much improvement comes from more data, as long as the models have sufficient complexity. For example, bigrams seem to saturate practically at some hundreds of millions of words, whereas trigrams should saturate at some billions of words [Rosenfeld, 2000]. Higher-order n-grams and other sufficiently complex models can learn from text data without saturating in the same manner [Brants et al., 2007, Huang et al., 2010, Buck et al., 2014].

Many TM tasks have small amounts of labeled data available compared to large quantities of unlabeled training data. Text classification tasks often require this type of semi-supervised learning. Both estimation based on the EM-algorithm [Nigam et al.] and semi-supervised estimation [Su et al., 2011] enable the utilization of unlabeled data. Large-scale data can also be used in unsupervised learning for tasks that do not require label information, such as the numerous uses of n-gram model counts [Lapata and Keller, 2005]. Alternatively, large-scale unlabeled datasets can be automatically labeled, producing reusable machine-annotated resources for supervised tasks [Napoles et al., 2012].

The most recent view is that scalability is necessary for TM [Řehůřek, 2011]. Traditional IR methods have been developed to scale as much data as possible, since IR systems are intended to retrieve as many relevant documents as possible. Unlike in NLP, scalability has always been a requirement of IR. Similarly, some recent text classification tasks require classification into potentially millions of classes, as in the case of Wikipedia categorization [Puurula, 2012a]. Tasks such as these require scaling to large dimensions, and compromising dimensionality would reduce effectiveness of the methods considerably. It can be further argued that some technologies, such as those utilizing machine translation, only became popular with the arrival of n-gram models trained on trillions of words [Brants et al., 2007].

This view of scalability as necessary stems from the definition of TM. In contrast to text analysis, TM connotes a process of sifting through large quantities of less valuable material to find material of interest. Without data of vast scale, TM reduces to a mere intersection of NLP and ML. Many new text processing tasks are starting to require processing at vast scale. The resulting processing systems are best called TM systems, as they incorporate methods from a variety of disciplines, and do not fit neatly into any of the related technical fields.

2.3.3 Approaches to Scalable Text Mining

A common solution to TM is to treat it as any large-scale processing problem: using general data processing components for processing, and managing scalability with architectural decisions. This can be called a generic approach to scalable TM, and it has a couple of benefits. Components used in other types of data processing can be used in the text domain, and scalability can likewise be managed using well-known solutions. Components of this type include models such as Decision Trees and SVM, while architectural solutions include parallelization solutions such as Map-Reduce. Using generic solutions for both the components and architecture means that the solutions are to a large degree better understood, and both the required software and expertise is more widely available.

A number of generic techniques for scalability can be used at the architectural level:

Selection can be applied to documents, features, classes in other possible dimensions. For the example of feature selection, selection removes from

the data the least important features under some measure of importance [Lewis, 1992, Yang and Pedersen, 1997, Forman, 2003, Elsayed et al., 2008]. Documents can be selected by removing documents classified as spam or noise from the collection [Chekuri et al., 1997, Manning et al., 2008].

Transformation can be likewise applied to variables to reduce dimensionality, as well as to the processing problems themselves. For example, a multi-label classification problem can be approximated by transforming it into a sequence of binary-label classification problems, a multiclass classification problem, or a label ranking problem [Tsoumakas et al., 2010]. Features can be combined or transformed using topic modeling and matrix decomposition methods [Hofmann, 1999].

Precomputing reduces inference time processing by computing and storing as much as possible of the processing offline [Zhang and Poole, 1994, Mohri, 2004].

Caching stores and reuses solutions computed earlier [Skobeltsyn et al., 2008]. Multiple caches can be used to store different types of subsolutions.

Pruning and regularization can be imposed on most generic modeling components, reducing the number of stored parameters and required computation [Zhang and Poole, 1994, Skobeltsyn et al., 2008].

Streaming processes data as a sequence of instances, rather than storing a full dataset in memory as a single batch. Mini-batch processing stores smaller parts of the data, trading memory requirements for model performance. Many algorithms have online versions for stream training, such as Stochastic Gradient Descent [Bottou, 2010], Online EM [Liang and Klein, 2009], and approximations for topic models [Řehůřek, 2011].

Parallelization solves computing problems by using several processors simultaneously to solve sub-problems. Two basic types of parallelization should be distinguished: parallel computing and concurrent computing. Parallel computing solves problems as an array of smaller identical subproblems, and combines the results. Concurrent computing solves problems as a pipeline, forwarding data from one component to another continuously. Multi-core processors, computing clusters, grid computing and cloud computing are possible configurations for utilizing parallelization.

Recent general scientific literature commonly equates scalability with parallelization [Hill, 1990, Kargupta et al., 1997, Dean and Barroso, 2013]. This

has become especially prominent due to the popularity of Big Data and cloud computing as topics over the last decade. Parallelization gives reductions in processing time, up to the number of processors used. For parallel computing, an upper bound of this reduction is known as Amdahl's law [Amdahl, 1967], stating that the maximum speedup gained from parallel computing is dependent on how much of the problem can be solved by parallel computations. The statement can be extended for concurrent computing. Interestingly, Amdahl's 1967 paper is one of the most cited publications in parallel computing, but is very critical of parallel computing. The original reason for Amdahl's law was to present a rigid proof that parallel computing is not a panacea for scalability [Amdahl, 1967].

Much of the TM literature advocates general architectural methods for scalability, rather than ones taking the properties of text data into account. In particular, parallelization and cloud computing are considered by many to be solutions to TM: Baumgartner et al. [2009], Chard et al. [2011], Řehůřek [2011] and Tablan et al. [2013] present TM systems relying exclusively on cloud computing for scalability, Dunlavy et al. [2010] presents a system relying on parallelization and feature selection, while Villalón and Calvo [2013] presents a TM software library advocating parallelization, but implicitly using concurrency and document selection. For most TM tasks, selection is implicitly used, as document selection is commonly considered one of the main stages in TM [Ahonen et al., 1997b], and in data mining generally [Fayyad et al., 1996]. Agichtein [2005] reviews solutions that have been used in scalable information extraction, identifying four main approaches for scalability: 1) scanning the collection using rules, 2) selecting documents using search engines, 3) using customized indexes and 4) distributed processing. The first two are cases of selection and the last one refers to parallelization. The use of customized indexes is more specific to information extraction, and is one type of non-generic solution.

The downside of this generic approach to scalability is that it does not always scale well to TM, and simplifying a problem to suit generic components cannot always be done without unacceptable approximations. As an example, consider categorization of large-scale multi-label ontologies. Tsoumakas et al. [2013] shows a leading solution to BioASQ[20] biomedical article categorization of documents into combinations of 26k possible labels. A majority voting ensemble of four SVMs was trained for ranking the labels, and one for predicting the number of labels per document. The scalability was managed by docu-

[20]http://bioasq.org

ment selection from 10.8M to 3.9M documents, feature selection by removing words occurring less than six times in the collection, and parallelization with a cluster of 40 processors. Training the SVM classifiers took one and a half days on the 40 processors. While this solution is still state-of-the-art for 26k labels, real-world ontologies are becoming larger in all relevant dimensions. A similar solution based on regularized hierarchical SVMs [Gopal and Yang, 2013] was proposed for the LSHTC4[21] datasets for Wikipedia categorization with 325k labels. Using a few approximations in the optimization, this resulted in training times of 37 hours with 512 processors, multiplying estimation times from a smaller Wikipedia dataset one tenth in size by a factor of 25. The real-world datasets and streams mentioned earlier in this chapter are on a totally different scale. Clearly, there is a limit to how much can be accomplished by generic data processing techniques. More fundamental innovations for text processing must be applied, if the emerging vast datasets are to be fully utilized.

A second approach to scalability in TM is to use algorithms intended to scale well with text data by taking the properties of text into account. This can be called the specialized approach to TM. The advantage of using specialized domain expertise is that highly developed solutions for the text domain can be utilized across different types of processing tasks. For example, statistical n-gram language models offer practical and efficient solutions across a variety of text processing tasks.

To some extent many of the applied generic algorithms have already taken properties of text data into account, as the development of the applied algorithms and TM are intertwined in many places. For example, SVMs were proposed for text classification explicitly due to features of word count vectors: high dimensionality, sparsity, high proportion of irrelevant to relevant features, and linear separability in the text classification datasets available at that time [Joachims, 1998]. Likewise, the first use of the Map-Reduce parallelization framework was the computation of web-scale n-grams used at Google [Dean and Ghemawat, 2008]. The vector space model was developed in the context of word vectors, and afterwards applied to other tasks [Salton, 1963]. Many of the related technical fields were developed in the context of processing text data, IR and IE in particular.

Taking the properties of text data into account does not warrant scalable processing. NLP has dealt exclusively with text data, but much of the research has focused on finding computational models that work with small sets

[21] http://lshtc.iit.demokritos.gr/

of text data, while possibly testing associated linguistic theories, such as formal frameworks for describing the grammar of natural languages. The emergence of vast text datasets has made scalable processing methods more common, in particular shallow parsing methods in combination with machine learning [Neumann and Piskorski, 2002, Lui and Wang, 2013].

Specialized solutions for scalable TM can be broadly categorized into *shallow processing*, *hierarchical inference* and *inverted indices*, explained next in detail.

Shallow processing refers to NLP methods that attempt to approximate theoretically grounded deep grammatical processing methods. The typical case is syntactic parsing, where the complexity of the correct model is still unknown, but is at least context free [Chomsky, 1956]. Finite state models provide an approximation to parsing with lower complexity [Koskenniemi, 1990]. Many processing tasks, such as part of speech tagging, phrase chunking and named entity recognition, can be done with low-complexity algorithms such as Hidden Markov Models [Church, 1988]. From a NLP perspective, shallow processing can be seen as an extension of text normalization [Neumann and Piskorski, 2002], whereas from the IE perspective it can be seen as a form of data enrichment [Stajner et al., 2010]. Shallow processing components themselves do not necessarily need to scale in training, since they can be estimated from smaller amounts of data [Tandon and de Melo, 2010, Collobert et al., 2011, Lui and Wang, 2013]. Some linguistic theories argue that language operates on gradually enriched semantic representations [Neumann and Piskorski, 2002, Daum et al., 2003, Sagae et al., 2007]. By constraining deeper processing methods, shallow processing both improves efficiency and effectiveness of further processing stages.

Hierarchical modeling and inference can be applied in numerous ways to utilize the rich structure of text collections. As discussed earlier, text data is embedded with multiple types of implicit and explicit structure. Hierarchical text classification organizes classes into a hierarchy, and classifies documents by traversing the hierarchy from the root towards the leaf classes. This reduces the number of classes that need to be considered to the logarithm of the number of classes [Koller and Sahami, 1997, Tsoumakas et al., 2010, Gopal and Yang, 2013]. Hierarchical clustering can be used for clustering documents, reducing the complexity of clustering to comparisons within each subcluster in the hierarchy [McCallum et al., 2000]. Hierarchical ranking in text retrieval uses a cascade of inverted indices with increasing degrees of granularity [Wang

et al., 2011]. Scalability in parsing and entity recognition can be improved by using hierarchical representations [Petrov and Klein, 2007, Kiddon and Domingos, 2010, Singh et al., 2011]. All of these cases of text processing use the same idea of coarse-to-fine processing, improving modeling effectiveness and scalability by organizing variables into a hierarchy. Graph-structured variables can be approximated using hierarchies, and unsupervised modeling can discover many types of variables implicitly present in text documents that can be used for hierarchical modeling.

Inverted indices have constituted the main data structure for efficient text retrieval for several decades [Zobel and Moffat, 2006]. A more recent development is the use of inverted indices in IE, starting from an IBM TM system called WebFountain [Gruhl et al., 2004]. Indices can be enriched with a variety of information obtained through shallow processing of the documents. Words can be indexed after enrichment with person identification, location [Gruhl et al., 2004], part-of-speech and dependency information [Cafarella et al., Cafarella and Etzioni, 2005], relation tuples [Banko et al., 2007], entity types, predicate-argument relationships, semantic frames, frame roles, frame-denoting elements, events, attributes and relations [Hickl et al., 2007]. Virtually any type of information can be included in an enriched inverted index [Gruhl et al., 2004, Agichtein, 2005]. This increases the index size, but lowers the complexity of retrieving documents matching an indexed annotation. For example, looking up documents that refer to a certain entity can be done simply by going through a postings list that contains documents classified to refer to that entity. This lowers the complexity of some types of processing, and has been considered the most promising method for making information extraction scalable [Agichtein, 2005].

These three strategies improve scalability of text processing by utilizing properties of text data. Shallow processing utilizes the structural nature of implicit variables in text data. Hierarchical processing utilizes hierarchical representations of both implicit and explicit variables associated with text. Enriched inverted indices and sparse processing utilize the sparsity of common representations of text, such as word vectors. Chapter 5 shows how hierarchical processing and enriched inverted indices can be combined to perform scalable probabilistic inference for a variety of TM tasks.

Chapter 3

Multinomial Naive Bayes for Text Mining

This chapter gives an overview of the Multinomial Naive Bayes (MNB) model for text mining, and its generative and graphical model extensions. A basic broad definition of MNB is first given. Generative models related to MNB are described, including mixture models, topic models, n-gram models, Hidden Markov Models, and Dynamic Bayes Networks. The notation of graphical models is introduced, and the connection of directed generative graphical models to the more general factor graphs is discussed. Dynamic programming algorithms for operating with directed graphical models are described, including Viterbi, forward, and expectation maximization.

3.1 Multinomial Naive Bayes

3.1.1 Introduction

Multinomial Naive Bayes (MNB) is a probabilistic model of count data commonly used for various tasks in text mining. MNB originates in text classification research, but the model goes under different names in fields related to text mining. In text clustering, the equivalent model is called a generative multinomial model [Zhong and Ghosh, 2005]. In information retrieval, a special case of MNB is the query likelihood language model (LM) [Kalt, 1996, Hiemstra and Kraaij, 1998, Ponte and Croft, 1998, Zhai and Lafferty, 2001a]. Many other methods can be related to MNB, either as extensions or modifications. MNB and related methods can be said to form one of the core statistical models of text mining.

The MNB model originates from the Naive Bayes (NB) model for text clas-

sification, which is a Bayes model that simplifies model estimation by making strong independence assumptions on features. NB was suggested in a 1961 paper by Maron [Maron, 1961]. This paper was pioneering and even visionary in multiple ways. The paper introduced the idea of automatic text classification, the Bernoulli NB model for text classification, a correction to the zero-frequency problem of NB models, evaluation using held-out test data, and used modern terminology as well as vector notation for describing the model. Maron's work received limited continuation until the early 90s, when text classification started to become a major topic in the machine learning (ML) and data mining fields. Text classification and models related to NB were extensively researched for a decade, until learned linear classifiers such as Support Vector Machines (SVM) became popular due to their superior accuracy [Joachims, 1998].

By the end of the 90s, MNB was identified to be considerably better than Bernoulli NB for most text classification uses [Lewis, 1998, McCallum and Nigam, 1998, Rennie et al., 2003], but generally less accurate than discriminative classifiers such as Logistic Regression (LR) and SVMs [Joachims, 1998, Rennie et al., 2003]. It was also noted that the strong modeling assumptions in MNB reduced performance, and modifying MNB could bring its performance closer to the discriminative classifiers [Rennie et al., 2003, Schneider, 2005, Frank and Bouckaert, 2006]. During the next decade, MNB and related methods spread to various other text mining tasks, in many cases becoming baseline methods, while research interest in generative models for text started to diversify into extensions such as mixture models [Li and Yamanishi, 1997, Monti and Cooper, 1999, Toutanova et al., 2001, Novovicova and Malik, 2003] and topic models [Hofmann, 1999, Blei et al., 2003].

MNB and generative models of text have remained popular due to several advantages, specifically:

Simplicity NB models are very simple to describe and implement. They are among the first models taught to students in ML, prior to learned linear models such as SVM and LR. Simplicity also means that the estimated model parameters can be intuitively understood. More complex ensemble methods can be used to combine a set of NB classifiers, providing a high performance solution that is not a black-box [Elkan, 1997].

Probabilistic Formulation The probabilistic formulation of MNB confers several advantages. The parameters and posterior probabilities of MNB can be easily visualized and interpreted. Text mining is sometimes used

as a component for general data mining and statistical analysis. Probabilistic models can be better integrated into complex modeling than non-probabilistic components.

Versatility Text mining applications vary greatly, and most text mining tools are specialized into solving particular problems. MNB can be directly used in text classification, ranking and clustering, among other uses. Graphical model extensions such as HMMs, mixture models, and conditional N-gram models can be used for modeling structured data. Specialized extensions can be made for handling different types of structured data, such as multi-label outputs [McCallum, 1999] and multi-field documents [Wang et al., 2010b].

Robustness The variability in text mining applications causes different types of modeling challenges, such as limited training data and mismatch between training and test datasets. Despite making strong assumptions, NB models seem to perform well empirically [Domingos and Pazzani, 1997] and MNB is commonly used as a baseline model in text mining applications.

Scalability The amount of available data is growing at an exponential rate. Text datasets as word vectors are high-dimensional in the number of documents, words and labels. MNB has a simple form that results in linear time and space complexity for both estimation and inference. Unlike many methods for text mining, NB models scale linearly in the number of words, documents and labels. This means that MNB can be used on vast datasets, where anything exceeding linear scaling is intractable.

On-line training The vast text datasets that have become available can no longer be stored in the memory of a single computer. This is causing a shift from batch processing to on-line processing, where the data is not kept in memory, but processed as a stream. Many data streams are time-ordered, so that older documents are less useful for building predictive models. Examples of data streams are news stories, microblog messages and other forms of media used for rapid communication. Models working on data streams should support on-line training and down-weighting of older data points. These are trivially implemented for NB models.

Parallelization Large-scale data processing can be tackled with parallelization across multiple processors. Parallel computing frameworks such as MapReduce and Hadoop have become popular solutions for dealing with scalability. One of the main original uses for the MapReduce framework

was training large-scale language models by parallelized count accumulation and combination [Dean and Ghemawat, 2008]. Other types of NB models can be parallelized in the same fashion in both estimation and inference.

The disadvantage of MNB is the effectiveness compared to more complex models. Even in earlier text classification research, NB was used as a "straw man" baseline for comparing more complex models [Domingos and Pazzani, 1997, Lewis, 1998, Rennie et al., 2003]. The introduction of SVM [Joachims, 1998] brought about a gradual decline of interest in MNB and other generative models for text classification, following a general trend in ML towards discriminative classifiers. Currently, discriminative classifiers such as SVMs, LR and Maximum Entropy models are considered the most effective solution for text classification uses such as spam classification, sentiment analysis and document categorization.

3.1.2 Definition

The MNB model considers word count vectors $\boldsymbol{w}$ as being generated by underlying multinomial distributions of words n associated with label variables l. Usually the instances of text are documents and the label variables are classes, document identifiers, or clusters, depending on the task. The label-dependent multinomial distributions $p_l(n)$ are called conditional distributions, and are combined with a categorical prior distribution $p(l)$ of the label variables. A common intuitive explanation is a "generative process", where documents are generated by first sampling a label from the categorical distribution, and then sampling words from the multinomial associated with the label.

A generative model in ML terminology is a model of the joint distribution $p(\boldsymbol{w}, l)$ of input and output variables [Bishop, 2006, Klinger and Tomanek, 2007, Sutton and McCallum, 2007]. For MNB the inputs are word vectors $\boldsymbol{w}$ and the outputs are document labels l. A generative Bayes model factorizes the joint distribution as $p(\boldsymbol{w}, l|\boldsymbol{\theta}) = p_l(\boldsymbol{w}|\boldsymbol{\lambda})p(l|\boldsymbol{\pi})$, so that the parameters $\boldsymbol{\theta}$ are assumed to factorize into parameters $\boldsymbol{\lambda}$ for the *conditionals* $p_l(\boldsymbol{w}|\boldsymbol{\lambda})$ and $\boldsymbol{\pi}$ for the *prior* $p(l|\boldsymbol{\pi})$. Posterior inference can be done by applying Bayes theorem: $p(l|\boldsymbol{w}) = p(\boldsymbol{w}|l)p(l)/p(\boldsymbol{w})$, where $p(l|\boldsymbol{w})$ is called the *posterior* and $p(\boldsymbol{w}) = \sum_l p(\boldsymbol{w}, l)$ the *marginal*. For ranking and classification, the marginal can be omitted and the inference done by maximizing the *joint* $p(\boldsymbol{w}, l)$ instead. In classification, the optimization becomes $\text{argmax}_l\, p_l(\boldsymbol{w})p(l)$. Regression can be performed with continuous variables for labels [Frank et al., 1998].

A problem with Bayes classifiers is estimating the dependencies of the input variables or features n for computing $p_l(\boldsymbol{w})$. By making independence assumptions, computing $p_l(\boldsymbol{w})$ can be simplified. A common assumption is "naive independence', which assumes that the conditional probabilities $p_l(n, w_n)$ are independent, that is, $p_l(\boldsymbol{w}) = \prod_n p_l(n, w_n)$. Models of this type are commonly called NB models. The parameterization of $p_l(\boldsymbol{w})$ can take many forms. A common type of NB model is the multivariate Bernoulli NB [Maron, 1961], where the counts w_n are restricted to binary values $\forall_n : w_n \in \{0, 1\}$ and each class conditional probability $p_l(n, w_n)$ is modeled by a Bernoulli distribution. The Bernoulli distribution models biased "coin-flip" outcomes of a variable by using a single parameter describing how biased the coin flips are. For example, with parameter $\lambda_{ln} = \log(0.1)$, the probability of the word n occurring in a document for label l would be 0.1. The Bernoulli parameters can be estimated simply by counting the training documents $\boldsymbol{w}^{(i)}$ for label l with the word n, and dividing by the number of documents for that label.

The multivariate Bernoulli model for NB [Maron, 1961, Robertson and Jones, 1976, Domingos and Pazzani, 1997, Lewis, 1998, McCallum and Nigam, 1998, Craven et al., 2000] is also known as the Binary Independence Model and Bernoulli NB, and was the first type of NB model suggested for text mining [Maron, 1961]. Other possible models for $p_l(\boldsymbol{w})$ in text mining include Gaussian [Domingos and Pazzani, 1997], multinomial [Lewis, 1998, McCallum and Nigam, 1998, Craven et al., 2000, Rennie et al., 2003, Schneider, 2005, Frank and Bouckaert, 2006, Puurula, 2012b], Poisson [Church and Gale, 1995, Kim et al., 2006, Li and Zha, 2006], Von Mises-Fisher [Banerjee et al., 2005], asymmetric distributions [Bennett, 2003], kernel densities [Ciarelli et al., 2009], and finite mixtures of distributions [Church and Gale, 1995, Banerjee et al., 2005, Li and Zha, 2006].

For most text mining uses the multivariate Bernoulli model has been replaced by the multinomial model of text and its extensions. A multinomial distribution can be seen as a generalization of the Bernoulli distribution into multiple coin-flip outcomes and multiple coin tosses, much like multiple rolls of a biased dice with N sides. A multinomial models the sums of n possible outcomes from $J = \sum_n w_n$ dice rolls. The exact order of the dice rolls in a sequence of J rolls $\underline{w}_j$ is not needed for counting the sums of outcomes w_n. The probability mass function for a label-conditional multinomial distribution of word vectors becomes $p_l(\boldsymbol{w}) = Z(\boldsymbol{w}) \prod_n p_l(n)^{w_n}$. The normalizer $Z(\boldsymbol{w}) = \frac{(\sum_n w_n)!}{\prod_n w_n!}$ takes into account that word vectors can correspond to a number of different word sequences. As it is constant for a given word vector,

it can be omitted in most uses. A common special case of the multinomial is the binomial distribution $N = 2$. Another special case of note is the categorical distribution $\sum_n w_n = 1$.

Using multinomials for Bayes model conditional distributions, we get the joint probability distribution for MNB:

$$\begin{aligned} p(\boldsymbol{w}, l) =& p(l)p_l(\boldsymbol{w}) \\ =& p(l)Z(\boldsymbol{w}) \prod_n p_l(\boldsymbol{w})^{w_n}, \end{aligned} \tag{3.1}$$

where $p(l)$ and $p_l(\boldsymbol{w})$ are parameterized with a categorical and a multinomial, respectively.

Since document lengths J vary, models are defined over all possible lengths, and the multinomials for different lengths have shared parameters. The shared parameters are "tied", since they are constrained to be equal regardless of length. In addition, a distribution such as Poisson must be assumed for generating different document lengths, so that the model can generate the joint distribution over documents of all lengths. The length factor has no practical effect in most applications, and is omitted for posterior inference uses such as clustering, ranking and classification. Therefore both the length factor and the multinomial parameter tying are commonly omitted in the MNB literature [McCallum and Nigam, 1998].

A problem with varying document lengths is that the posterior probabilities for MNB models get increasingly close to either 0 or 1 as the document length increases, since the conditional probability $p_l(\boldsymbol{w})$ is computed by multiplying the N probabilities $p_l(\boldsymbol{w})$ independently [Frank et al., 1998, Monti and Cooper, 1999, Bennett, 2000, Craven et al., 2000]. This scale distortion of the posteriors does not affect classification, ranking or hard clustering, since the rank-order of probabilities for different labels is preserved. For uses such as soft clustering the posterior probabilities can be improved with feature transforms [Pavlov et al., 2004], feature selection [Rigouste et al., 2007, Pinto et al., 2007], and using KL-divergence instead of posterior probabilities to correct for the document lengths [Craven et al., 2000, Schneider, 2005, Pinto et al., 2007]. Nevertheless, for some applications the indirectly estimated posterior probabilities from MNB can be insufficient, and a model directly optimizing the posterior probabilities is preferred, such as LR.

3.1.3 Estimation

Estimation of MNB parameters is commonly done by applying maximum likelihood estimation [McCallum, 1999, Rennie, 2001, Juan and Ney, 2002, Vilar et al., 2004, Madsen et al., 2005, Frank and Bouckaert, 2006]. Due to the data sparsity problem with text data, various smoothing methods are commonly used to correct maximum likelihood parameter estimates. In some cases the smoothed estimation is presented as maximum a posteriori estimation [Rennie, 2001, Schneider, 2005, Smucker and Allan, 2007]. Despite the name "Bayes", NB models are commonly not Bayesian in the sense of Bayesian estimation, where a distribution over parameters is maintained instead of a point estimate of parameters. A fully Bayesian version of MNB has been proposed, but shown to be less suitable than maximum likelihood point estimates [Rennie, 2001]. The most common type of estimation is supervised estimation, where a training dataset D consists of I pairs $D^{(i)} = (\boldsymbol{w}^{(i)}, l^{(i)})$ of word vectors and labels, assumed to be independent and identically distributed (IID). The unsmoothed maximum likelihood estimation approach for the supervised case is described next.

The maximum likelihood method of statistical estimation selects a vector of parameters for a model that maximizes the likelihood of the parameters given the data $\mathcal{L}(\boldsymbol{\theta}|D)$. This equals the probability of the data given the parameters $p(D|\boldsymbol{\theta})$. A conceptual difference between these two is that likelihood is a function of parameters for given training data, whereas probability assumes a model with parameters and can refer to both seen and future data. The MNB likelihood function can be derived:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}|D) &= p(D|\boldsymbol{\theta}) \\
&= \prod_i p(l^{(i)}|\boldsymbol{\theta}) p_{l^{(i)}}(\boldsymbol{w}^{(i)}|\boldsymbol{\theta}) && \text{IID data assumption} \\
&= \prod_i p(l^{(i)}|\boldsymbol{\lambda}) p_{l^{(i)}}(\boldsymbol{w}^{(i)}|\boldsymbol{\pi}) && \text{Bayes model} \\
&= \prod_i p(l^{(i)}|\boldsymbol{\pi}) \prod_n p_{l^{(i)}}(n|\boldsymbol{\lambda})^{w_n^{(i)}} \frac{(\sum_n w_n^{(i)})!}{\prod_n w_n^{(i)}!} && \text{Multinomial conditional}
\end{aligned} \tag{3.2}$$

Maximizing the likelihood can be simplified by noting that the log of the likelihood has the same maximum, but is easier to handle computationally. The MNB log-likelihood decomposes into terms that can be separately optimized. The maximization of the log-likelihood function can be derived:

$$\operatorname*{argmax}_{\boldsymbol{\theta}}(\mathcal{L}(\boldsymbol{\theta}|D)) = \operatorname*{argmax}_{\boldsymbol{\theta}}(\log \mathcal{L}(\boldsymbol{\theta}|D)) = \operatorname*{argmax}_{\boldsymbol{\theta}}(\log p(D|\boldsymbol{\theta}))$$

$$
\begin{aligned}
&= \underset{(\boldsymbol{\lambda},\boldsymbol{\pi})}{\operatorname{argmax}}(\log(\prod_i p(l^{(i)}|\boldsymbol{\pi}) \prod_n p_{l^{(i)}}(n|\boldsymbol{\lambda})^{w_n^{(i)}} \frac{(\sum_n w_n^{(i)})!}{\prod_n w_n^{(i)}!})) \\
&= \underset{(\boldsymbol{\lambda},\boldsymbol{\pi})}{\operatorname{argmax}}(\sum_i (\log p(l^{(i)}|\boldsymbol{\pi}) + \sum_n w_n^{(i)} \log p_{l^{(i)}}(n|\boldsymbol{\lambda}) + \log(\frac{(\sum_n w_n^{(i)})!}{\prod_n w_n^{(i)}!}))) \\
&= \underset{(\boldsymbol{\lambda},\boldsymbol{\pi})}{\operatorname{argmax}}(\sum_i \log p(l^{(i)}|\boldsymbol{\pi}) + \sum_i \sum_n w_n^{(i)} \log p_{l^{(i)}}(n|\boldsymbol{\lambda})) \\
&= \underset{(\boldsymbol{\lambda},\boldsymbol{\pi})}{\operatorname{argmax}}(\sum_l C(l) \log p(l|\boldsymbol{\pi}) + \sum_l \sum_n C(l,n) \log p_l(n|\boldsymbol{\lambda})), \qquad (3.3)
\end{aligned}
$$

where $C(l)$ and $C(l,n)$ refer to the accumulated counts of the variables in the training data.

Optimizing the parameters for the prior and conditionals can now be done separately, and for both cases this consists of choosing the vector of parameters that most likely generated the accumulated vector of counts. The maximum likelihood solution to this estimation of a categorical distribution $p(l|\boldsymbol{\pi})$ is the relative frequency estimate $C(l)/\sum_{l'} C(l')$: $\pi_l = \log(C(l)/\sum_{l'} C(l'))$ and $\lambda_{ln} = \log(C(l,n)/\sum_{n'} C(l,n'))$. This is a standard result in statistics and commonly proven using the method of Lagrange multipliers [Bilmes, 1998, Juan and Ney, 2002].

In practice, estimating the MNB model consists of accumulating the counts in training data and normalizing by the sums of counts, with time complexity $O(IN)$ and space complexity $O(LN)$. Taking sparsity into account, the space complexity of estimation is reduced to $O(L + \sum_l \sum_{n:\exp(\lambda_{ln})>0} 1)$ and time complexity to $O(\sum_i |\boldsymbol{w}^{(i)}|_0)$. The parameters can be represented efficiently using sparse matrix representations. A hash table with (l,n) pairs as keys and parameters as values is one popular choice, with amortized constant time complexities for updating the counts. Another common choice is sparse vectors of word counts and periodic merging of the accumulated vectors with list merge operations. With large-scale datasets this can be done using the map-reduce framework [Dean and Ghemawat, 2008].

Given labeled training data, maximum likelihood estimation of parameters for MNB can be done 1) exactly, 2) with a closed form solution, 3) as online learning, and 4) in linear time complexity in terms of features, documents and classes. These four advantages make MNB highly useful in practical applications. The maximum likelihood estimates are exact, so there are no approximations required, and any system using MNB does not have to consider possible errors from approximations. The closed form solution means that the estimates can be computed by applying elementary mathematical operators,

such as summation and division. The estimation can be done as online learning from streams of text documents, and the effect of older documents can be removed from the estimates trivially. Lastly, the maximum likelihood estimation has linear complexities in all relevant dimensions. Among the common text mining methods, only Centroid and K-nearest Neighbours classifiers have the same scalability in training.

Extending MNB estimation to weighted data can be done by weighting the accumulated counts from each document. In some cases documents are softly labeled, by using a distribution of weights over labels instead of a single label. Extending the estimation to soft labeling can be done similarly, by weighting the counts by the label weights. In case of unsupervised and semi-supervised estimation, the expectation maximization (EM) algorithm [Dempster et al., 1977, Bailey and Elkan, 1994, Bilmes, 1998, Zhong and Ghosh, 2005, Nigam et al., Gupta and Chen, 2011] can be used to estimate parameters. This consists of initialization of parameters followed by EM-iterations of computing the soft labeling $p(l|\boldsymbol{w}^{(i)})$ (expectation step) and re-estimating the model from the soft labeling (maximization step). Each iteration improves the log-likelihood until a stationary point of the likelihood function is reached. Combining EM with multiple random initializations reduces the probability of not reaching a global optimum of the likelihood. Improvements of EM such as online EM [Liang and Klein, 2009] can be used to estimate parameters from stream data and reduce the required amount of EM-iterations.

3.2 Generative Models Extending MNB

3.2.1 Mixture Models

Mixture modeling techniques are commonly used to extend MNB and multinomial models of text, providing both improved modeling precision and a means for incorporating structure into the models. The use of mixtures enables the modeling of multi-modal distributions, with accuracy increasing as a function of the number of added components and the amount of available data to estimate the components.

A basic type of mixture is the Finite Mixture Model [Pearson, 1894]. This models data as being generated by a normalized linear combination of M component distributions, so that for each component m a weight $p(m)$ and a component-conditional distribution $p_m()$ is estimated. The component weights are constrained $0 \leq p(m) \leq 1$ and $\sum_m p(m) = 1$. For example, a finite mixture

of multinomials takes the form:

$$p(\boldsymbol{w}) = \sum_m p(m) Z(\boldsymbol{w}) \prod_n p_m(n)^{w_n} \tag{3.4}$$

By replacing the component variable with the label variable in the MNB model of Equation 3.1, supervised Bayes models can be viewed as mixture models with known component probabilities $p(m|\boldsymbol{w}^{(i)})$ for each training document [McCallum and Nigam, 1998, Novovicova and Malik, 2003, Nigam et al.]. Commonly the components are unknown variables which are estimated in training using approximate algorithms such as EM and optimization on held-out data. Depending on the application and the type of mixture modeling, different optimization algorithms and constraints on the parameters can be used to simplify the estimation problem.

Multinomial models of text can be extended by adding mixtures over both documents and words. Document-clustering mixtures treat documents as being generated by a mixture, with each component corresponding to a prototypical document. Word-clustering mixtures treat words similarly, with each component corresponding to a prototypical word. The document-clustering components can be called *themes*, while word-clustering components can be called *topics* [Keller and Bengio, 2004]. Combination and extension of these two basic types of mixture for text data result in the various mixture and topic models of text.

The earliest proposed document-clustering mixture extension of MNB conditions the label variables on the components [Kontkanen et al., 1996, Monti and Cooper, 1999]. In this use the mixture components cluster the training data into soft partitions:

$$p(\boldsymbol{w}, l) = \sum_m p(m) p_m(l) Z(\boldsymbol{w}) \prod_n p_{ml}(n)^{w_n} \tag{3.5}$$

Replacing each conditional multinomial $p_l(\boldsymbol{w})$ in MNB by a finite mixture of M multinomial components produces the more common Multinomial Mixture Bayes Model [Monti and Cooper, 1999, Novovicova and Malik, 2003, Nigam et al.]:

$$p(\boldsymbol{w}, l) = p(l) \sum_m p_l(m) Z(\boldsymbol{w}) \prod_n p_{lm}(n)^{w_n} \tag{3.6}$$

Word-clustering mixtures are used in topic models and multi-label mixture models [Li and Yamanishi, 1997, Hofmann, 1999, Li and Yamanishi, 2000, Blei et al., 2003, McCallum, 1999, Ueda and Saito, 2002a]. A basic model of this type extends the multinomial:

$$p(\boldsymbol{w}) = Z(\boldsymbol{w}) \prod_n (\sum_m p(m) p_m(n))^{w_n} \tag{3.7}$$

The earliest proposed topic model of this form [Li and Yamanishi, 1997] replaced the multinomial in MNB, using hard clustering of words to form shared components $p_m(n)$, with separate distributions $p_l(m)$. A related model called the Stochastic Topic Model [Li and Yamanishi, 2000] used this type of modeling without label variables, performing inference using the components directly for topic segmentation and analysis. Multi-label classification models use this type of topic model as well, but learn the components from multi-label data. The Multi-label Mixture Model [McCallum, 1999] uses Equation 3.7, but performs inference by greedily adding label components l and estimates $p_l(m)$ for each document, using a prior $p(\boldsymbol{c})$ over the labelsets $\boldsymbol{c} = [1, ..., l, ..., L]$ instead of labels. Parametric Mixture Model [Ueda and Saito, 2002a] is similar, but uses a uniform distributions for $p_l(m)$ and the labelset prior $p(\boldsymbol{c})$. Further multi-label mixture models have built on these two models [Ueda and Saito, 2002b, Kaneda et al., 2004, Sato and Nakagawa, 2007, Wang et al., 2008, Ramage et al., 2009].

Probabilistic topic models became more widely known with Probabilistic Latent Semantic Analysis [Hofmann, 1999]. This uses a unique label variable for each document, so that the joint probability becomes:

$$p(\boldsymbol{w}, l) \propto p(l) \prod_n (\sum_m p_l(m) p_m(n))^{w_n} \tag{3.8}$$

Although popular, the Probabilistic Latent Semantic Analysis model is not a fully generative model of documents [Blei et al., 2003, Keller and Bengio, 2004], since it does not generate new document variables l. The number of components M needs to be optimized, and L is tied to the number of training set documents. Thus the model is not very scalable and is prone to overfitting [Blei et al., 2003]. To address these issues, a model called Latent Dirichlet Allocation [Blei et al., 2003, Minka and Lafferty, 2002] was proposed by re-

placing the document variables in Probabilistic Latent Semantic Analysis with a Dirichlet distribution $p(\boldsymbol{\omega})$ of the component weights $p(m|\boldsymbol{\omega}) = \omega_m$:

$$p(\boldsymbol{w}) = \int p(\boldsymbol{\omega}|\boldsymbol{\tau}) Z(\boldsymbol{w}) \prod_n (\sum_m p(m|\boldsymbol{\omega}) p_m(n))^{w_n} d\boldsymbol{\omega}, \tag{3.9}$$

where $p(\boldsymbol{\omega}|\boldsymbol{\tau})$ is modeled by a Dirichlet distribution, given by:

$$p(\boldsymbol{\omega}|\boldsymbol{\tau}) \propto \frac{\Gamma(\sum_m \tau_m)}{\prod_m \Gamma(\tau_m)} \prod_m \omega_m^{\tau_m}, \tag{3.10}$$

where $\boldsymbol{\tau}$ are the parameters for the Dirichlet distribution and Γ is the gamma function.

The integration over possible component weight vectors has no closed form solution, and must be approximated using algorithms such as variational Bayes, Gibbs sampling, and expectation propagation [Minka and Lafferty, 2002, Asuncion et al., 2009]. The Latent Dirichlet Allocation model proved exceptionally popular and turned probabilistic topic modeling into an active field of research [Griffiths and Steyvers, 2004, Blei and Lafferty, 2006, Li and McCallum, 2006, Asuncion et al., 2009, Blei, 2012].

A substantial literature exists on various models based on Latent Dirichlet Allocation. One conceptually useful model is the Theme Topic Mixture Model [Keller and Bengio, 2004]. This presents a discretized version of Latent Dirichlet Allocation that does not require approximate inference. The Dirichlet over component weights is replaced by a document-clustering mixture:

$$p(\boldsymbol{w}) = \sum_l p(l) Z(\boldsymbol{w}) \prod_n (\sum_m p_l(m) p_m(n))^{w_n} \tag{3.11}$$

Theme Topic Mixture Model presents a direct combination of the document clustering and word clustering finite mixture models. Combinations and extensions of these two types of mixture modeling are used throughout text mining applications, both with multinomial models and distributions other than the multinomial.

3.2.2 N-grams and Hidden Markov Models

The multinomial model of text considers words within documents to be distributed independent of their context. Consequently all phrase- and sentence-level information present in the documents is left unmodeled. This sequence information is crucial for many applications of generative models of text, including machine translation, speech recognition, optical character recognition and text compression. Even in tasks where multinomial text models are considered sufficient, sequence modeling has been shown to be beneficial [Song and Croft, 1999, Miller et al., 1999, Peng and Schuurmans, 2003, Medlock, 2006]. Incorporating sequence information is most commonly done using higher order sequential models called n-grams, also known as Markov chain models. These models originate from the early days of computer science, and aside from the many practical uses they have been instrumental in the development of computer science and information theory [Shannon, 1948, Chomsky, 1956, Markov, 1971].

An n-gram model generalizes the multinomial model to take the preceding sequence of $M-1$ words into account, where M is the order of the n-gram. Each *history* of preceding $M-1$ words models a separate categorical. The models of the first three orders are commonly called unigram, bigram and trigram models, corresponding to zeroth, first and second order Markov chain models, respectively. Over the last decades higher order models such as 4-grams and 5-grams have become standard, with the availability of web-scale text datasets and the development of computer processors and memory. A full n-gram model of order M and vocabulary size of N requires N^M parameters, i.e. counts, for the N^{M-1} categorical distributions. Due to the Zipf-law distribution of text data, the models will be extremely sparse, and only a fraction of these counts will be seen in any amount of training data. For these reasons the main foci in n-gram research have been efficiency of implementation [Siivola et al., 2007, Brants et al., 2007, Watanabe et al., 2009, Pauls and Klein, 2011] and methods for smoothing high order n-grams with lower order estimates [Jelinek and Mercer, 1980, Ney et al., 1994, Chen and Goodman, 1996, 1999, Goodman, 2000, Huang and Renals, 2010, Schütze, 2011].

Let $\underline{\boldsymbol{w}}$ denote any word sequence that corresponds to the counts in the word vector $\boldsymbol{w}$: $w_n = \sum_{j:\underline{w}_j=n} 1$. Let $\underline{w}_{j-M+1}...\underline{w}_j$ denote a subsequence of M words ending at word j. The history or context of an n-gram is the sequence $\underline{w}_{j-M+1}...\underline{w}_{j-1}$ of preceding $M-1$ words, a sequence of 0 words in the unigram

case $M = 1$. The probability of a word sequence is given by:

$$\begin{aligned} p(\underline{\boldsymbol{w}}) &= \prod_j p_M(w_j|j, \underline{\boldsymbol{w}}) \\ &= \prod_j p_M(w_j|\underline{w}_1 ... \underline{w}_{j-1}) && \text{Markov chain} \\ &= \prod_j p_M(w_j|\underline{w}_{j-M+1} ... \underline{w}_{j-1}) && \text{finite history} \end{aligned} \tag{3.12}$$

The probabilities for the first $M-1$ n-grams are undefined, since their histories would span over the word sequence boundaries. To correct this, the sequence can be "padded" by adding start symbols "<s>" at the beginning of the sequence. Sequence end symbols "</s>" can be added, and both of these improve modeling accuracy at the boundaries.

There is a considerable literature on methods for smoothing the n-gram language models. Virtually all of these interpolate n-grams hierarchically with lower order n-grams [Chen and Goodman, 1999]. Let $p_m()$ denote the smoothed m-th order model in the hierarchy and $p^u_m()$ denote the unsmoothed model of the same order. The interpolation smoothed n-gram probabilities can be expressed as:

$$\begin{aligned} p_m(w_j|\underline{w}_{j-m+1} ... \underline{w}_{j-1}) =& (1 - \alpha_m) p^u_m(\underline{w}_j|\underline{w}_{j-m+1} ... \underline{w}_{j-1}) \\ &+ \alpha_m \, p_{m-1}(\underline{w}_j|\underline{w}_{j-m+2} ... \underline{w}_{j-1}), \end{aligned} \tag{3.13}$$

where α_m are backoff weights for order m chosen by the smoothing method. With Jelinek-Mercer smoothing [Jelinek and Mercer, 1980, Chen and Goodman, 1999], α_m are simply fixed parameters estimated on held-out data.

Often a uniform zerogram model is included to end the recursion. Without a zerogram, the different n-grams in hierarchical interpolation methods form a hierarchy of smoothing with M levels. The smoothing weights can be expanded:

$$\begin{aligned} p(\underline{\boldsymbol{w}}) &= \prod_j p_M(\underline{w}_j|\underline{w}_{j-M+1} ... \underline{w}_{j-1}) \\ &= \prod_j \sum_m p(m) p^u_m(\underline{w}_j|\underline{w}_{j-m+1} ... \underline{w}_{j-1}) \\ &= \prod_j \sum_m (\prod_{m'=m+1}^{M} \alpha_{m'} - \prod_{m'=m}^{M} \alpha_{m'}) p^u_m(\underline{w}_j|\underline{w}_{j-m+1} ... \underline{w}_{j-1}), \end{aligned} \tag{3.14}$$

In this form it is seen that n-gram smoothing methods utilize a word-level mixture, generating each word in a sequence as a finite mixture model of the different order models. Hierarchical smoothing means that the mixture component weights are generated dynamically as a product of the higher order back-off weights: $p(m) = \prod_{m'=m+1}^{M} \alpha_{m'} - \prod_{m'=m}^{M} \alpha_{m'}$ [Bell et al., 1989]. Writing a word-clustering mixture of Equation 3.7 in the sequence form shows a further surprising connection:

$$p(\underline{\boldsymbol{w}}) = \prod_j \sum_m p(m) p_m(\underline{w}_j) \tag{3.15}$$

Comparing the word-clustering mixture in this form to the expanded n-gram model of Equation 3.14, we can note the similarity between topic models and hierarchically smoothed n-grams. Both models generate words as a mixture of components. In topic modeling the components correspond to topics, and weights are generated by the chosen topic modeling method. In n-gram modeling the components correspond to the n-gram smoothing hierarchy, and weights are generated dynamically by the chosen smoothing method.

Unlike mixture models, n-gram models have no hidden (unknown) variables in estimation and are efficiently estimated by normalizing the known count statistics. Extending n-grams with hidden variables leads to a more powerful class of models called Hidden Markov Models (HMM) [Baum and Petrie, 1966, Rabiner, 1989, Bilmes, 1998, Miller et al., 1999]. A categorical HMM considers text to be generated as categorical outputs of a hidden Markov chain model, so that only the outputs $\underline{w}_j$ of the Markov chain are seen. A HMM can be seen as an extension of a mixture model, so that the weights of each component become dependent on the history of the last $M-1$ components that generated an output. In HMM terminology the outputs are called emissions or observations, and the hidden variables are called states. In text modeling the outputs are commonly words, and the hidden variables are structural variables such as parts of speech, named entities, sections, topics and authors that are to be discovered using the HMM.

Due to the abundance of applications for HMMs, a number of variants exist that can be mentioned. Arc-emission HMMs output a variable on each transition to a state, whereas state-emission HMMs have the output variables attached to the states. Historically HMMs were defined more commonly as arc-emission HMMs, rather than state-emission HMMs. These two types of HMMs are equivalent, in the sense that either can be transformed to the other.

In many applications the output distributions are Gaussian or mixture models, instead of categoricals. Epsilon or ϵ-states can be used, that have no attached output distribution. HMMs of this type are called ϵ-transition HMMs and are more powerful in the distributions that can be represented, since ϵ-states can be used to model complex sequences of hidden variables behind each output. But these also pose problems for inference, since each output word can be generated by arbitrarily long loops of ϵ-transitions.

Let $\underline{\boldsymbol{k}}$ denote a hidden state sequence of states m corresponding to a word sequence $\underline{\boldsymbol{w}}$. Let M be the number of hidden states, $p_{\underline{k}_j}(\underline{w}_j)$ the categorical output distribution of state $\underline{k}_j$, and $p(\underline{k}_j|\underline{k}_{j-1})$ the transition probability to state $\underline{k}_j$ given the previous state $\underline{k}_{j-1}$. A first-order categorical state-emission HMM without ϵ-transitions produces the joint probabilities:

$$p(\underline{\boldsymbol{w}}, \underline{\boldsymbol{k}}) = \prod_j p(\underline{k}_j|\underline{k}_{j-1}) p_{\underline{k}_j}(\underline{w}_j), \tag{3.16}$$

where the hidden $p(\underline{k}_1|\underline{k}_0) = p(\underline{k}_1)$ is provided by a categorical distribution $p(\underline{k}_1)$ for the initial states $\underline{k}_1$. Alternatively, the sequence can be padded with boundary symbols the same way as with n-gram models.

3.2.3 Directed Graphical Models

MNB and the discussed extensions can be described in the general framework of graphical models [Pearl, 1986, Lauritzen and Spiegelhalter, 1988, Loeliger, 2004, Frey and Jojic, 2005, Klinger and Tomanek, 2007, Parikh and Drezde, 2007, Sutton and McCallum, 2007] as generative directed graphical models. A graphical model is a model of a joint distribution of variables that factorizes according to an underlying graph. Commonly this is an *independency graph* that encodes the independence assumptions of the model. Nodes in an independency graph represent the variables, and lack of an edge between two variables indicates an independence assumption. Since computation is only required for related variables, factorizing a joint distribution according to the assumptions greatly simplifies modeling. Algorithms developed for the estimation and inference of graphical models are applicable to new types of models, reducing the time required for research and development, while the graphical representation reduces the time required for presentation of new models.

The various types of graphical models use different factorizations and graphical notations. Traditionally, graphical models come from two types, called Bayesian Networks [Pearl, 1986] and Markov Random Fields [Kindermann and Snell, 1980]. Both types visualize the variable nodes in the graph with a circle,

(a) Multinomial Naive Bayes (b) Logistic Regression

Figure 3.1: Independency graphs for a Multinomial Naive Bayes and Logistic Regression models. Multinomial Naive Bayes is a directed graphical model, Logistic Regression is an undirected graphical model

and encode the independence assumptions in a model by omissions of edges. Both represent the assumptions using an independency graph $G = (V, E)$ of variable nodes V and edges E. Bayesian Networks are called *directed graphical models*, as the edges, depicted as arrows, encode a directional conditionality of variables. Markov Random Fields are called *undirected graphical models*, and the undirected edges imply dependency, but not directionality. The two types of models are different in the distributions they can represent and the used algorithms. MNB forms a basic directed graphical model, while LR is a corresponding undirected graphical model. Figure 3.1 shows a comparison of MNB and LR using elementary independency graph notation, with the word vector $\boldsymbol{w}$ and label variables l forming the variable nodes $\boldsymbol{x}$ of the graph.

Graphical models factorize a joint distribution $p(\boldsymbol{x})$ to T factors Ψ_t: $p(\boldsymbol{x}) = \prod_t \Psi_t(\mathrm{nd}(t))$, where $\mathrm{nd}(t)$ is the subset of the variable nodes x_t connected to factor Ψ_t. The factors for undirected graphical models are also called cliques, and are constrained to be arbitrary non-negative functions $\Psi_t \geq 0$, where $\Psi_0 = Z = 1/\sum \Psi_{t=1}^{T}(\mathrm{nd}(t))$ is an additional normalization factor, also called the partition function. The factors for a directed graphical model are conditional probability distributions of the form $\Psi_t(\mathrm{nd}(t)) = p(x_t|\mathrm{nd}(t))$ for each node t, with the special case $\mathrm{nd}(t) = \emptyset$ producing marginal probability distributions $p(x_t)$. No additional normalization factor is required for directed graphs, as the conditional distributions are normalized probabilities. Since the factors for directed graphs are defined for each node t, the factorization can be read directly from the graph, whereas the clique factors for undirected graphs are visualized less directly by the independency graph notation.

The notation for graphical models is ongoing constant evolution, and additional notation has been introduced [Minka and Winn, 2008, Dietz, 2010, Andres et al., 2012]. Usually shaded nodes represent known variables, and

clear nodes represent hidden ones. Additional plate notation is standard for repeating parts in graphical models, such as repeated segments in variable sequences. This represents "unrolling" of the graph, so that the segment is repeated a certain number of times. Nevertheless, in practice the plate notation is inconvenient for representing dependencies in sequences. Visualizations of sequence graphical models depict repeated fragments of the models instead [Murphy, 2002, Deviren et al., 2004, 2005, Frey and Jojic, 2005, Wiggers and Rothkrantz, 2006, Sutton and McCallum, 2007, Parikh and Drezde, 2007, Klinger and Tomanek, 2007], sometimes indicating the repetition by use of ellipses, or combining the fragment notation with the plate notation [Wang et al., 2007].

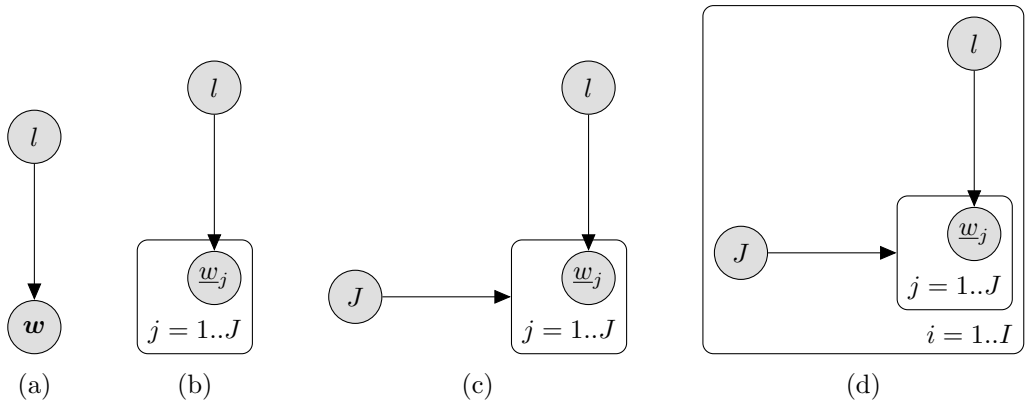
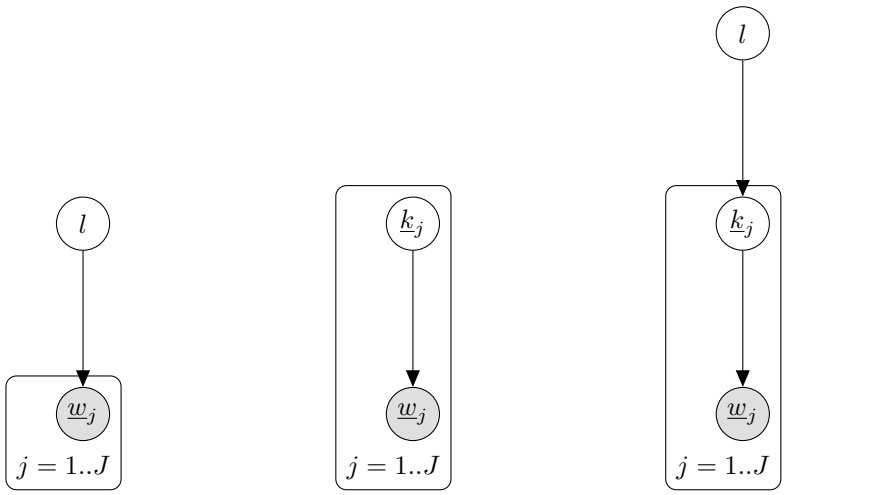


Figure 3.2: Independency graph notations for illustrating MNB, with an increasing degree of explicitness from left to right. The label variable l is considered known for estimation on training data and unknown for inference on test data. Edge from variable J to the plate indicates the plate is unrolled J times

As an example of a directed graphical model, we can first take the MNB $p(l, \boldsymbol{w}) = \Psi_1(l)\Psi_2(l, \boldsymbol{w})$. In this case the factors are categoricals $\Psi_1(l) = p(l)$ and multinomials $\Psi_2(l, \boldsymbol{w}) = p_l(\boldsymbol{w})$. An equivalent definition can be done using sequence variables and graph unrolling. In this case $p(l, \underline{\boldsymbol{w}}) = \prod_t \Psi_t(\mathrm{nd}(t))$, where $\Psi_1(l) = p(l)$ is categorical and $\Psi_{t+1}(l, \underline{w}_t) = p_l(\underline{w}_t)$ for $2 \leq t \leq J+1$ are categorical draws from the multinomial, corresponding to the unrolled variables. Figure 3.2 shows directed independency graph notations for MNB, with varying degrees of explicitness.

Graphical model notations are most useful for expressing an overview of a statistical model, compared to similar models. The mixture models discussed in this section are compared Figure 3.3. When some properties of the model are not crucial for presenting the main modeling ideas, they can be omitted



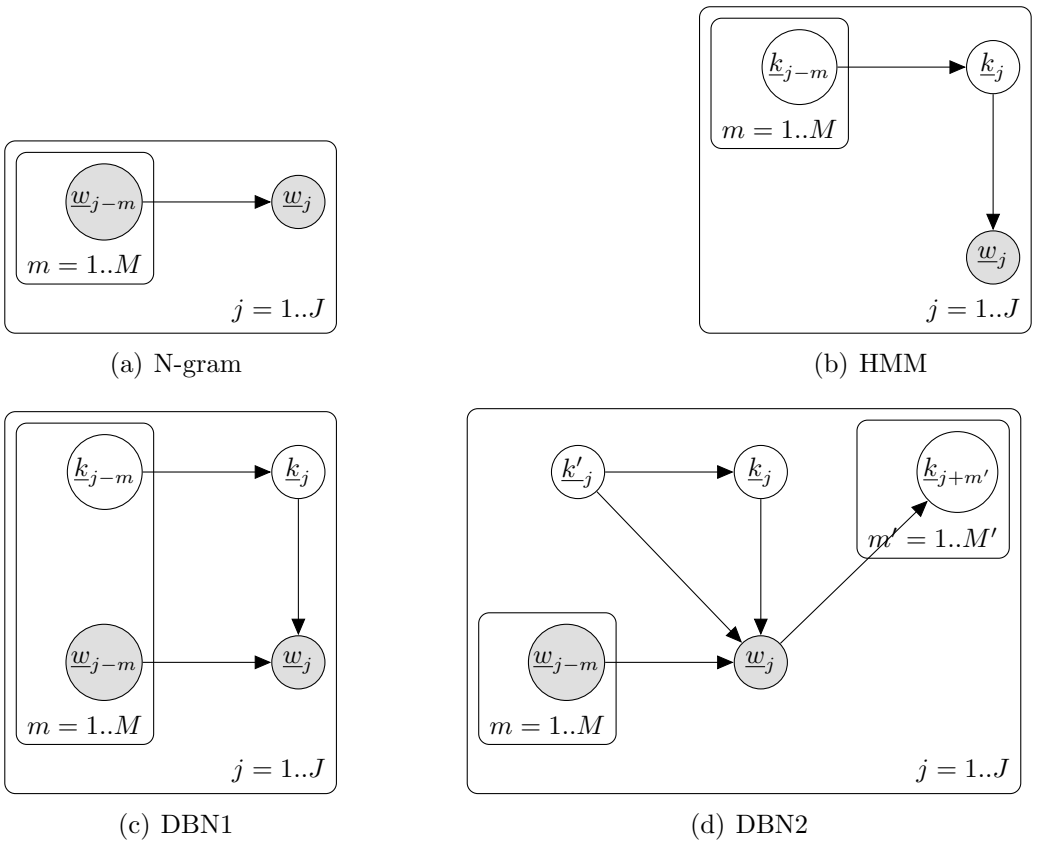
(a) Document mixture (b) Word mixture (c) Theme-topic mixture

Figure 3.3: Comparison of the basic mixture model extensions of multinomials using directed independency graph notation

from the illustration. For example, the models discussed in this chapter all share common modeling ideas, such as the use of categorical distributions and assumption of IID data. Inclusion of these types of properties in the visualization would cause the graphical notations to be less accessible.

The use of unrolled Bayes Networks for sequence modeling has been called Dynamic Bayes Network models (DBN) [Deviren et al., 2004, 2005, Wiggers and Rothkrantz, 2006]. Graph unrolling enables describing sequence models such as topic models, HMMs, and n-grams in the notation of directed graphical models. Topic variables, HMM hidden states and n-gram order interpolation weights can be described as variables in the graph. For example, a categorical HMM can be expressed as $p(\underline{\boldsymbol{w}}, \underline{\boldsymbol{k}}) = \prod_t \Psi(\text{nd}(t))$, where $\Psi_t(\text{nd}(t)) = p_{\underline{k}_t}(\underline{w}_t)$ for $1 \leq t \leq J$ and $\Psi_t(\text{nd}(t)) = p(\underline{k}_{t\%M}|\underline{k}_{t\%M-1})$ for $J+1 \leq t \leq (J+1)M$, where M is the number of HMM hidden states m.

DBNs extend this graphical model view of HMMs, so that a number of hidden variables can underlie an observed word output, instead of a single hidden state variable. For example, a model of text can have topic variables, dialog types, word history, word clusters, parts of speech etc. as the hidden variables [Deviren et al., 2004, 2005, Wiggers and Rothkrantz, 2006]. Including some of these variable types can considerably improve over simple n-gram models of text. Interpolated n-gram models can be included in DBNs by linking to each word the different n-gram order nodes, and an interpolation variable node that outputs the n-gram order mixture weights. With DBNs any conceivable variables can be used to condition the sequence variables, as long as the con-



(a) N-gram (b) HMM

(c) DBN1 (d) DBN2

Figure 3.4: Comparison of the sequence model extensions of multinomials using directed graphical model notation. N-gram and HMM are M-order models. DBN1 combines the n-gram and HMM. DBN2 is a highly connected example of a DBN, with a second hidden variable type $\underline{m'}_j$, and $\underline{w}_j$ conditioning M' future hidden variables $\underline{k}_{j+m'}$.

ditioning does not form cycles in the graph. Figure 3.4 shows a comparison of the sequence model extensions of multinomials.

3.2.4 Factor Graphs and Gates

Graphical models using both directed and undirected edges include chain graphs [Frydenberg, 1990] and ancestral graph Markov models [Richardson and Spirtes, 2002]. More recently, factor graphs [Kschischang et al., 2001, Frey, 2003, Loeliger, 2004, Frey and Jojic, 2005, Lazic et al., 2013] has been proposed as a superset of the earlier frameworks. The original formalism itself has been followed by a number of extensions [Loeliger, 2004, Frey, 2003, Minka and Winn, 2008, McCallum et al., 2009, Dietz, 2010, Andres et al., 2012], such as directed factors [Frey, 2003, Dietz, 2010], gates [Frey, 2003, Minka and Winn, 2008, Dietz, 2010, Oberhoff et al., 2011] and factor templates [McCallum et al., 2009].

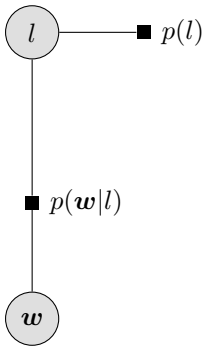


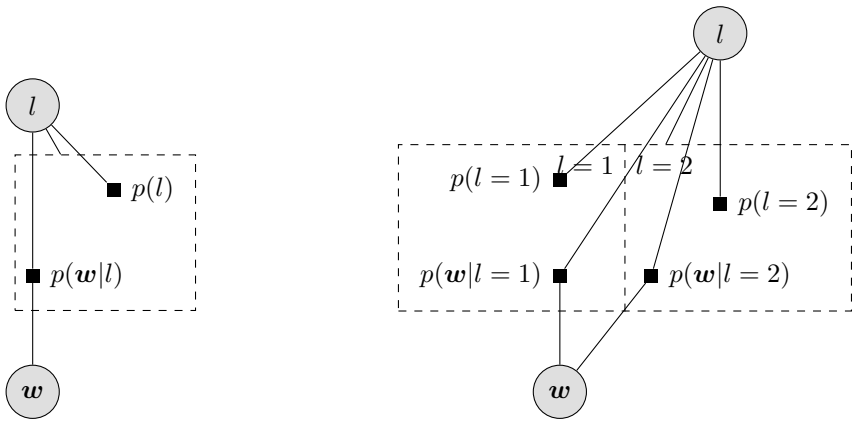
Figure 3.5: Multinomial Naive Bayes illustrated using factor graph notation, with the factors $p(l, \boldsymbol{w}) = \Psi_1(l)\Psi_2(l, \boldsymbol{w})$

The factor graph notation visualizes any graphical model using a graph $G = (V, F, E)$ of variable nodes V, factor nodes F and edges E. The graph is bipartite, so that each edge connects a variable node to a factor node. The variable nodes are visualized using circles, the factor nodes using small squares and the edges using lines. Figure 3.5 shows MNB as a factor graph. Factor graphs are a strict superset of undirected and directed graphical models, since both types of models can be represented as factor graphs, while many factor graphs cannot be represented by either types of models [Frey, 2003].

Gates are a proposed extension of factor graph notation that allows more explicit visualization of mixture models and context-dependent models [Minka and Winn, 2008, Winn, 2012]. Gates notation uses a gate or switch function [Frey, 2003] to select the behaviour of a sub-graph based on a key variable, such as a mixture model component indicator. For example, labels for MNB can be written as a vector of variables $\boldsymbol{c}$: $\forall l : 0 \leq c_l \leq 1$, and $\sum_l c_l = 1$. The joint distribution for MNB can then be rewritten as a gate:

$$p(\boldsymbol{w}, \boldsymbol{c}) = \prod_l (p(\boldsymbol{w}, l))^{c_l} \tag{3.17}$$

The label indicator variables c_l in Equation 3.17 performs the role of a switch, changing the output of the gate to 1 for all labels l with key value $c_l = 0$. The gate formalism enables expression of factor graphs where more general functions are used for the key variables, such as context-dependent variables. For a given key value, only the parts of the gated sub-graph with key value $c_l > 0$ need to be computed.



(a) Gates (b) Expanded gates, for a model with L=2

Figure 3.6: Multinomial Naive Bayes illustrated using gated factor graph notation, with the gate $p(\boldsymbol{w}, \boldsymbol{c}) = \prod_l (p(\boldsymbol{w}, l))^{c_l}$ for factors $p(l, \boldsymbol{w}) = \Psi_1(l)\Psi_2(l, \boldsymbol{w})$

Gates are visualized using dashed rectangles, either as a single rectangle similar to a plate, or in an expanded form with separate rectangles for the different values of the key. The latter is useful in complex cases, when the key values result in different types of computations within the gate. The key variable is shown connected to the gate rectangle by a line. Figure 3.6 shows gated factor graphs for MNB with gates and expanded gates. The gate notation is unnecessary for simple models such as MNB, but becomes useful for illustrating inference in more complex graphical models, as will be shown in Chapter 5 of the thesis.

3.2.5 Inference and Estimation with Directed Generative Models

3.2.5.1 Overview of Algorithms

Directed generative models commonly have efficient algorithms for both inference and estimation. MNB has linear time and space complexity algorithms for both, as do constrained extensions of multinomials such as n-grams. Mixture extensions generally complicate the estimation, and multiply the inference complexities by the number of components in each mixture. Dynamic programming [Bellman, 1952, Viterbi, 1967] is an algorithmic innovation that can be used to reduce the complexities for HMM and DBN extensions. Latent Dirichlet Allocation and some of the more complex directed graphical models require other approximate algorithms for both inference and estimation, such as Variational Bayes and Gibbs Sampling [Asuncion et al., 2009, Blei, 2012].

The joint $p(\boldsymbol{w}, l)$ for MNB is computed by the product $p(l) \prod_n p_l(n)^{w_n}$ over the prior and the label conditionals, and the naive inference algorithm for this is a trivial product over the terms. Inferring the marginal $p(\boldsymbol{w})$ from the joint and posterior $p(l|\boldsymbol{w})$ with the Bayes rule can be done with the closed form operations of sums and products. The naive inference time and space complexities for the MNB posterior $p(l|\boldsymbol{w})$ is $O(|\boldsymbol{w}|_0 L)$ [Manning et al., 2008]. Document-clustering mixtures introduce sums over documents, and word-clustering mixtures introduce sums over words; both multiply the space and time complexities of inference by the number of components J. Replacing the multinomial in MNB with a HMM introduces a sum over all possible hidden state sequences. This can be reduced using dynamic programming techniques. Inference on more complex directed graphical models can require other algorithms, such as the approximate algorithms for Latent Dirichlet Allocation [Asuncion et al., 2009, Blei, 2012].

MNB is estimated by gathering and normalizing the sufficient statistics of counts. When sparsity is utilized, this has the time complexity $O(\sum_i |\boldsymbol{w}^{(i)}|_0)$ and space complexity $O(L + \sum_l \sum_{n:\exp(\lambda_{ln})>0} 1)$. Extension with mixtures complicates the estimation, due to a sum in the likelihood function that does not decompose into separate optimizations. A common practical solution to this problem is the EM algorithm based on dynamic programming. This treats the mixture components as random variables, iteratively optimizing the expected conditional log-likelihood until a stationary point of the likelihood function is reached. As with inference, more complex graphical model extensions can require approximations such as Variational Bayes and Gibbs Sampling. Aside from parameter estimation, sometimes the graphical model structure itself is unknown and learned using a variety of approximate methods [Friedman et al., 1997, Lazic et al., 2013].

Additional meta-optimization in estimation is commonly required, to choose the meta-parameters required by the models, and to avoid bad local minima of the EM-estimated likelihood function. Meta-parameters required by the models can include the number of components in the mixtures and smoothing parameters. Basic solutions for setting these are heuristic values and grid searches. Local minima are encountered with complex multimodal likelihood functions, such as those for mixture models. A basic solution is random restarts for lowering the probability of a low quality local optima. Chapter 6 of the thesis describes a Gaussian random search algorithm that can be used to solve the meta-optimization problem in a principled manner.

3.2.5.2 Dynamic Programming

Dynamic programming [Bellman, 1952] is an algorithmic innovation that can be used to solve problems with overlapping subproblems efficiently. The application of dynamic programming makes HMMs practical, by lowering the complexity of the required computations. These are next described in brief for the case of a first-order categorical HMM. The probability of a word sequence $\underline{\boldsymbol{w}}$ for the HMM of Equation 3.16 can be marginalized:

$$p(\underline{\boldsymbol{w}}) = \sum_{\underline{\boldsymbol{k}}} \prod_j p(\underline{k}_j|\underline{k}_{j-1}) p_{\underline{k}_j}(\underline{w}_j) \tag{3.18}$$

Marginalizing $p(\underline{\boldsymbol{w}})$ by summation over the possible state sequences is not usually feasible, as there are M^J possible state sequences. Computing this observation state probability efficiently is considered the first of the three main computational problems with HMMs [Rabiner, 1989]. The second problem is optimizing $\operatorname{argmax}_{\underline{\boldsymbol{k}}} p(\underline{\boldsymbol{k}}|\underline{\boldsymbol{w}})$, used for segmenting data to the hidden state sequences. The third problem is estimating the model, given that the hidden states are unknown in training data.

The summation over the state sequences $\sum_{\underline{\boldsymbol{k}}}$ in computing $p(\underline{\boldsymbol{w}})$ can be considered a brute-force solution to the problem. A dynamic programming solution is computing the probabilities $p(\underline{\boldsymbol{w}})$ over the sequence indices j instead, summing for each possible state m the probability of all sequences leading to the state, called the forward probability $\xi_j(m)$. This is known as the forward algorithm. Using the forward variables, the marginal probability for the HMM can be rewritten in a recursive form:

$$\begin{aligned} p(\underline{\boldsymbol{w}}) &= \sum_{\underline{\boldsymbol{k}}} \prod_j p(\underline{k}_j|\underline{k}_{j-1}) p_{\underline{k}_j}(\underline{w}_j) \\ &= \sum_m \xi_J(m) \\ \xi_j(m) &= \begin{cases} p(m) p_m(\underline{w}_j), & \text{if } j = 1 \\ \sum_{m'} (\xi_{j-1}(m') p(m|m')) p_m(\underline{w}_j), & \text{otherwise,} \end{cases} \end{aligned} \tag{3.19}$$

where $p(m) = p(\underline{k}_j)$ and $p_m(\underline{w}_j) = p_{\underline{k}_j}(\underline{w}_j)$.

The forward algorithm solves $p(\underline{\boldsymbol{w}})$ recursively by computing $\xi_j(m)$ from $j = 1$ to $j = J$, reducing the time complexity from $O(JM^J)$ to $O(JM^2)$. The space complexity is also reduced to $O(2K)$, since only the forward variables for the current and previous sequence index need to be kept in memory.

Alternatively, the algorithm can be run in the reverse order. This is called the backward algorithm and it produces exactly the same results. These algorithms can be extended to virtually all types of HMMs. The zeroth order HMM can be shown to be a special case, requiring only $O(JM)$:

$$\begin{aligned} p(\underline{\boldsymbol{w}}) &= \sum_m \xi_J(m) \\ &= \sum_m \sum_{m'} ((\xi_{J-1}(m')p(m))p_m(\underline{w}_J)) \\ &= \sum_{m'} (\xi_{J-1}(m')) \sum_m (p(m)p_m(\underline{w}_J)) \\ &= \prod_j \sum_m p(m)p_m(\underline{w}_j)) \end{aligned} \tag{3.20}$$

Related dynamic programming algorithms are used for solving the other two problems with HMMs. Changing the sum over m in Equation 3.20 to a max returns the probability of the word sequence by the single most likely sequence of states, solving the second problem. This is known as the Viterbi algorithm [Viterbi, 1967] and its efficient implementations underlie many of the applications of HMMs into sequence classification. Lastly, the forward and backward algorithms can be combined to compute the posterior probabilities $p(\underline{k}_j|\underline{\boldsymbol{w}})$ of each state $\underline{k}_j$ for each sequence index j. This is known as the forward-backward algorithm, and the posteriors can be used for the Expectation step in EM estimation [Baum et al., 1970, Rabiner, 1989, Bilmes, 1998].

In case a directed graphical model has no cycles of edges, efficient exact dynamic programming inference is possible using extensions of the forward, Viterbi and forward-backward algorithms to general graphs [Pearl, 1986, Kschischang et al., 2001, Loeliger, 2004]. For more complex graphical models a variety of less efficient exact and approximate inference algorithms exist. Most of these build on the idea of variable elimination [Zhang and Poole, 1994], that works by marginalizing variables away to arrive at the inferred probability distribution. The generalized case of forward algorithm to graphs is the sum-product algorithm or belief propagation. Analogously the generalized case of Viterbi algorithm is the max-product algorithm. These extend the use of the forward and maximum variables $\xi_j(m)$, so that a message variable is computed for each node in the graph and the inference is done by passing the messages in an efficient order.

3.2.5.3 Expectation Maximization

The training data for the HMM of Equation 3.16 consists of instances $D^{(i)} = (\underline{\boldsymbol{w}}^{(i)}, \underline{\boldsymbol{k}}^{(i)})$ of known and hidden variables, where $\underline{\boldsymbol{w}}^{(i)}$ is the known sequence and $\underline{\boldsymbol{k}}^{(i)}$ is unknown variables for the i-th instance of training data. The likelihood function becomes:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}|D) &= p(D|\boldsymbol{\theta}) \\ &= \prod_i \sum_{\underline{\boldsymbol{k}}^{(i)}} p(\underline{\boldsymbol{w}}^{(i)}, \underline{\boldsymbol{k}}^{(i)}|\boldsymbol{\theta}) \\ &= \prod_i \sum_{\underline{\boldsymbol{k}}^{(i)}} \prod_j (p(\underline{k}_j^{(i)}|\underline{k}_{(j-1)}^{(i)}, \boldsymbol{\theta})\, p_{\underline{k}_j^{(i)}}(\underline{w}_j^{(i)}|\boldsymbol{\theta}))\end{aligned} \tag{3.21}$$

The summation over components causes a problem for optimizing the log-likelihood of the model, since the log-likelihood no longer factorizes into parts that can be separately optimized. If the hidden states were known and each word was generated by a single component, the log-likelihood function would factorize easily. In this case the likelihood function is:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}|D) &= \prod_i p(\underline{\boldsymbol{w}}^{(i)}, \underline{\boldsymbol{k}}^{(i)}|\boldsymbol{\theta}) \\ &= \prod_i \prod_j (p(\underline{k}_j^{(i)}|\underline{k}_{(j-1)}^{(i)}, \boldsymbol{\theta})\, p_{\underline{k}_j^{(i)}}(\underline{w}_j^{(i)}|\boldsymbol{\theta}))\end{aligned} \tag{3.22}$$

In most cases the hidden states are not known. With mixture models and HMMs the main solution to this problem is using the EM algorithm to estimate the model. Instead of attempting to maximize the log-likelihood, the EM algorithm treats the components as random variables and iteratively optimizes the conditional expectation of the log-likelihood $Q(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}})$. Let $p(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$ indicate the expectation of a hidden variable sequence $\underline{\boldsymbol{k}}^{(i)}$ for word sequence $\underline{\boldsymbol{w}}^{(i)}$ according to some prior parameters $\hat{\boldsymbol{\theta}}$: $p(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}}) = p(\underline{\boldsymbol{k}}^{(i)}, \underline{\boldsymbol{w}}^{(i)}|\hat{\boldsymbol{\theta}}) / \sum_{\underline{\boldsymbol{k}}^{(i)}} p(\underline{\boldsymbol{k}}^{(i)}, \underline{\boldsymbol{w}}^{(i)}|\hat{\boldsymbol{\theta}})$ The Q-function becomes:

$$\begin{aligned}Q(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}}) &= E(\log(\mathcal{L}(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}}))) \\ &= \sum_i \sum_{\underline{\boldsymbol{k}}^{(i)}} p(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}}) \log(p(\underline{\boldsymbol{w}}^{(i)}, \underline{\boldsymbol{k}}^{(i)}|\boldsymbol{\theta})) \\ &= \sum_i \sum_{\underline{\boldsymbol{k}}^{(i)}} \sum_j p(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}}) \log(p(\underline{k}_j^{(i)}|\underline{k}_{(j-1)}^{(i)}, \boldsymbol{\theta})\, p_{\underline{k}_j^{(i)}}(\underline{w}_j^{(i)}|\boldsymbol{\theta})))\end{aligned} \tag{3.23}$$

Since $p(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$ is treated as a random variable, $Q(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}})$ becomes a random variable as well. With random variables the sum of expectations equals the expectation of the sum, so that for computing $\mathcal{L}(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}}))$ the expectation can be pushed inside the sums, placing the expectation $p(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$ as a weight for each possible hidden state sequence. Let $\boldsymbol{\lambda}_m$ indicate the conditional parameters for component m, $\boldsymbol{\lambda}$ the conditional parameters for all components and $\boldsymbol{\alpha}$ the parameters for component weights. The conditional log-likelihood can be optimized:

$$
\begin{aligned}
&\underset{\boldsymbol{\theta}}{\operatorname{argmax}}(Q(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}})) \\
&= \underset{(\boldsymbol{\lambda},\boldsymbol{\alpha})}{\operatorname{argmax}}(\sum_i \sum_{\underline{\boldsymbol{k}}^{(i)}} \sum_j p(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}}) \log(p(\underline{k}_j^{(i)}|\underline{k}_{(j-1)}^{(i)}, \boldsymbol{\alpha})\, p_{\underline{k}_j^{(i)}}(\underline{w}_j^{(i)}|\boldsymbol{\lambda}))) \\
&= \underset{(\boldsymbol{\lambda},\boldsymbol{\alpha})}{\operatorname{argmax}}(\sum_i \sum_{\underline{\boldsymbol{k}}^{(i)}} \sum_j p(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}}) \log(p(\underline{k}_j^{(i)}|\underline{k}_{(j-1)}^{(i)}, \boldsymbol{\alpha})) \\
&+ \sum_i \sum_{\underline{\boldsymbol{k}}^{(i)}} \sum_j p(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}}) \log(p_{\underline{k}_j^{(i)}}(\underline{w}_j^{(i)}|\boldsymbol{\lambda}))) \qquad (3.24)
\end{aligned}
$$

Replacing the current set of parameters $\hat{\boldsymbol{\theta}}$ with the estimated parameters $\hat{\boldsymbol{\theta}}' = \operatorname{argmax}_{\boldsymbol{\theta}}(Q(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}}))$ and repeating the optimization until $Q(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}}) = Q(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}}')$ produces the EM-estimated stationary point of the likelihood function. Depending on the likelihood function and initialization, the stationary point can remain far from a global optimum, and EM is often repeated with randomly sampled initial parameters. Other practical variants include online versions of EM [Liang and Klein, 2009] and combination with genetic algorithms [Martínez and Virtriá, 2000, Pernkopf and Bouchaffra, 2005, Puurula and Compernolle, 2010].

For more scalable algorithmic implementation, EM is implemented using the forward-backward variables $p(\underline{k}_j^{(i)}|\underline{\boldsymbol{w}}^{(\boldsymbol{i})}, \hat{\boldsymbol{\theta}})$ instead of the state sequence variables $p(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$. Summing these weighted posterior probabilities produces expected counts $E(C())$ in place of the counts $C()$ used in Equation 3.3. Using the forward-backward algorithm, the posteriors can be computed in the same time complexity as the forward algorithm, but require $O(JM)$ in space complexity, as the full JM matrix needs to be stored for estimating with the weights.

Chapter 4

Reformalizing Multinomial Naive Bayes

This chapter presents a thorough reformalization of Multinomial Naive Bayes (MNB) as a probabilistic model. The issue of smoothing is first discussed, noting that most of the multinomial smoothing methods necessary for MNB are not correctly formalized under maximum likelihood estimation. A unifying framework for the methods is proposed, and a two-state Hidden Markov Model (HMM) formalization is shown to derive the smoothed parameter estimates. Feature weighting is formalized for both estimation and inference as well-defined approximation with expected log-probabilities, given probabilistically weighted word sequences. A formalization of MNB is defined that takes these corrections into account, followed by a more general graphical model extension that includes label-conditional document length modeling, and scaling the influence of label priors.

4.1 Formalizing Smoothing

4.1.1 Smoothing Methods for Multinomials

The sparsity of text data causes problems for maximum likelihood estimation of multinomials. Words occurring zero times for a label will cause the corresponding parameter estimates to be zero as well, resulting in 0-probabilities when computing the probabilities with *unsmoothed models* $p_l^u(\boldsymbol{w}|\boldsymbol{\lambda}^u)$. This complication is known as the zero-frequency problem in statistics, and smoothing methods for solving this problem have been extensively researched. However, the *smoothed models* $p_l(\boldsymbol{w}|\boldsymbol{\lambda})$ are no longer justified by maximum likelihood, and only a subset of the smoothing methods can be formalized under other principles, such as maximum a posteriori [MacKay and Peto, 1995, Rennie,

2001]. This section proposes a unified framework for smoothing generative models of text, and shows that practically all of the smoothing methods for multinomials can be formalized as approximate maximum likelihood estimation on a constrained Hidden Markov Model (HMM).

An early solution to the zero-frequency problem dates two centuries and is known as Laplace correction [Laplace, 1814]. On discussing the probability of the sun rising tomorrow, Laplace argued that despite the event of the sun not rising has never been seen, it should still have a probability assigned to it. The proposed Laplace correction adds a single count to each possible event, thereby avoiding the zero frequency problem. In language modeling this correction is known as Laplace smoothing, and a parametric generalization adding a fractional count μ instead was proposed by Lidstone in 1920 [Lidstone, 1920]. By further multiplying the added count by a prior background model $p^u(n)$, this correction becomes the Dirichlet prior method for smoothing.

Maximum a posteriori estimation of multinomials with a Dirichlet prior takes the form:

$$\lambda_{ln} = \log(\frac{C(l,n) + \mu\, p^u(n)}{\sum_{n'} C(l,n') + \mu\, p^u(n')}), \tag{4.1}$$

where the special case $p^u(n) = 1/N$ is Lidstone smoothing and $p^u(n) = 1/N$, $\mu = N$ is Laplace correction [Rennie, 2001, Smucker and Allan, 2007].

Another basic way to avoid the zero-frequency problem is to linearly interpolate parameter estimates with a background model. This is known as Jelinek-Mercer smoothing [Jelinek and Mercer, 1980], and takes the form:

$$\lambda_{ln} = \log((1-\beta)\frac{C(l,n)}{\sum_{n'} C(l,n')} + \beta\, p^u(n)) \tag{4.2}$$

Dirichlet prior and Jelinek-Mercer differ only in how the weight for the background model is chosen. A general interpolation function covers both types of smoothing [Johnson, 1932, Smucker and Allan, 2007]:

$$\lambda_{ln} = \log((1-\alpha_l)p_l^u(n) + \alpha_l p^u(n)), \tag{4.3}$$

where $p_l^u(n)$ is the unsmoothed label-conditional multinomial for label l. Fixing α_l to a pre-determined value $\alpha_l = \beta$ produces Jelinek-Mercer smoothing. Fixing $\alpha_l = \frac{\mu}{\mu+\sum_n C(l,n)} = 1 - \frac{\sum_n C(l,n)}{\mu+\sum_n C(l,n)}$ produces Dirichlet prior smoothing. Combining these as $\alpha_l = 1 - \frac{\sum_n (C(l,n)-\beta C(l,n))}{\mu+\sum_n C(l,n)}$ results in two-stage smoothing, a smoothing method suggested for information retrieval [Zhai and Lafferty, 2001a, Smucker and Allan, 2007].

The background model is commonly a uniform distribution $p^u(n) = \frac{1}{N}$, or a label-independent collection model $p^u(n) = \frac{\sum_l C(l,n)}{\sum_{n'} \sum_l C(l,n')}$. A uniform-smoothed collection model with smoothing weight Υ interpolates between a uniform and a collection model: $p^u(n) = (1-\Upsilon)\frac{\sum_l C(l,n)}{\sum_{n'} \sum_l C(l,n')} + \Upsilon\frac{1}{N}$. In n-gram language modeling literature the uniform background model is called the zerogram, and the collection model is called the unigram background model [Chen and Goodman, 1999]. Alternatively, if several documents per label exist, the counts from each document can be normalized by length, or weighted according to usefulness. External datasets can likewise be used to estimate the background model, such as large text datasets of the same language. The choice of background model can also be motivated by the task, such as using a collection model to introduce relevance information in ranked retrieval [Zhai and Lafferty, 2001a].

Jelinek-Mercer and Dirichlet prior are the two most common types of smoothing for MNB models. One view of smoothing is that smoothing methods discount seen occurrences of words in order to redistribute the subtracted probability mass a_l to the background model. Under this view both of these discount the seen counts linearly by α_l. A third basic type of smoothing is called is called absolute discounting [Ney et al., 1994, Chen and Goodman, 1996, 1999, Zhai and Lafferty, 2001b, 2004]. This works similar to Jelinek-Mercer smoothing, but subtracts a parameter value $\delta : 0 \leq \delta \leq 1$ from all counts for a label, and uses the subtracted probability mass for choosing the smoothing coefficient. The discounted counts can be denoted $C'(l,n) = C(l,n) - \delta$. Using Equation 4.3 and choosing $p_l^u(n) = \frac{C'(l,n)}{\sum_{n'} C'(l,n')}$, and $\alpha_l = 1 - \frac{\sum_n C'(l,n)}{\sum_n C(l,n)}$ produces absolute discounting.

A problem with absolute discounting is that usually separate discount values are optimal for different counts, with higher discounts for higher counts [Ney et al., 1994]. Empirical analyses [Chen and Goodman, 1999, Durrett and Klein, 2011, Schütze, 2011, Neubig, 2012] have shown that optimal discount values seem to follow a power-law distribution, rather than the constant ones in absolute discounting. A recent improvement over absolute discounting is power-law discounting [Momtazi and Klakow, 2010, Momtazi, 2010,

Huang and Renals, 2010]. This method discounts according to a power function $C'(l,n) = C(l,n) - \delta C(l,n)^\delta$, with $0 \leq \delta \leq 1$. Combining this with a Dirichlet prior and reorganizing terms produces Pitman-Yor smoothing, with $p_l^u(n) = \frac{C(l,n)-\delta C(l,n)^\delta}{\sum_{n'}(C(l,n')-\delta C(l,n')^\delta)}$ and $\alpha_l = 1 - \frac{\sum_n (C(l,n)-\delta C(l,n)^\delta)}{\mu+\sum_n C(l,n)}$, that approximates inference on a Pitman-Yor process [Momtazi and Klakow, 2010, Momtazi, 2010, Huang and Renals, 2010].

Table 4.1: Smoothing methods used with MNB models. Parameter estimation formulas for smoothing weights α_l and unsmoothed multinomials $p_l^u(n)$.

Smoothing method	Smoothing weight α_l	Multinomial $p_l^u(n)$
Jelinek-Mercer	β	$\frac{C(l,n)}{\sum_{n'} C(l,n')}$
Dirichlet prior	$1 - \frac{\sum_n C(l,n)}{\mu+\sum_n C(l,n)}$	$\frac{C(l,n)}{\sum_{n'} C(l,n')}$
Two-stage smoothing	$1 - \frac{\sum_n (C(l,n)-\beta C(l,n))}{\mu+\sum_n C(l,n)}$	$\frac{C(l,n)}{\sum_{n'} C(l,n')}$
Absolute discounting	$1 - \frac{\sum_n (C(l,n)-\delta)}{\sum_n C(l,n)}$	$\frac{C(l,n)-\delta}{\sum_{n'}(C(l,n')-\delta)}$
Power-law discounting	$1 - \frac{\sum_n (C(l,n)-\delta C(l,n)^\delta)}{\sum_n C(l,n)}$	$\frac{C(l,n)-\delta C(l,n)^\delta}{\sum_{n'}(C(l,n')-\delta C(l,n')^\delta)}$
Pitman-Yor approx.	$1 - \frac{\sum_n (C(l,n)-\delta C(l,n)^\delta)}{\mu+\sum_n C(l,n)}$	$\frac{C(l,n)-\delta C(l,n)^\delta}{\sum_{n'}(C(l,n')-\delta C(l,n')^\delta)}$
Generalized smoothing	$1 - \frac{\sum_n (C(l,n)-D(l,n))}{\mu+\sum_n C(l,n)}$	$\frac{C(l,n)-D(l,n)}{\sum_{n'}(C(l,n')-D(l,n'))}$

The discussed smoothing methods can be covered by a general function that is called here generalized smoothing. By choosing $\alpha_l = 1 - \frac{\sum_n (C(l,n)-D(l,n))}{\mu+\sum_n C(l,n)}$ and $p_l^u(n) = \frac{C(l,n)-D(l,n)}{\sum_{n'}(C(l,n')-D(l,n'))}$, and $D(l,n)$ according to the chosen discounting, we recover all of the smoothing methods as special cases of Equation 4.3. Generalized smoothing with $\mu = 0$ and $D(l,n) = \beta C(l,n)$ implements Jelinek-Mercer smoothing as linear discounting, $D(l,n) = C(l,n) - \delta$ implements absolute discounting and $D(l,n) = \delta C(l,n)^\delta$ implements power-law discounting. A discounting function combining Jelinek-Mercer and power-law discounting can be defined as: $D(l,n) = \delta C(l,n)^\delta + \beta C'(l,n)$, where $C'(l,n) = C(l,n) - \delta C(l,n)^\delta$. Chapter 6 of the thesis experiments with this combined discounting function. Table 4.1 summarizes the smoothing methods in terms of smoothing weights a_l and unsmoothed multinomials $p_l^u(n)$.

The parameters can be chosen to maximize the likelihood of held-out data, or a performance measure related to the task. Closed form approximations requiring no held-out data are possible for some parameters. The discounting parameter δ for absolute and power-law discounting can be approximated using a leave-one-out likelihood estimate [Ney et al., 1994]. Denoting the frequency of 1-counts as $n_1 = \sum_{n:(\sum_l C(l,n))=1} 1$ and 2-counts as $n_2 = \sum_{n:(\sum_l C(l,n))=2} 1$, the discount parameters can be approximated as $\delta = n_1/(n_1+2n_2)$ [Ney et al., 1994, Chen and Goodman, 1999, Huang and Renals, 2010, Zhang and Chiang,

2014]. This Kneser-Ney estimate provides an approximate upper bound of the optimal discount value, and has been demonstrated to work well in practice [Chen and Goodman, 1999, Goodman, 2000, Vilar et al., 2004, Zhang and Chiang, 2014].

The smoothing methods for multinomial text models can be extended into the smoothing methods used for higher-order n-gram language models. A substantial literature exists for advanced n-gram smoothing techniques [Chen and Goodman, 1999, Rosenfeld, 2000]. These extend the multinomial smoothing techniques hierarchically, by placing the lower-order $m-1$ n-gram as the background model for each order m. For example, Witten-Bell smoothing [Moffat, 1990] is hierarchical linear interpolation with a nonparametric estimate for the interpolation weights [Chen and Goodman, 1999]. Using the label-conditional model as a higher order model and the background distribution as a lower-order model, the Witten-Bell estimate for the smoothing weight is $\alpha_l = 1 - \frac{\sum_n C(l,n)}{\sum_{n:C(l,n)>0} 1+\sum_n C(l,n)}$. We can note that Witten-Bell smoothing is a case of Dirichlet prior smoothing with a heuristic estimate for the Dirichlet parameter: $\mu_l = \sum_{n:C(l,n)>0} 1 = \sum_n \min(1, C(l,n))$. Witten-Bell smoothing originates from text compression modelling, where linearly interpolated n-gram models are known as Prediction by Partial Matching (PPM) models [Cleary and Witten, 1984]. The Witten-Bell smoothed PPM is known as PPM-C and has been a baseline for text compression for over two decades. In general applications of n-grams more effective smoothing methods can be applied.

Interpolated Kneser-Ney smoothing [Chen and Goodman, 1999, James, 2000, Goodman, 2000, Siivola and Pellom, 2005, Goldwater et al., 2006, Teh, 2006, Heafield et al., 2013] has been the standard LM smoothing method for n-grams for over a decade. This combines n-grams hierarchically using absolute discounting, but replaces the lower order $m < M$ estimates of counts by the number of contexts the count occurs in [Kneser and Ney, 1995]. For a multinomial, the modified background model becomes $p^u(n) = \frac{\sum_{l:C(l,n)>0} 1}{\sum_{n'} \sum_{l:C(l,n')>0} 1}$. A common example for this method is the phrase "San Francisco". A unigram-model estimate for "Francisco" could be very high, but since the unigram is likely to occur in only this one context, the bigram Kneser-Ney estimate for this lower-order n-gram count would likely be 1. The modified model would correctly consider the unigram "Francisco" as very unlikely to occur outside this context. All but the highest-order unsmoothed models are replaced by the modified counts before discounting, providing a considerable improvement in modeling precision [Chen and Goodman, 1999, James, 2000].

Some improvements over interpolated Kneser-Ney have been suggested over the years, with limited acceptance. Modified Kneser-Ney smoothing [Chen and Goodman, 1999, Siivola and Pellom, 2005, Heafield et al., 2013, Zhang and Chiang, 2014] replaces the discount for each order with three different discounts for counts 1, 2 and 3+, optimized together on held-out data for perplexity [Chen and Goodman, 1999, Siivola and Pellom, 2005]. For both interpolated and modified Kneser-Ney, the discount parameters can be estimated on held-out data, or approximated with heuristics such as the discussed discount estimate $\delta = n_1/(n_1 + 2n_2)$. Power-law discounting LM [Huang and Renals, 2010] replaces the absolute discounting in interpolated Kneser-Ney with Pitman-Yor Process smoothing.

Smoothing the parameter estimates directly as in Equation 4.3 would cause zero-value parameters to become non-zeros, resulting in loss of the parameter sparsity. The parameter estimates $p_l^u(n) = \lambda_{ln}^u$ and $p^u(n) = \lambda_n^u$ are often kept separate, so that the complexity of storing $p_l(n)$ is not increased. The space complexity of estimation for a smoothed MNB model is $O(L + N + \sum_l |\boldsymbol{\lambda}_l^u|_0)$ and the time complexity is $O(\sum_i |\boldsymbol{w}^{(i)}|_0)$.

4.1.2 Formalizing Smoothing with Two-State Hidden Markov Models

Parameter interpolation is the standard formalization of parameter smoothing for multinomial and n-gram language models [Jelinek and Mercer, 1980, Chen and Goodman, 1996, 1999, Zhai and Lafferty, 2001b, 2004, Smucker and Allan, 2007]. Although parameter interpolation provides a principled method for estimating multinomial parameters, the estimated parameters are not strictly speaking maximum likelihood estimates of the multinomial model, but rather ad-hoc estimates [Hiemstra et al., 2004]. The parameter interpolation introduced in Equation 4.3 and Table 4.1 shows that all smoothing methods can be expressed as a normalized mixture of an unsmoothed multinomial and a background distribution. This analysis is extended next by showing that the smoothing methods can be formulated as maximum expected log-likelihood estimation on a constrained generative model. The proof proceeds by showing that the interpolated parameters used in the smoothed multinomials can be implemented by a categorical HMM, and that constraining the HMM appropriately in parameter estimation reproduces a model with the same joint probabilities as the smoothed multinomial. Compared to formalization of smoothing methods as approximate inference on a Pitman-Yor process and other Bayesian models [MacKay and Peto, 1995, Teh, 2006, Goldwater et al., 2006, Neubig,

2012], the proposed HMM formalization has the advantage that inference using the estimated models is exact.

An early formulation of smoothed document models for IR used a two-state HMM for formalizing Jelinek-Mercer smoothing [Miller et al., 1999]. This model used a HMM with one hidden state for the document distribution and one for collection smoothing [Miller et al., 1999, Xu and Weischedel, 2000]. In addition, this work showed how to integrate bigram models, feature weighting, translation models and relevance feedback using the HMM model. However, the parameter estimation for this model was not well defined, the model was considered to be very different and unrelated to multinomial LMs, and model smoothing was limited to Jelinek-Mercer smoothing. The connection to LMs was discovered shortly afterwards [Hiemstra, 2001], as was the need to use constraints such as parameter tying and fixed parameters for the formalization [Hiemstra, 2001]. Using parameter tying, Jelinek-Mercer smoothed higher-order LMs could be implemented as HMMs [Manning and Schütze, 1999]. Despite considerable interest at the time, the two-state HMM model never became popular in IR, nor was its connection to LM smoothing methods explored. We show in the following that this model can be used to formalize all of the discussed smoothing methods in the maximum likelihood framework.

Let l be a label variable indicating one of the L multinomials sharing a background model. Let $\underline{\boldsymbol{w}}$ be any sequence of J words, and $\underline{w}_j : 1 \leq \underline{w}_j \leq N$ correspond to the words counted in $\boldsymbol{w}$, so that $w_n = \sum_{j:\underline{w}_j=n} 1$. Let $\underline{\boldsymbol{k}}$ be an unknown sequence of $M = 2$ hidden states $\underline{k}_j : 1 \leq \underline{k}_j \leq M$ generating the sequence $\underline{\boldsymbol{w}}$. A probability model over the joint probabilities $(l, \underline{\boldsymbol{w}}, \underline{\boldsymbol{k}})$ can be defined:

$$p(l, \underline{\boldsymbol{w}}, \underline{\boldsymbol{k}}) = p(l) \prod_j p_l(\underline{k}_j) p^u_{l\underline{k}_j}(\underline{w}_j), \tag{4.4}$$

where $p_l(\underline{k}_j)$ is a categorical and $p^u_{l\underline{k}_j}(\underline{w}_j)$ is a categorical conditional on the label l and the hidden state $\underline{k}_j$.

The model of Equation 4.4 is a special case of a 0th order categorical HMM, where $p(l)$ correspond to initial state probabilities, $p_l(\underline{k}_j)$ to HMM state transition probabilities, and $p^u_{l\underline{k}_j}(\underline{w}_j)$ to state emission probabilities. Transitions between the label-conditional states are not allowed, so that each word sequence is produced by a single label. Figure 4.1 shows the graphical model for the two-state HMM.

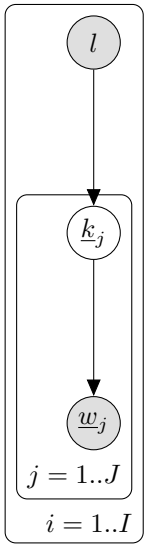


Figure 4.1: Graphical model for the two-state HMM. The hidden variables $\underline{k}_j$ for smoothing components are unknown and the context label variables l are known in estimation

The conditional probabilities $p_l(\boldsymbol{w})$ for the model can be derived:

$$\begin{aligned} p_l(\boldsymbol{w}) &= Z(\boldsymbol{w}) \sum_{\underline{\boldsymbol{k}}} \prod_j p_l(\underline{k}_j) p^u_{l\underline{k}_j}(\underline{w}_j) \\ &= Z(\boldsymbol{w}) \prod_j \sum_m (p_l(m) p^u_{lm}(\underline{w}_j)) \\ &= Z(\boldsymbol{w}) \prod_n (\sum_m p_l(m) p^u_{lm}(n))^{w_n}, \end{aligned} \tag{4.5}$$

where $p_l(m) = p_l(\underline{k}_j)$, $p^u_{lm}(n) = p^u_{l\underline{k}_j}(\underline{w}_j)$, and $Z(\boldsymbol{w})$ is the multinomial normalizer.

The multinomial normalizer $Z(\boldsymbol{w}) = \frac{(\sum_n w_n)!}{\prod_n w_n!}$ accounts for the fact that a count vector $\boldsymbol{w}$ can be generated by the number $Z(\boldsymbol{w})$ of permutations of the sequence $\underline{\boldsymbol{w}}$. As discussed in Chapter 3, computing the product of sums $\prod_j \sum_m$ corresponds to the forward algorithm, whereas computing the equivalent sum of products $\sum_{\underline{\boldsymbol{k}}} \prod_j$ gives the brute-force solution to the same problem [Rabiner, 1989].

The conditional probabilities $p_l(\boldsymbol{w}) = Z(\boldsymbol{w}) \prod_n (\sum_m p_l(m) p^u_{lm}(n))^{w_n}$ can be implemented by a multinomial $p_l(\boldsymbol{w}|\boldsymbol{\lambda}) = Z(\boldsymbol{w}) \prod_n \exp(\lambda_{ln})^{w_n}$, with parameter vector $\boldsymbol{\lambda}_l$: $\lambda_{ln} = \log(\sum_m p_l(m) p^u_{lm}(n))$. From this form it can readily be seen that all of the smoothing methods can be implemented with the two-state HMM of Equation 4.4, by choosing $p^u_{lm=1}(n) = p^u_l(n)$, $p^u_{lm=2}(n) = p^u(n)$, and $p_l(m = 2) = \alpha_l$ as the smoothing weight.

The training data for model of Equation 4.4 consists of documents $D^{(i)} = (\underline{\boldsymbol{w}}^{(i)}, l^{(i)}, \underline{\boldsymbol{k}}^{(i)})$, where $\underline{\boldsymbol{w}}^{(i)}$ are known word sequences, $l^{(i)}$ are label indicators, and $\underline{\boldsymbol{k}}^{(i)}$ are unknown component assignments for the i-th document of training data. The likelihood function becomes:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}|D) &= p(D|\boldsymbol{\theta}) \\
&= \prod_i p(l^{(i)}|\boldsymbol{\theta}) \sum_{\underline{\boldsymbol{k}}^{(i)}} p_{l^{(i)}}(\underline{\boldsymbol{w}}^{(i)}, \underline{\boldsymbol{k}}^{(i)}|\boldsymbol{\theta}) \\
&= \prod_i p(l^{(i)}|\boldsymbol{\theta}) \sum_{\underline{\boldsymbol{k}}^{(i)}} \prod_j p_{l^{(i)}}(\underline{k}_j^{(i)}|\boldsymbol{\theta})\, p^u_{l^{(i)}\underline{k}_j^{(i)}}(\underline{w}_j^{(i)}|\boldsymbol{\theta}) \\
&= \prod_i p(l^{(i)}|\boldsymbol{\theta}) \prod_j \sum_m p_{l^{(i)}}(m|\boldsymbol{\theta})\, p^u_{l^{(i)}m}(\underline{w}_j^{(i)}|\boldsymbol{\theta}) \qquad (4.6)
\end{aligned}$$

This is difficult to optimize, due to the sum within the product. Similarly to the EM-algorithm, $\underline{\boldsymbol{k}}$ can be treated as random variables. Given an initial distribution over component assignments $p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$, the conditional expectation of the log-likelihood becomes:

$$\begin{aligned}
Q(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}}) =& E(\log(\mathcal{L}(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}}))) \\
&= \sum_i \log(p(l^{(i)}|\boldsymbol{\theta})) + \sum_{\underline{\boldsymbol{k}}^{(i)}} p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}}) \log(p_{l^{(i)}}(\underline{\boldsymbol{w}}^{(i)}, \underline{\boldsymbol{k}}^{(i)}|\boldsymbol{\theta})) \\
&= \sum_i \log(p(l^{(i)}|\boldsymbol{\theta})) \\
&+ \sum_{\underline{\boldsymbol{k}}^{(i)}} \sum_j p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}}) \log(p_{l^{(i)}}(\underline{k}_j^{(i)}|\boldsymbol{\theta})\, p^u_{l^{(i)}\underline{k}_j^{(i)}}(\underline{w}_j^{(i)}|\boldsymbol{\theta})) \qquad (4.7)
\end{aligned}$$

Maximizing the conditional expected likelihood parameters for the unsmoothed multinomial $p^u_{lm=1}(n)$, background model $p^u_{lm=2}(n)$, and weights $p_l(m)$ decouples into separate optimizations, given the distribution over component assignments $p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$. Let $\boldsymbol{\lambda}_{lm} = [\lambda_{l1}, ..., \lambda_{lN}]$ indicate the conditional parameters for component m of label l, $\boldsymbol{\alpha}_{lm} = [a_{l1}, ..., a_{lM}]$ the parameters for component weights of label l, and $\boldsymbol{\pi} = [\pi_1, ..., \pi_L]$ parameters for the label priors. Let $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$ indicate the combined parameters for the label-conditionals and component weights. Given $p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$, the maximization decouples:

$$\begin{aligned}
&\operatorname*{argmax}_{\boldsymbol{\theta}}(Q(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}})) \\
=& \operatorname*{argmax}_{(\boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{\pi})}(\sum_i \log(p(l^{(i)}|\boldsymbol{\pi})) \\
&+ \sum_i \sum_j p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}}) \log(p_{l^{(i)}}(\underline{k}_j^{(i)}|\boldsymbol{\alpha})\, p^u_{l^{(i)}\underline{k}_j^{(i)}}(\underline{w}_j^{(i)}|\boldsymbol{\lambda}))
\end{aligned}$$

$$
\begin{aligned}
&= \underset{(\boldsymbol{\lambda},\boldsymbol{\alpha},\boldsymbol{\pi})}{\operatorname{argmax}}\Big(\sum_i \log(p(l^{(i)}|\boldsymbol{\pi})) \\
&\quad + \sum_i \sum_j p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}}) \log(p_{l^{(i)}}(\underline{k}_j^{(i)}|\boldsymbol{\alpha})) \\
&\quad + \sum_i \sum_j p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}}) \log(p^u_{l^{(i)}\underline{k}_j^{(i)}}(\underline{w}_j^{(i)}|\boldsymbol{\lambda}))\Big)
\end{aligned} \tag{4.8}
$$

The parameter estimates for the smoothing methods derive from different assumed distributions $p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$, except the background model for some smoothing methods. Both discounting and linear interpolation imply distributing the expected count mass from a label-conditional model to a background model. Estimation of a shared background model requires further tying of parameters, removing the label-conditional dependency for the second component: $\forall l : p^u_{lm=2}(n) = p^u(n)$. The label-conditional unsmoothed models $p^u_{lm=1}(n)$ and the shared background model $p^u(n)$ become estimated from the count mass distributed by $p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$, while the smoothing weight $p_l(m = 2)$ equals the proportion of count mass distributed to the background model for the label l.

Table 4.2: Two-state HMM parameter estimates with different assumptions for $p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$, with the unsmoothed label-conditional probabilities $p^u_{lm=1}(n)$, smoothing weights $p_l(m = 2)$, and shared background model probabilities $p^u_{lm=2}(n) = p^u(n)$

$p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}\|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$	$p^u_{lm=1}(n)$	$p_l(m=2)$	$p^u(n)$
Jelinek-Mercer	$\frac{C(l,n)}{\sum_{n'} C(l,n')}$	β	$\frac{\sum_{l'} C(l',n)}{\sum_{n'}\sum_{l'} C(l',n')}$
Absolute disc.	$\frac{C(l,n)-\delta}{\sum_{n'}(C(l,n')-\delta)}$	$1 - \frac{\sum_n(C(l,n)-\delta)}{\sum_n C(l,n)}$	$\frac{\sum_{l:C(l,n)>0} 1}{\sum_{n'}\sum_{l:C(l,n')>0} 1}$
Power-law disc.	$\frac{C(l,n)-\delta C(l,n)^\delta}{\sum_{n'}(C(l,n')-\delta C(l,n')^\delta)}$	$1 - \frac{\sum_n(C(l,n)-\delta C(l,n)^\delta)}{\sum_n C(l,n)}$	$\frac{\sum_{l'} \delta C(l',n)^\delta}{\sum_{n'}\sum_{l'} \delta C(l',n')^\delta}$

For some common smoothing methods, the background model is correctly estimated as derived from the two-state HMM formalization. Table 4.2 shows parameter estimates, when $p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$ distributes to form the parameter estimates for Jelinek-Mercer smoothing, absolute discounting and power-law discounting. With Jelinek-Mercer, the collection model estimates derive as the background model. With absolute discounting and power-law discounting, the derived background model estimates equal the modified background models [Kneser and Ney, 1995, Huang and Renals, 2010] proposed for these methods. These modified background models were proposed to satisfy marginal constraints of interpolating a single background model to label-conditional models, but are commonly used to form all $M-1$ order background models in

hierarchical n-gram smoothing [Goodman, 2000, Zhang and Chiang, 2014]. According to the two-state HMM derivation, the models for $M-2$ orders should be estimated from the expected count mass recursively distributed from the higher order models, instead of observed counts [Goodman, 2000], or expected counts [Zhang and Chiang, 2014] of each order independently.

Smoothing methods including a Dirichlet prior will result in untypical background models, since a Dirichlet prior gives weight to the background model in proportion of the sum of counts: $\alpha_l = 1 - \frac{\sum_n C(l,n)}{\mu + \sum_n C(l,n)}$. This distributes the residual count mass for estimating $p^u(n)$ unevenly, so that labels with more accumulated counts contribute less to the background model: $p^u(n) = \frac{\sum_l \alpha_l C(l,n)}{\sum_{n'} \sum_l \alpha_l C(l,n')}$. Using a uniform background model will analogously imply untypical label-conditional models. A uniform background model would distribute the same count mass for each word type n from the label-conditional models. Discounting a fraction δ from the first observation of a word would result in a uniform background model, as would discounting uniformly from each observation of a word: $D(l,n) = \delta / \sum_{l'} C(l',n)$. A uniform background model constrains the choices for the label-conditional and smoothing weight. For example, if a single smoothing weight is used for all labels, the weight will be upper bound by the proportion of discounted count mass: $p_l(m = 2) \leq \text{argmin}_n(\sum_{l'} D(l',n))/(\sum_{l'} C(l',n))$. Common practice chooses the background model as either a collection or uniform distribution, regardless of the label-conditional models and smoothing weights. Alternatively, the background model and the smoothing method hyperparameters can be optimized together for a performance measure of the task [Puurula, 2012b].

Maximizing the expected conditional likelihood corresponds to a single iteration of the EM algorithm on the HMM. Full EM estimation of the parameters would replace $\hat{\boldsymbol{\theta}}$ by $\boldsymbol{\theta}$, and iterate the expectation step of computing $p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)} | \underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$ and the maximization step of solving Equation 4.8 with the updated expectation [Hiemstra et al., 2004]. Nevertheless, the EM parameter estimates would not necessarily reach a global optimum of the model likelihood [Baum et al., 1970, Rabiner, 1989, Bilmes, 1998]. The Q-function parameters estimates can be exact maximum likelihood estimates in some cases, and exact closed form estimates can exist for some sets of constraints. For example, if segmentations of words into states $\underline{\boldsymbol{k}}^{(i)}$ are provided, the parameters are exact maximum likelihood estimates. The case of linear interpolation with the background model and smoothing parameters fixed has a closed form exact maximum likelihood solution [Zhang and Xu, 2008].

The HMM framework for formalizing smoothed multinomials can be extended for formalizing smoothing in more structured generative models. This involves constraining the transitions according to the model [Miller et al., 1999, Manning and Schütze, 1999, Hiemstra, 2001], but the approximate maximum likelihood estimation remains the same. Due to tied parameters, the resulting HMM topologies can become complicated, especially if the model structure and parameter tying is complex. Visualizing the HMMs can be simplified by presenting label-dependent parts of the HMM topology separately [Miller et al., 1999] and illustrating fragments of the model [Manning and Schütze, 1999, Hiemstra, 2001].

4.2 Extending MNB for Fractional Counts

4.2.1 TF-IDF and Feature Transforms with MNB

Parameter smoothing constitutes the primary means for correcting assumptions in models for text such as MNB. In some fields of text mining this is seen as sufficient. For example, in information retrieval the common view is that collection smoothing performs the same task as Inverse Document Frequency (IDF) weighting, and as such integration of IDF to LMs is not necessary [Zhai and Lafferty, 2001a, Hiemstra et al., 2004, Zhai, 2008]. This has been challenged by both experimental results [Smucker and Allan, 2006, Momtazi et al., 2010, Puurula, 2013] and analyses of IDF compared to collection smoothing [Robertson, 2004]. In text classification and clustering, MNB has been combined with Term Frequency (TF) and IDF transforms (TF-IDF) [Rennie et al., 2003, Kibriya et al., 2004, Pavlov et al., 2004, Frank and Bouckaert, 2006, Puurula, 2012b].

Various versions of both IDF and TF-IDF have been proposed for text data [Robertson and Jones, 1976, Salton and Buckley, 1988]. TF-IDF functions comprise three transforms: term count normalization, document length normalization and document frequency weighting. The first two are commonly performed with a combined function and form the TF-part of feature weighting. Document frequency weighting is usually considered an independent factor, and forms the IDF-part of TF-IDF. A combined TF-IDF feature transform is given by:

$$w_n = \log(1 + \frac{w'_n}{||\boldsymbol{w}'||_0}) \log \frac{I}{I_n}, \tag{4.9}$$

where $\boldsymbol{w}'$ is the original unweighted word vector, and I_n is the number of col-

lection documents having the word n.

The choice of log1+ transform for TF normalization can be justified as a correction to multinomials for better modeling the power-law distributions seen in text data [Rennie et al., 2003]. The use of "L0 norm" or the number of non-zeros in the word vector is called unique length normalization, and has been shown to be robust across datasets [Singhal et al., 1996]. The IDF factor $\log \frac{I}{I_n}$ is called Robertson-Walker IDF [Robertson, 2004], and forms the most commonly used version of IDF.

A large number of variations exist for each of the three components and for how they are combined. Term count normalization can be omitted, or use stronger damping [Singhal et al., 1998]. Length normalization can use L1 norm or L2 norm, and can be applied before or after term count normalization. Document frequency weighting by IDF can take a number of forms, one common variant being Croft-Harper IDF [Croft and Harper, 1979], which downweights common words more severely. Parameterized versions also exist, adding versatility to the transforms [Lee, 2007]. A generalized version of Equation 4.9 can be defined [Puurula, 2012b]:

$$w_n = \frac{\log(1 + \frac{w'_n}{|\boldsymbol{w}'|_0^\phi})}{|\boldsymbol{w}|_0^{1-\phi}} \log(\max(1, \upsilon + \frac{I}{I_n})), \tag{4.10}$$

where ϕ controls length scaling and υ IDF lifting. $\phi = 1$ performs length normalization before term count normalization, $\phi = 0$ performs it after. Values $0 < \phi < 1$ produce smooth combinations of term count and length normalization, while $\phi > 1$ and $\phi < 0$ produce more extreme normalizations. $\upsilon = 0$ produces Robertson-Walker IDF, while $\upsilon = -1$ produces unsmoothed Croft-Harper IDF. Values $\upsilon > 1$ produce weaker IDF normalizations, while $\upsilon < -1$ produces stronger IDF normalizations.

4.2.2 Methods for Fractional Counts with Multinomial Models

Transforming data is standard practice for correcting model assumptions in statistical modeling. This enables the use of well-understood simple models with complex data. Common transforms include flooring and ceiling values to accepted bounds, binning, log transforms, and standard score normalization. Feature transforms are less commonly used in text modeling since the models are defined on count data, and most normalizations of count data would produce fractional counts that are undefined for multinomial and categorical

models [Juan and Ney, 2002, Vilar et al., 2004, Tam and Schultz, 2008, Bisani and Ney, 2008, Zhang and Chiang, 2014]. This means that the common use of TF-IDF with MNB produces models that are not well-defined in a probabilistic sense. However, there are a few methods in common use that allow fractional counts in restricted uses.

For inference uses, a method commonly used to weight words in ranking is the Kullback-Leibler (KL) divergence [Zhai and Lafferty, 2001c]. In ranking, KL-divergence is mostly used to incorporate feedback information into test documents, but it has also been used to integrate IDF weighting of words [Smucker and Allan, 2006, Momtazi et al., 2010, Momtazi, 2010]. In soft classification it has been used to correct differences in scores caused by varying document lengths [Craven et al., 2000, Schneider, 2005]. KL-divergence is a measure between two probability functions. For the case of two multinomial distributions $p_l(\boldsymbol{w})$ and $p'(\boldsymbol{w})$, the negative KL-divergence is:

$$\begin{aligned} -D(p_l(\boldsymbol{w})||p'(\boldsymbol{w})) &= -\sum_n p'(n) \log \frac{p'(n)}{p_l(n)} \\ &= \sum_n p'(n) \log p_l(n) - \sum_n p'(n) \log p'(n) \end{aligned} \tag{4.11}$$

The second term $-\sum_n p'(n) \log p'(n)$ is the entropy for model $p'(\boldsymbol{w})$. When KL-divergence is used for ranking or posterior scoring, the model $p'(\boldsymbol{w})$ is the test document or query model, and its entropy can be omitted since it has constant effect on each label model $p_l(\boldsymbol{w})$. If the model $p'(\boldsymbol{w})$ is estimated as the unsmoothed estimate $p'(\boldsymbol{w}) = \frac{w_n}{\sum_n w_n}$ for a test document, then scoring by negative KL-divergence gives rank-equivalent scores to the posterior log-probabilities $-D(p_l(\boldsymbol{w})||p'(\boldsymbol{w})) \stackrel{rank}{=} p(l|\boldsymbol{w}) = \log(p(l)) \sum_n w_n \log(p_l(n))$.

KL-divergence thus provides a framework for generalizing posterior inferences $p(l|\boldsymbol{w})$ by replacing the counts for a test document by model parameters. A common use is to incorporate pseudo-feedback information from the top ranked labels to the document [Zhai and Lafferty, 2001c], in order to rerank the document with the updated model. A more recent use is transforming features [Smucker and Allan, 2006, Momtazi et al., 2010], so that parameters are weighted and renormalized, according to a weighting such as Inverse Collection Frequency [Smucker and Allan, 2006, Momtazi et al., 2010] or IDF [Momtazi et al., 2010]. For example, using IDF would replace the test document model as $p'(n) = Z\, w_n\, IDF(n)$, where Z is a normalization term and $IDF(n)$ the IDF-weight of word n. A problem with the KL-divergence framework is that

it is not probabilistic in a strict sense, since the KL-divergence scores are not probabilities. In addition, it cannot be used to incorporate feature transforms or normalizations in model estimation.

A fully probabilistic alternative to KL-divergence is to define a model that directly uses feature weights. The query term weighting model has been proposed [Momtazi, 2010] for weighting MNB conditional probabilities, so that $p(\underline{\boldsymbol{w}}|l, \underline{\boldsymbol{r}}) = Z_{\underline{\boldsymbol{r}}} \prod_j p_l(\underline{w}_j)^{\underline{r}_j}$, $Z_{\underline{\boldsymbol{r}}}$ is a document-dependent normalizer and $\underline{r}_j$ is an arbitrary non-negative weight, for example $\underline{r}_j = IDF(\underline{w}_j)$. This can be seen as a log-linear model [Darroch and Ratcliff, 1972], where the weights for each feature in the sequence are fixed. The same method was demonstrated earlier for the two-state categorical HMM models in IR, for the special case of weighting query sections, without considering normalization [Miller et al., 1999]. This method presents a simple modification that provides well-defined posterior probabilities for infererence. Like KL-divergence inference, this method cannot be used for model estimation.

For model estimation, static model interpolation [Stolcke, 2002] enables weighting of training data for n-gram LMs. This works identically to Jelinek-Mercer method used for smoothing, but combines weighted components from different training data sources. For example, using K component datasets with multinomial parameters $p_k(n)$ and weights $p(k)$, the interpolated parameters would be $p(n) = \sum_k p(k)p_k(n)$. Basic linear interpolation can be equally implemented by weighting and storing fractional counts from each dataset. Details of the interpolation can vary, and some smoothing heuristics such as the Kneser-Ney discounting estimate require integer count information [Ney et al., 1994, Zhang and Chiang, 2014]. In general this method allows integration of fractional counts in estimation, while maintaining the probabilistic framework.

4.2.3 Formalizing Feature Transforms and Fractional Counts with Probabilistic Data

A method that enables transformed features for both estimation and inference is the formalization of fractional counts as probabilistic data. Concurrent research in estimation of n-grams for machine translation has formalized fractional counts as expectations of counts, given a probability distribution over possible word sequences [Zhang and Chiang, 2014]. This method can be extended for estimation and inference of generative models in a variety of applications, as discussed next in detail.

Table 4.3: Statistics for a word sequence weighted by probabilities

(a) Document word sequence $\underline{\boldsymbol{w}}$ with weights $\underline{\boldsymbol{r}}$

j	$\underline{w}_j$	$\underline{r}_j$
1	1	0.7
2	2	0.8
3	1	0.9

(b) Realizations $\underline{\hat{\boldsymbol{w}}}$ and probabilities $p(\underline{\hat{\boldsymbol{w}}})$

$\underline{\hat{\boldsymbol{w}}}$	$p(\underline{\hat{\boldsymbol{w}}})$
ϵ	0.006
1	0.068
2	0.024
12	0.056
11	0.126
21	0.216
121	0.504

(c) Probabilities of counts $p(w_n = c)$ and expectations of count frequencies $E(\sum_{n:w_n=c} 1)$

	$c = 1$	$c = 2$	$c > 0$
$p(w_1 = c)$	0.34	0.63	0.97
$p(w_2 = c)$	0.80	0.00	0.80
$E(\sum_{n:w_n=c} 1)$	1.14	0.63	

(d) Expected fractional counts $E(\boldsymbol{w}_n)$

$E(w_1)$	1.6
$E(w_2)$	0.8

A weight sequence $\underline{\boldsymbol{r}} = [\underline{r}_1, ..., \underline{r}_J]$ matching a word sequence $\underline{\boldsymbol{w}}$ can be interpreted as probabilities of words occurring, similar to the distribution over hidden components provided by the EM-algorithm. Each weight $\underline{r}_j$ indicates the probability of the corresponding word $\underline{w}_j$ to have occurred in the data, so that the weights define a distribution over possible word sequences. A possible word sequence $\underline{\hat{\boldsymbol{w}}}$ given $\underline{\boldsymbol{w}}$ and $\underline{\boldsymbol{r}}$ can be called a *realization*, a special case being a realization with no words $\underline{\hat{\boldsymbol{w}}} = \epsilon$. A sequence of binary indicator variables $\underline{\hat{\boldsymbol{r}}}$ called an *occurrence sequence* indicates a draw from distribution defined by the weight variables $\underline{\boldsymbol{r}}$. A realization can be generated by different occurrence sequences, and the mapping can be denoted $\underline{\hat{\boldsymbol{w}}} = d(\underline{\hat{\boldsymbol{r}}}, \underline{\boldsymbol{w}})$.

The probability of an occurrence sequence $\underline{\hat{\boldsymbol{r}}}$ can be computed by multiplying the weights: $p(\underline{\hat{\boldsymbol{r}}}|\underline{\boldsymbol{w}}, \underline{\boldsymbol{r}}) = \prod_j \underline{r}_j^{\hat{r}_j}(1 - \underline{r}_j)^{|\hat{r}_j - 1|}$. The probability of each realization $p(\underline{\hat{\boldsymbol{w}}})$ can be computed by summing its occurrence sequences: $p(\underline{\hat{\boldsymbol{w}}}|\underline{\boldsymbol{w}}, \underline{\boldsymbol{r}}) = \sum_{\underline{\hat{\boldsymbol{r}}}=d(\underline{\hat{\boldsymbol{r}}},\underline{\boldsymbol{w}})} p(\underline{\hat{\boldsymbol{r}}}|\underline{\boldsymbol{w}}, \underline{\boldsymbol{r}})$. Assuming the words occur independently, the probabilities of counts $p(w_n = c)$ are distributed according to a Poisson-binomial distribution: $p(w_n = c) = \sum_{j:\underline{w}_j=n} \underline{r}_j$. The expectations of count frequencies $E(\sum_{n:w_n=c} 1)$ can then be computed with a recursive algorithm [Zhang and Chiang, 2014]. Table 4.3 summarizes these basic statistics that can be computed by treating weights for words as probabilities.

The method proposed by Zhang and Chiang [2014] uses the expectations of count frequencies $E(\sum_{n:w_n=c} 1)$ in place of the n_1 and n_2 statistics for the Kneser-Ney estimate, and the expected counts $E(w_n)$ in place of the counts

w_n. The resulting LMs maximize the expected conditional likelihood given the distribution over realizations, similarly to the two-state HMMs in Section 4.1.2. Let training data D for a multinomial distribution consist of word and weight sequences: $D^{(i)} = (\underline{\boldsymbol{w}}^{(i)}, \underline{\boldsymbol{r}}^{(i)})$. The expected conditional log-likelihood can be written:

$$\begin{aligned} Q(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}}) &= E(\log(\mathcal{L}(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}}))) \\ &= \sum_i \sum_{\underline{\hat{\boldsymbol{w}}}^{(i)}} p(\underline{\hat{\boldsymbol{w}}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \underline{\boldsymbol{r}}^{(i)}) \log(p(\underline{\hat{\boldsymbol{w}}}^{(i)})) \\ &= \sum_i \sum_{\underline{\hat{\boldsymbol{w}}}^{(i)}} \sum_{\underline{\hat{w}}_j^{(i)}} p(\underline{\hat{\boldsymbol{w}}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \underline{\boldsymbol{r}}^{(i)}) \log(p(\underline{\hat{w}}_j^{(i)})) \\ &= \sum_n E(C(n)|D) \log(p(n)) \end{aligned} \tag{4.12}$$

This method formalizes the use of fractional counts for estimation, but contains one flaw. With higher order n-grams different realizations have different word histories, since omission of a word causes an n-gram history to skip a word. For example, with a realization $\underline{\hat{\boldsymbol{w}}} = [93]$ for a weighted word sequence $\underline{\boldsymbol{w}} = [92453]$, the subsequence [245] would not be realized, and an n-gram history for the last word $\underline{w}_5 = 3$ in the sequence would have to start with $\underline{w}_1 = 9$. Different realizations will yield different sets of n-gram histories, and assuming fixed histories would become increasingly incorrect with long word sequences and low weights $\underline{r}_j$. Correct estimation should take the differing histories into account. The method presented by Zhang and Chiang [2014] sidesteps this issue by allowing weighting only at the level of sentences. Nevertheless, experimental improvements are demonstrated from the expected Kneser-Ney smoothing [Zhang and Chiang, 2014].

Probabilistic data can be applied equally for inference. The expectation of log-probability $E(\log(p(\underline{\boldsymbol{w}})))$ equals the Q-function for a single document:

$$\begin{aligned} E(\log(p(\underline{\boldsymbol{w}}))) &= \sum_{\underline{\hat{\boldsymbol{w}}}} p(\underline{\hat{\boldsymbol{w}}}|\underline{\boldsymbol{w}}, \underline{\boldsymbol{r}}) \log(p(\underline{\hat{\boldsymbol{w}}})) \\ &= \sum_{\underline{\hat{\boldsymbol{w}}}} \sum_{j=1}^{|\underline{\hat{\boldsymbol{w}}}|} p(\underline{\hat{\boldsymbol{w}}}|\underline{\boldsymbol{w}}, \underline{\boldsymbol{r}}) \log(p(\underline{\hat{w}}_j)) \\ &= \sum_j \underline{r}_j \log(p(\underline{w}_j)) \end{aligned} \tag{4.13}$$

The expectation of probability $E(p(\underline{\boldsymbol{w}}))$ takes the form:

$$\begin{aligned} E(p(\underline{\boldsymbol{w}})) &= \sum_{\underline{\hat{\boldsymbol{w}}}} p(\underline{\hat{\boldsymbol{w}}}|\underline{\boldsymbol{w}}, \underline{\boldsymbol{r}}) p(\underline{\hat{\boldsymbol{w}}}) \\ &= \sum_{\underline{\hat{\boldsymbol{w}}}} p(\underline{\hat{\boldsymbol{w}}}|\underline{\boldsymbol{w}}, \underline{\boldsymbol{r}}) \prod_{j=1}^{|\underline{\hat{\boldsymbol{w}}}|} p(\underline{\hat{w}}_j) \\ &= \sum_{\underline{\hat{\boldsymbol{r}}}} \prod_j p(\underline{\hat{r}}_j) \prod_j p(\underline{w}_j)^{\hat{r}_j} \\ &= \sum_{\underline{\hat{\boldsymbol{r}}}} \prod_j p(\underline{\hat{r}}_j) p(\underline{w}_j)^{\hat{r}_j} \end{aligned} \tag{4.14}$$

These two are not necessarily equivalent, as demonstrated by Jensen's inequality: $E(\log(p(\underline{\boldsymbol{w}}))) \leq \log(E(p(\underline{\boldsymbol{w}})))$. It is not clear which one to use for inference. $E(p(\underline{\boldsymbol{w}}))$ seems to be the natural choice, since it equals the mean of the probability over the distribution. If a model is estimated using the Q-function of equation 4.13, then $E(\log(p(\underline{\boldsymbol{w}})))$ is consistent with the estimation. $E(\log(p(\underline{\boldsymbol{w}})))$ is trivially computed from the weighted counts, whereas $E(p(\underline{\boldsymbol{w}}))$ is more complicated, but can be computed using the forward algorithm from the word and weight sequences in time linear to the sequence length. For this thesis, we will use $E(\log(p(\underline{\boldsymbol{w}})))$, since it gives results that equal the use of fractional counts in existing literature, while no results using $E(p(\underline{\boldsymbol{w}}))$ exist.

4.3 Formalizing MNB as a Generative Directed Graphical Model

The present chapter has formalized the smoothing and feature weighting commonly used with MNB, but there are several omissions in the standard descriptions of MNB. It can be argued that the MNB model is not multinomial, naive, or Bayesian:

1) MNB is usually not a Bayesian model since no distribution over parameters is kept, but rather a generative model using the Bayes rule for posterior inference.

2) MNB is not a Naive Bayes model, since the conditional distribution is modeled by a single multinomial, not by conditionally independent models for each feature.

3) The conditional multinomials in MNB are in fact tied multinomials with parameters shared for all possible documents lengths, combined with a

length-generating distribution such as Poisson.

4) The conditional multinomials themselves have not been formalized correctly, but using a variety of smoothing methods with no connection to maximum likelihood.

5) Feature weighting such as TF-IDF has not been generally formalized for either the training or test set documents, despite being very commonly used with MNB.

Given the common misconceptions about MNB, it is useful to formalize it with a precise definition. We can redefine MNB as a generative model over sequences that factorizes into distributions for labels, word sequence lengths and 2-state HMMs for label-conditionals. Feature weighting can be formalized as estimation and inference using expectations of log likelihoods and log probabilities over probabilistic data. The joint probability for the model factorizes as:

$$\begin{aligned} p(\underline{\boldsymbol{w}}, l, \underline{\boldsymbol{k}}) &= p(l)p(J)p(\underline{\boldsymbol{w}}, \underline{\boldsymbol{k}} |\, l, J) \\ &= p(l)p(J)\prod_j p_l(\underline{k}_j)p^u_{l\underline{k}_j}(\underline{w}_j), \end{aligned} \tag{4.15}$$

where the prior $p(l)$ is categorical, the length generation factor $p(J)$ is Poisson and label-conditionals $p(\underline{\boldsymbol{w}}, \underline{\boldsymbol{k}} |\, l, J)$ are modeled by the 2-state HMMs with $M = 2$ and categoricals $p_l(\underline{k}_j)$ for each l, and $p_{l\underline{k}_j}(\underline{w}_j)$ for each l and m.

The two-state HMM terms correspond to the original MNB terms: $p^u_{lm=1}(n) = p^u_l(n)$ for the label-conditional models, $p^u_{lm=2}(n) = p^u(n)$ for the shared background model, and $p(m = 2) = \alpha_l$ for the smoothing weight. Figure 4.2 shows the graphical model notation for the correctly formalized MNB.

The hidden states can be marginalized away:

$$\begin{aligned} p(\underline{\boldsymbol{w}}, l) &= p(l)p(J)\sum_{\underline{\boldsymbol{k}}}\prod_j (p_l(\underline{k}_j)p^u_{l\underline{k}_j}(\underline{w}_j)) \\ &= p(l)p(J)\prod_j \sum_m (p_l(m)p^u_{lm}(\underline{w}_j)) \\ &= p(l)p(J)\prod_n ((1-\alpha_l)p^u_l(n) + \alpha_l\, p^u(n))^{w_n} \end{aligned} \tag{4.16}$$

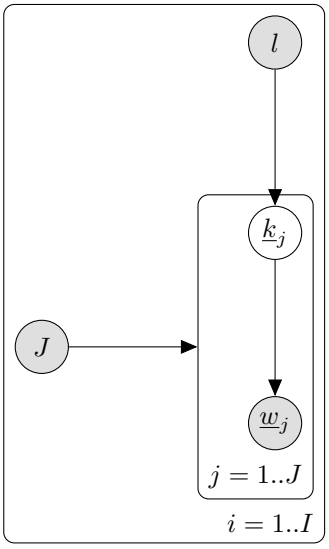


Figure 4.2: Graphical model for Multinomial Naive Bayes, formalizing multinomial smoothing as a 2-state Hidden Markov Model

The label posterior $p(l|\underline{\boldsymbol{w}})$ given a word sequence becomes:

$$\begin{aligned} p(l|\underline{\boldsymbol{w}}) &= \frac{p(\underline{\boldsymbol{w}}, l)}{\sum_{l'} p(\underline{\boldsymbol{w}}, l')} \\ &\propto p(l) \prod_n ((1-\alpha_l) p_l^u(n) + \alpha_l \, p^u(n))^{w_n} \end{aligned} \tag{4.17}$$

The label posterior $p(l|\boldsymbol{w})$ given a word vector becomes:

$$\begin{aligned} p(l|\boldsymbol{w}) &= \frac{p(l)p(J)Z(\boldsymbol{w}) \prod_j \sum_m (p_l(m) p_{lm}^u(\underline{w}_j))}{\sum_{l'} p(l')p(J)Z(\boldsymbol{w}) \prod_j \sum_m (p_{l'}(m) p_{l'k}^u(\underline{w}_j))} \\ &\propto p(l) \prod_n ((1-\alpha_l) p_l^u(n) + \alpha_l \, p^u(n))^{w_n}, \end{aligned} \tag{4.18}$$

where $Z(\boldsymbol{w})$ is the multinomial normalizer.

The word vectors can be used to compute exactly the same label posteriors, meaning that for posterior inference only the word vectors are required, and not the word sequence information. The joint can be marginalized to produce the probability of a word sequence:

$$\begin{aligned} p(\underline{\boldsymbol{w}}) &= \sum_l \sum_{\underline{\boldsymbol{k}}} p(\underline{\boldsymbol{w}}, l, \underline{\boldsymbol{k}}) \\ &= p(J) \sum_l p(l) \sum_{\underline{\boldsymbol{k}}} p_l(\underline{\boldsymbol{w}}, \underline{\boldsymbol{k}}) \end{aligned}$$

$$= p(J) \sum_l p(l) \prod_n ((1-\alpha_l) p_l^u(n) + \alpha_l \, p^u(n))^{w_n} \tag{4.19}$$

From the marginal probabilities MNB can be seen as a mixture of label-conditional models [McCallum and Nigam, 1998, Novovicova and Malik, 2003, Nigam et al.], where the label variables are known for each document in a training dataset. The MNB factorization is simplified by a number of independence assumptions. The length factor $p(J)$ is assumed to be mutually independent with the other factors [Juan and Ney, 2002].

The training data consists of documents $D^{(i)} = (l^{(i)}, \underline{\boldsymbol{w}}^{(i)}, \underline{\boldsymbol{k}}^{(i)})$, where $l^{(i)}$ and $\underline{\boldsymbol{w}}^{(i)}$ are known, and $\underline{\boldsymbol{k}}^{(i)}$ is unknown. The likelihood function is:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}|D) &= p(D|\boldsymbol{\theta}) \\ &= \prod_i p(l^{(i)}|\boldsymbol{\pi}) p(|\underline{\boldsymbol{w}}|_0^{(i)}|\boldsymbol{\chi}) \sum_{\underline{\boldsymbol{k}}^{(i)}} \prod_j p_{l^{(i)}}(\underline{k}_j^{(i)}|\boldsymbol{\alpha}) \, p^u_{l^{(i)}\underline{k}_j^{(i)}}(\underline{w}_j^{(i)}|\boldsymbol{\lambda}) \end{aligned} \tag{4.20}$$

Treating $\underline{\boldsymbol{k}}$ as a random variable distributed according to a prior distribution $p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$, the conditional expectation of the log-likelihood becomes:

$$\begin{aligned} Q(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}}) &= E(\log(\mathcal{L}(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}}))) \\ &= \sum_i (\log(p(l^{(i)}|\boldsymbol{\theta})) + \log(p(|\underline{\boldsymbol{w}}|_0^{(i)}|\boldsymbol{\theta})) \\ &+ \sum_{\underline{\boldsymbol{k}}^{(i)}} \sum_j p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}}) \log(p_{l^{(i)}}(\underline{k}_j^{(i)}|\boldsymbol{\theta}) \, p^u_{l^{(i)}\underline{k}_j^{(i)}}(\underline{w}_j^{(i)}|\boldsymbol{\theta}))) \end{aligned} \tag{4.21}$$

The maximization decouples into separate optimizations:

$$\begin{aligned} \underset{\boldsymbol{\theta}}{\operatorname{argmax}}(Q(\boldsymbol{\theta}|D, \hat{\boldsymbol{\theta}})) = \underset{(\boldsymbol{\pi}, \boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{\chi})}{\operatorname{argmax}}(& \sum_i \log(p(l^{(i)}|\boldsymbol{\pi})) \\ &+ \sum_i \log(p(|\underline{\boldsymbol{w}}|_0^{(i)}|\boldsymbol{\chi})) \\ &+ \sum_i \sum_{\underline{\boldsymbol{k}}^{(i)}} \sum_j p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}}) \log(p_{l^{(i)}}(\underline{k}_j^{(i)}|\boldsymbol{\alpha})) \\ &+ \sum_i \sum_{\underline{\boldsymbol{k}}^{(i)}} \sum_j p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}}) \log(p^u_{l^{(i)}\underline{k}_j^{(i)}}(\underline{w}_j^{(i)}|\boldsymbol{\lambda}))) \end{aligned} \tag{4.22}$$

The parameters $\boldsymbol{\pi}$ and $\boldsymbol{\chi}$ are invariant to the expectation over the hidden states, and are therefore exact maximum likelihood estimates. For many uses the parameters $\boldsymbol{\chi}$ for word sequence lengths can be omitted, since these have no effect on the posterior probabilities. The parameters $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$ are expected log-likelihood estimates given $p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$, as discussed in Section 4.1.2. Choosing $p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$ and a form for the shared background model $p^u(n)$ implements smoothing.

Feature weighting is incorporated by performing estimation and inference over probabilistic data, as described in Section 4.2.3. Given a probabilistic weight sequence $\underline{\boldsymbol{r}}$ matching the word sequence $\underline{\boldsymbol{w}}$, probabilities can be approximated by the expectations of log-probabilities given $\underline{\boldsymbol{r}}$. For both inference and estimation, this reproduces the results that come from simply using fractional counts in algorithms instead of integer counts, as has been done with applications of MNB using weighted words. In Equations 4.16, 4.17, 4.18, and 4.19, approximation with expected log-probabilities replaces the integer vector $\boldsymbol{w}$ by the fractional expected counts $E(\boldsymbol{w}|\underline{\boldsymbol{r}})$, that are provided by any chosen feature weighting function. Maximum expected log-likelihood estimation introduces the weight terms r_j to Equations 4.21 and 4.22. The multinomials for the two-state HMM become weighted by both the distribution of occurring/non-occurring terms defined by $\underline{\boldsymbol{r}}$, and the assumed distribution over the HMM component assignments $p_{l^{(i)}}(\underline{\boldsymbol{k}}^{(i)}|\underline{\boldsymbol{w}}^{(i)}, \hat{\boldsymbol{\theta}})$. The length-modeling factor $p(|\underline{\boldsymbol{w}}|_0^{(i)}|\boldsymbol{\chi}))$ should also take the weighted distribution over occurring word sequences into account, unless the length model is approximated by ignoring weights. Further derivation of Poisson length model estimation on expected sequence lengths is omitted, as the experiments conducted in the thesis evaluate length modeling separately to feature weighting.

4.4 Extending MNB with Prior Scaling and Document Length Modeling

The extension of MNB so far has formalized smoothing methods as a directed graphical model. Further useful extensions to MNB can be defined by modifying the graphical model factorization. Two such extensions are label-conditional document length modeling and scaling of the label prior.

Document lengths in MNB are assumed to be generated by a shared distribution such as Poisson [McCallum and Nigam, 1998, Blei et al., 2003], that can be omitted for most uses. Use of label-conditional distributions has been

suggested in the literature [McCallum and Nigam, 1998], but has not experimented. Since document lengths are known to be among the strongest features for some tasks, such as automatic essay scoring [Larkey, 1998], explicit length modeling could prove useful.

Prior scaling is applied to LMs in uses such as speech recognition, where a LM models the prior distribution of possible word sequences in a Naive Bayes framework [Gales and Young, 2007]. In these uses the prior has a very different scale than the conditional distribution, and a scaling factor is applied to match the contributions of the prior and conditional optimally. Prior scaling can be applied to TM tasks equally, but this has not been attempted.

An Extended MNB incorporating prior scaling and document length modeling can be defined as:

$$\begin{aligned} p(\underline{\boldsymbol{w}}, l, \underline{\boldsymbol{k}}) &= p(l)p(J|l)p(\underline{\boldsymbol{w}}, \underline{\boldsymbol{k}}|\, l, J) \\ &= p(l)p(J|l) \prod_j p_l(\underline{k}_j) p^u_{l\underline{k}_j}(\underline{w}_j), \end{aligned} \tag{4.23}$$

where the label prior $p(l)$ and length model $p(J|l)$ are scaled and renormalized versions of the original distributions $p'(l)$ and $p'(J|l)$: $p(l) = Z(\vartheta)\, p'(l)^{\vartheta}$ and $p(J|l) = Z(\varsigma) p'(J|l)^{\varsigma}$. $Z(\vartheta)$ and $Z(\varsigma)$ normalize the factors to be probability distributions, and ϑ and ς are meta-parameters to be estimated on held-out data.

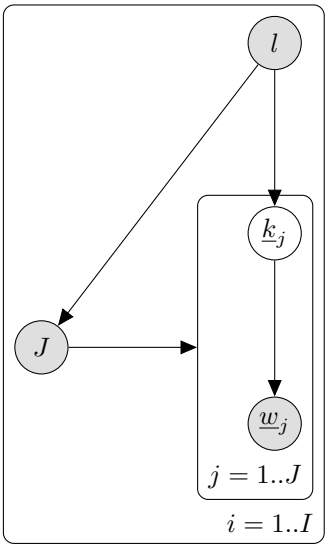


Figure 4.3: Graphical model for the Extended MNB, with label-conditional document length modeling, but without scaling of the factors for $p(l)$ and $p(J|l)$.

The Extended MNB model with fixed scaling factors $\vartheta = 1$ and $\varsigma = 1$ constitutes a directed generative model, and the corresponding graphical model is illustrated in Figure 4.3. With scaling weights other than 1 and 0, the parameter estimates of the scaled factors no longer derive from a directed factorization. One solution to formalize the model is to consider the combination of factors as a type of log-linear model combination [Klakow, 1998, Hinton, 2002, Bouchard and Triggs, 2004, Suzuki et al., 2007]:

$$p(\underline{\boldsymbol{w}}, l, \underline{\boldsymbol{k}}) = Z\, p'(l)^{\vartheta}\, p'(J|l)^{\varsigma}\, p(\underline{\boldsymbol{w}}, \underline{\boldsymbol{k}}|\, l, J) \tag{4.24}$$

With the log-linear model of Equation 4.24, the factors for label priors, document lengths and label-conditional word distributions become feature functions combined using the log-linear weights ϑ and ς. Maximum likelihood estimation of the model becomes complicated. One approximate solution is to keep the maximum likelihood estimates for the directed model, and directly optimize the new parameters ϑ and ς for a performance measure on held-out development data [Metzler and Croft, 2005]. This approximation is used for the experiments conducted in Chapter 6. It maintains the simplicity of directed models, while allowing the additional parameters to improve model performance.

Chapter 5

Sparse Inference

This chapter proposes computation based on sparse model representations for scalable inference. This reduces the time and space complexity of inference for a variety of linear models and structural extensions. First a basic case of sparse posterior inference is derived for uses such as Multinomial Naive Bayes (MNB) classification, ranking and clustering. This is extended into a more general case of joint inference on hierarchically smoothed sequence models, and further into joint inference on mixtures of such models. Additional efficiency improvements for the inference are discussed, and a structural extension of MNB benefiting from sparse inference is proposed, called Tied Document Mixture.

5.1 Basic Case: Sparse Posterior Inference

Once estimated, a generative model such as Multinomial Naive Bayes (MNB) can be used for several types of inference, depending on the task and application. The most common types are classification in text categorization, ranking in information retrieval, soft classification in document clustering, and model combination. All of these perform inference related to the posterior $p(l|\boldsymbol{w})$. In classification, the most likely label to generate a document is selected: $\text{argmax}_l\, p(l|\boldsymbol{w})$. In ranking, scores are computed for each label: $y(\boldsymbol{\theta}_l, \boldsymbol{w}) \stackrel{rank}{=} p(l|\boldsymbol{w})$. In soft classification and model combination, the posterior is used directly.

The time and space complexity of the inference is crucial in many applications of MNB, since this determines the scalability of the model to large-scale tasks. The numbers of labels, words and training documents can exceed millions in many practical tasks involving web-scale text datasets. The required computation can be reduced depending on the type of inference. For exam-

ple, for posterior inference the multinomial normalizer can be omitted, since $p(l|\boldsymbol{w}) = \frac{Z\, p(l) \prod_n p(w_n|l)}{\sum_l Z\, p(l) \prod_n p(w_n|l)} = \frac{p(l) \prod_n p(w_n|l)}{\sum_l p(l) \prod_n p(w_n|l)}$.

A naive algorithm for performing posterior inference with MNB computes $p(l|\boldsymbol{w}) = \frac{p(l,\boldsymbol{w})}{\sum_{l'} p(l',\boldsymbol{w})} = Z\, p(l) \prod_{n:w_n \neq 0} p_l(n)^{w_n}$ for each label l by computing the joint probabilities $p(l, \boldsymbol{w})$ as a product over terms n for each label l, and normalizing by the sum over the joints $Z = \sum_{l'} p(l', \boldsymbol{w})$. This has complexity $O(||\boldsymbol{w}||_0 L)$, and this is widely considered to be optimal for linear models [Manning et al., 2008]. However, the time complexity for posterior inference can be substantially reduced by taking into account sparsity in the parameters. Earlier work with inverted indices has shown that classifier scores for Centroid classifier [Shanks et al., 2003] and K-nearest Neighbours [Yang, 1994] can be done as a function of sparsity. The posteriors for MNB with uniform priors can be computed similarly [Hiemstra, 1998, Zhai and Lafferty, 2001b], for both Jelinek-Mercer [Hiemstra, 1998] and other basic smoothing methods [Zhai and Lafferty, 2001b]. The *sparse inference* proposed here generalizes this inference to MNB with categorical priors, a variety of linear models and structural extensions. A basic sparse posterior inference algorithm for MNB is described next. It combines three techniques for efficient computation:

Log-domain computation refers to use of log probability values, a standard practice with large-scale statistical models. This changes products into sums and exponents into products in the log-domain, which are much cheaper to compute with modern processors. For example, $\log(p(l)p_l(\boldsymbol{w})) = \log(p(l)) + \log(p_l(\boldsymbol{w}))$ and $\log(p_l(n)^{w_n}) = \log(p_l(n))w_n$. Another benefit of log-computation is reducing numerical inaccuracy that comes from computing with small probabilities. Alternatives such as scaling [Rabiner, 1989] for correcting inaccuracy with large-scale models are less useful in general.

Precomputing organizes an algorithm and necessary data structures so that less computation is necessary when the algorithm is used. A basic sparse posterior inference algorithm computes first the smoothing distribution probabilities $p^u(n)$, and then updates these for each l by multiplying with $\frac{(1-\alpha_l)p_l^u(n)+\alpha_l p^u(n)}{p^u(n)}$. The original parameter values $p_l^u(n)$ can be replaced by precomputed ones, since for computing $p(l, \boldsymbol{w})$, $p_l^u(n)$ are only needed for updating $p^u(n)$ for each l. The parameters for $p(l)$, $p^u(n)$, α_l and $p_l^u(n)$ can be replaced by the precomputed log values $p'(l) = \log p(l)$, $p'^u(n) = \log(p^u(n))$, $\alpha'_l = \log(\alpha_l)$, and $p'^u_l(n) =$

$\log((1-\alpha_l)p_l^u(n) + \alpha_l p^u(n)) - \alpha'_l - p'^u(n)$ for all $p_l^u(n) > 0$.

Inverted index data structures have formed the core technique of information retrieval for the last decades [Zobel and Moffat, 2006]. An inverted index uses a vector $\boldsymbol{\zeta}$ of *postings lists* for each word, so that for each word a list $\boldsymbol{\zeta}_n$ of occurrence information can be accessed in constant time. For sparse inference with MNB, it is sufficient to maintain the label and parameter in the postings, so that each posting is a pair $(l, p_l'^u(n))$. $p_l(n)$ can first be computed by $p^u(n)$, and updated for each l with $p_l^u(n) \neq 0$. Using an inverted index, the set of parameters to update can be retrieved with a time complexity $O(|\boldsymbol{w}|_0 + \sum_n \sum_{l:p_l^u(n)\neq 0} 1)$ instead of $O(L|\boldsymbol{w}|_0)$.

A basic sparse posterior inference algorithm computes $p'^u(\boldsymbol{w})$, updates this for all l with $p'(l)$ and $p_l'^u(\boldsymbol{w})$, and normalizes to obtain the posterior $p(l|\boldsymbol{w})$. This has the time complexity $O(L + |\boldsymbol{w}|_0 + \sum_n \sum_{l:p_l^u(n)\neq 0} 1)$. Using the precomputed values, we can compute joint probabilities as $p(\boldsymbol{w}, l) \propto \exp(p'(l) + \sum_n w_n(p'^u(n) + \alpha'_l) + \sum_{n:p_l^u(n)\neq 0} w_n p_l'^u(n)))$. The posterior can be computed by normalizing $p(l|\boldsymbol{w}) = p(\boldsymbol{w}, l)/\sum_{l'} p(\boldsymbol{w}, l')$. A key algorithmic idea here is that the conditional probabilities $p(\boldsymbol{w}|l)$ decompose into separately computed terms for the smoothing distribution, the smoothing weight, and the label-conditional: $p(\boldsymbol{w}|l) \propto \exp(\sum_{n:w_n\neq 0} p'^u(n) + ||\boldsymbol{w}||_1 \alpha'_l + \sum_{n:p^u(n)\neq 0} p_l'^u(n))$.

Algorithm 5.1 Sparse Posterior Inference for MNB

```
smooth_logprob = 0
for all n : w_n ≠ 0 do
    smooth_logprob+ = p'^u(n) * w_n
for all l do
    logprobs_l = p'(l) + ||w||_1 α'_l + smooth_logprob
for all n : w_n ≠ 0 do
    for all (l, p_l'^u(n)) ∈ ζ_n do
        logprobs_l+ = p_l'^u(n) * w_n
normalizer = −1000000
for all l do
    normalizer = log(exp(normalizer) + exp(logprobs_l))
for all l do
    logprobs_l− = normalizer
return logprobs
```

Pseudo-code for the resulting sparse posterior inference algorithm returning $p(l|\boldsymbol{w})$ is given in Algorithm 5.1. Although the sparse inference algorithm is described here for MNB posterior inference, it is applicable to any linear model.

The inference algorithm with precompiled values corresponds to a sparsely computed dot product $y(\boldsymbol{\theta}, \boldsymbol{w}) = \theta_0 + \sum_{n=1}^{N} \theta_n w_n$, which is used for all linear classifiers, including Centroid, Perceptron, Logistic Regression and Support Vector Machine classifiers. Moreover, it can be generalized into structured models such as hierarchical mixture models. Therefore the textbook statement [Manning et al., 2008] on optimality of $O(||\boldsymbol{w}||_0 L)$ posterior inference is not only incorrect for MNB, but for machine learning methods in general.

5.2 Extension to Joint Inference on Hierarchically Smoothed Sequence Models

The sparse inference in Algorithm 5.1 can be extended to hierarchically smoothed models, while retaining the same benefits in computational complexity. As discussed in Chapter 3, there are many cases where multinomials and MNB models are extended so that the probabilities in a sequence model back off to less context-dependent models. A variety of hierarchically smoothed sequence models are used with text, such as interpolated n-gram models. The smoothing hierarchy can come from sources such as word clusters [Zitouni and Zhou, 2008], the local word context [Chen and Goodman, 1999], or collection structure [McCallum and Nigam, 1999, Zhang et al., 2002, Krikon and Kurland, 2011]. Any combination of hierarchies can equally be used for backing-off. For example, a label-conditional passage bigram could first back-off to a label-conditional passage unigram, then to a label-conditional document unigram, and finally to a collection unigram model.

Let M denote the depth of hierarchical smoothing for a sequence extension of MNB, so that the label-conditional probabilities for the model are smoothed by the $M-1$ back-off layers in the hierarchy, where each node for layer m is a categorical distribution that is used in the smoothing. An example of this would be a label-conditional n-gram model, where M is the n-gram order. Node for layer $m = 1$ corresponds to the root node of the back-off hierarchy, $m = M$ to the leaf node to be smoothed. The joint probability of a sequence $\underline{\boldsymbol{w}}$ and label l for the model becomes:

$$
\begin{aligned}
p(\underline{\boldsymbol{w}}, l) &= p(l) \prod_j p_l(\underline{w}_j | \underline{\boldsymbol{w}}) \\
&= p(l) \sum_{\underline{\boldsymbol{k}}} \prod_j p_l(\underline{k}_j)\, p^u_{l\underline{k}_j}(\underline{w}_j | \underline{\boldsymbol{w}})
\end{aligned}
$$

$$
\begin{aligned}
&= p(l) \prod_j \sum_m p_l(m) p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}) \\
&= p(l) \prod_j \sum_m (\prod_{m'=m+1}^{M} \alpha_{lm'} - \prod_{m'=m}^{M} \alpha_{lm'}) p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}), \qquad (5.1)
\end{aligned}
$$

where $p_l(\underline{w}_j|\underline{\boldsymbol{w}})$ is the smoothed label-conditional probability, $p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}})$ the component-conditional probabilities, and α_{lm} the back-off weight of component m. The conditioning variables l, m and $\underline{\boldsymbol{w}}$ for the probabilities can be extended to include more variables, or can be tied to reduce the number of variables. For a label-conditional n-gram these would be tied as $p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}) = p^u_{lm}(\underline{w}_j|\underline{w}_{j-m+1} \cdots \underline{w}_{j-1})$.

Sparsity can be utilized by storing the non-zero parameters for each node in a precomputed form, by first smoothing and then precomputing the parameters for each node. Smoothed parameters are computed first for the root node $m = 1$: $p''^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}) = p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}})$, and then for each $m > 1$ up to to the leaf nodes: $p''^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}) = \alpha_{lm} p''^u_{l(m-1)}(\underline{w}_j|\underline{\boldsymbol{w}}) + (1 - \alpha_{lm}) p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}})$. Precomputed smoothed log-parameters are computed starting from the leafs $p'^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}) = \log(p''^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}})) - \log(\alpha_{lm}\, p''^u_{l(m-1)}(\underline{w}_j|\underline{\boldsymbol{w}}))$ for the nodes $m > 1$, and finally for the root-node $m = 1$: $p'^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}) = \log(p''^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}))$. The smoothing weights and label priors can be precomputed: $\alpha'_{lm} = \log(\alpha_{lm})$ and $p'(l) = \log(p(l))$.

With the precomputed values, joint probabilities of sequences $p(l|\underline{\boldsymbol{w}}, l)$ can be computed scalably by utilizing sparsity. The joint probability in Equation 5.1 can be expressed in a factorized form as:

$$
\begin{aligned}
p(\underline{\boldsymbol{w}}, l) &= p(l) \prod_j \sum_m (\prod_{m'=m+1}^{M} \alpha_{lm'} - \prod_{m'=m}^{M} \alpha_{lm'}) p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}) \\
&= p(l) \prod_m \xi(l, m) \\
&= \exp(p'(l) + \sum_m \xi'(l, m)) \\
\xi'(l, m) &= \begin{cases} \sum_{j: p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}) \neq 0} p'^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}), & \text{if } m = 1 \\ |\underline{\boldsymbol{w}}|\, \alpha'_{lm} + \sum_{j: p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}) \neq 0} p'^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}), & \text{otherwise,} \end{cases} \qquad (5.2)
\end{aligned}
$$

where $\xi(l, m)$ are factors for nodes (l, m) explained in the following, and $\xi'(l, m) = \log(\xi(l, m))$.

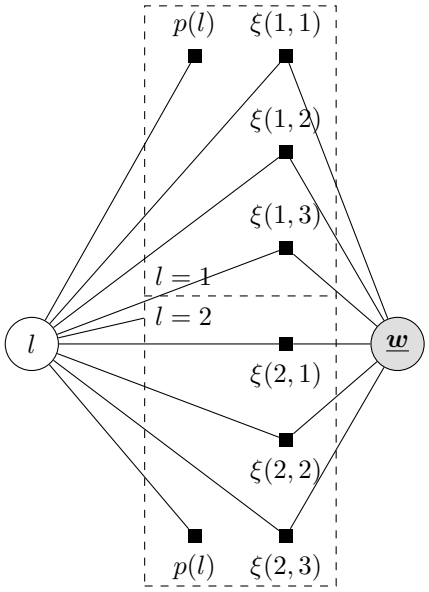
Equation 5.2 can be solved directly using dynamic programming. The fac-

tors $\xi(l,m)$ provide updates to computing $p(\underline{\boldsymbol{w}}|l)$ given its $m-1$ ancestor nodes. Starting from the root node, $\xi(l,m=1)$ computes $\log p(\underline{\boldsymbol{w}}|l)$ assuming that no descendant nodes exist. This is then updated iteratively by the descendant nodes $m>1$. The complexity of inference is reduced, because for each node only the non-zero unsmoothed parameters need to be considered, and these can be stored in postings lists retrieved from an inverted index. The time complexity is reduced from the dense $O(LMJ)$ to the sparse $O(L\,M+|\boldsymbol{w}|_0+\sum_l\sum_m\sum_{j:p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}})\neq 0}1)$.

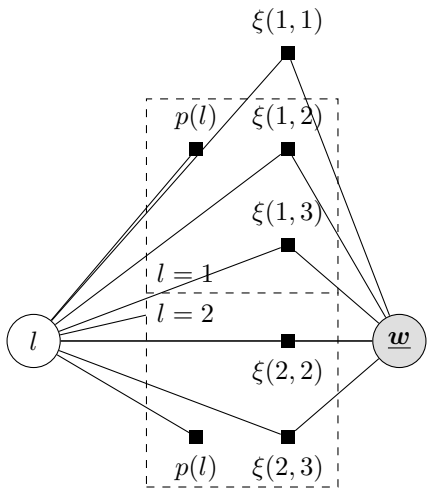
By using the shared hierarchical components between l and the other conditioning variables, this complexity can be further reduced. Since in a hierarchy the auxiliary variable values $\xi(l,m)$ are the same for all children of a node, $\xi(l,m)$ needs to be computed only once for each shared node $h=(l,m)$ in the hierarchy, and updated according to the children. Exact hierarchical computation reduces the complexity of computing $p(\underline{\boldsymbol{w}},l)$ for all l from $O(L\,M+|\boldsymbol{w}|_0+\sum_l\sum_m\sum_{j:p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}})\neq 0}1)$ to $O(L+|\boldsymbol{w}|_0+\sum_h(1+\sum_{j:p^u_h(\underline{w}_j|\underline{\boldsymbol{w}})\neq 0}1))$.

The hierarchical complexity can be further reduced in the case of constrained models or approximation. Assume that $p(l)$ are uniform in the following, and only the highest probability label is needed. If a node has no words that match the word sequence $\forall j:\underline{w}_j=0\vee p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}})=0$, then its update will be $\xi'(l,m)=|\underline{\boldsymbol{w}}|\,\alpha'_{lm}$. These nodes can be called the *no-match nodes* for a given word sequence. If Jelinek-Mercer smoothing is used for estimating α_{lm}, the back-off weights are the same for all children and $|\underline{\boldsymbol{w}}|\,\alpha'_{lm}$ can be added directly to the parent node $\xi'(l,m-1)$. Assuming matching nodes have no no-match ancestors, the sum $\sum_m\xi'(l,m))$ can be done only over the matching nodes m. Otherwise, a gap in the sum can be filled by computing $\xi(l,m-1)$ iteratively down to the first matching node. With the assumptions of no gaps in the hierarchy, Jelinek-Mercer for smoothing, and uniform $p(l)$, it suffices to compute the maximum probability l by $\operatorname{argmax}_l p(\underline{\boldsymbol{w}},l)=\operatorname{argmax}_l\sum_m\xi(l,m)$, where the sums $\sum_m\xi(l,m)$ can be done dynamically over the shared nodes h. The resulting time complexity is reduced to $O(|\boldsymbol{w}|_0+\sum_h\sum_{j:p^u_h(\underline{w}_j|\underline{\boldsymbol{w}})\neq 0}1)$, the sum of non-zero features and matching nodes.

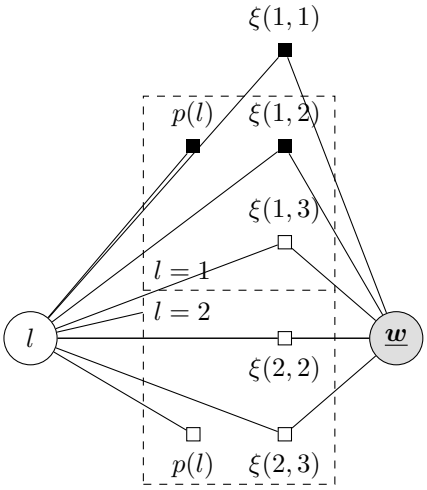
Figure 5.1 illustrates the resulting three types of sparse inference algorithms using factor graph notation. There are several other cases where the hierarchical time complexity can be reduced using constraints and approximation. If Dirichet prior or discounting methods are used for estimating α_{lm}, the back-off weights differ for the children. In this case the probabilities for the no-match children can be either computed exactly or approximated. A simple



(a) Sparse inference for each label l: $O(LM + |\boldsymbol{w}|_0 + \sum_l \sum_m \sum_{j:p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}})\neq 0} 1)$



(b) Hierarchical sparse inference with shared node $m = 1$: $O(L + |\boldsymbol{w}|_0 + \sum_h \sum_{j:p^u_h(1+(\underline{w}_j|\underline{\boldsymbol{w}})\neq 0} 1))$



(c) Constrained or approximate hierarchical sparse inference, with none of the nodes for label $l = 2$ matching the input: $O(|\boldsymbol{w}|_0 + \sum_h \sum_{j:p^u_h(\underline{w}_j|\underline{\boldsymbol{w}})\neq 0} 1)$

Figure 5.1: Factor graph models visualizing the sparse inference complexity reductions, for the same input word sequence $\underline{w}$ and model with $L = 2$, $M = 3$. White factors correspond to no-match nodes for the word sequence and do not need to be considered in inference. The complexity of inference within each factor further reduces complexity, and is not illustrated.

approximation is to precompute the mean back-off weight of the children for each node, and compute the probabilities for the no-match children using this mean weight. The priors $p(l)$ can be approximated by 0 for the labels that have only no-match nodes in the smoothing mixture. Using these two weaker approximations for α_{lm} and $p(l)$, the same complexity reduction can be obtained by computing $\text{argmax}_l\, p(\underline{\boldsymbol{w}}, l) = \text{argmax}_l \sum_m \xi(l, m)$ as before over the shared nodes. Other similar cases exist where the required computation is reduced by the number of no-match nodes. One practical case is ranked retrieval, where a set of top ranked labels for documents can be inferred given the input word sequence for a a query.

5.3 Extension to Joint Inference on Mixtures of Sequence Models

The sparse inference discussed so far has mainly dealt with the scalable computation of hierarchically smoothed sequence models. It can be extended to cases where mixtures are defined over the sequence models, as well as to cases where mixtures are used both within and over sequences. A basic implementation for inference with mixtures over sequence models would multiply the inference time complexity by the number of components, but the inference complexity can be reduced again using sparsity, constraints and approximation.

Similar to the mixture model view of MNB, we can consider the previous example of Equation 5.1 as a mixture model over the label-conditional hierarchically smoothed sequence models:

$$
\begin{aligned}
p(\underline{\boldsymbol{w}}) &= \sum_l p(\underline{\boldsymbol{w}}, l) \\
p(\underline{\boldsymbol{w}}, l) &= p(l) \prod_j \sum_m \Big(\prod_{m'=m+1}^{M} \alpha_{lm'} - \prod_{m'=m}^{M} \alpha_{lm'} \Big) p^u_{lm}(\underline{w}_j | \underline{\boldsymbol{w}}) \\
&= \exp(p'(l) + \sum_m \xi'(l, m)) \qquad (5.3)
\end{aligned}
$$

Lets assume a simple two-state HMM case of $M = 2$, where $m = 2$ is a label-independent background model and Jelinek-Mercer smoothing is used for selecting α'_{lM}. The factor for the smoothing background model becomes shared: $\forall_l : \xi'(l, m = 2) = \xi'(1, m = 2)$. Given a model where most of the leaf nodes $p^u_{lM}(\underline{w}_j | \underline{\boldsymbol{w}})$ are sparse, many of the labels l will have no-match leaf

nodes (l, M), so that $p(\underline{\boldsymbol{w}}|l) = \exp(\xi'(l = 1, m = 2) + |\underline{\boldsymbol{w}}|\ \alpha'_{lM})$. Let $L'(\underline{\boldsymbol{w}})$ indicate the number of no-match leaf nodes in Equation 5.3, l' labels with leaf nodes matching the word sequence $\exists j : \underline{w}_J > 0 \wedge p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}) > 0$, and α'' the back-off weight for the leaf nodes. The marginalization can be computed as:

$$p(\underline{\boldsymbol{w}}) = L'(\underline{\boldsymbol{w}})(1 - \sum_{l'} p(l')) \exp(\xi'(l = 1, m = 2) + |\underline{\boldsymbol{w}}|\alpha'') + \sum_{l'} p(\underline{\boldsymbol{w}}, l), \tag{5.4}$$

reducing marginalization time complexity from $O(L)$ to $O(L - L'(\underline{\boldsymbol{w}}))$.

Figure 5.2 illustrates the sparse marginalization using factor graphs. The complexity of marginalizations can be reduced in more complex cases as well, such as: different smoothing methods, deeper smoothing hierarchies, and multi-layer mixtures over the labels variables. If smoothing other than Jelinek-Mercer is used, the back-off weights of the no-match children can be approximated by a mean value, or grouped in bins to reduce the approximation. If deeper smoothing hierarchies are used, the marginalizations can be conducted iteratively for each layer m from M to 1. If multi-layer mixtures over the label variables are used, the marginalizations can be conducted similarly by iterating from the leaf nodes to the root.

5.4 Further Specialized Efficiency Improvements for Sparse Inference

Sparse inference can be made substantially more efficient for many uses. Combination with parallelization and stream processing is trivial, as subsets of the precomputed parameters $p'(l)$ and $\xi'(l, m)$ in Equation 5.2 can be stored and processed by separate computing nodes, each node containing a shard of the full inverted index for the parameters. Aside from the generic methods for improving efficiency discussed earlier in Chapter 2, methods more specific to text mining can be applied. For ranking or classification only a subset of the labels is required, and therefore a number of further efficiency improvements are possible. These can be categorized as within-node pruning, between-node pruning, and search network minimization:

Within-node Pruning Computing a factor score $\xi(l, m)$ can be halted if it is unlikely to affect the result. If classification or ranking with top-scoring labels is required, a ranked list of the top-scoring labels and their scores

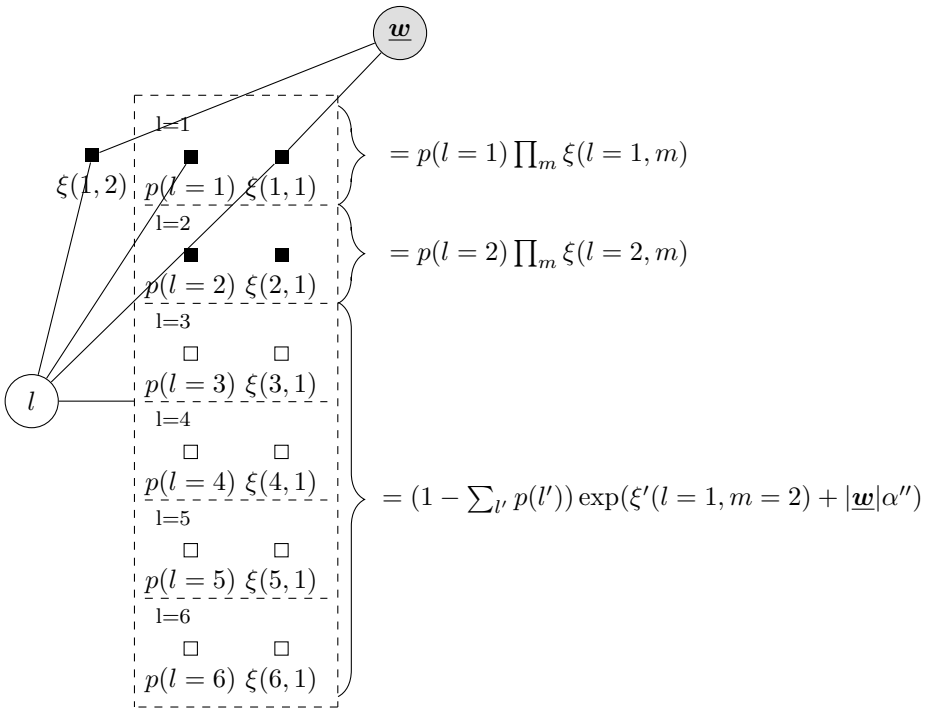


Figure 5.2: Sparse marginalization over l illustrated using a factor graph for the hierarchically smoothed sequence model $p(\underline{\boldsymbol{w}}) = \sum_l p(l) \prod_j \sum_m p_l(m) p^u_{lm}(\underline{w}_j|\underline{\boldsymbol{w}}) = \sum_l p(l) \prod_m \xi(l,m)$, with $L = 6$, $M = 2$ and component $m = 1$ shared between labels. Leaf nodes for labels $l > 2$ are no-match nodes and the white colored factors for these labels can be computed sparsely. For readability, factor edges for labels $l > 1$ are not shown.

can be maintained, and evaluation of each label can be terminated once it cannot reach the top labels. In information retrieval, heuristics using top-ranked lists form one of the main techniques for improving search efficiency, one notable algorithm being the MaxScore query evaluation algorithm [Strohman et al., 2005]. The postings lists can be sorted in an order such as decreasing magnitude of the parameters, enabling pruning of labels as early as possible. The inverted index can likewise be split into several indices, so that the apriori most likely labels are fully evaluated first; the labels in the following indices can be evaluated with much tighter pruning bounds.

Between-node Pruning Computing the factor score $\xi(l, m)$ can be avoided altogether, if it is likely (or bound) to have no effect on the result. If top-scoring labels are requested instead of the full posterior, the search can be organized hierarchically. Tree-based searches are commonly used with instance-based learning algorithms such as k-nearest neighbour clas-

sifiers. In tree-based search a hierarchy is constructed based on clustering label similarities, and a branch-and-bound search is used to retrieve the top labels [Ram and Gray, 2012]. Algorithm 5.1 can be extended into tree-based search by splitting the index into indices for each layer in a clustered label hierarchy, and skipping a branch in the evaluation if their higher-level node does not reach the current score bound. This performs between-node pruning, since nodes are discarded even before their scoring is considered. Hierarchical top-ranking could be combined with hierarchical smoothing either separately or together, so that the same hierarchy would be used both for smoothing models and for scoring the labels efficiently.

Search Network Minimization Search network minimization is used in complex graph search problems such as speech recognition systems [Aubert, 2002]. For example, Finite State Machines and Hidden Markov Models are minimized by combining nodes where possible. Examples of this would be merging all nodes with a single child with the child node, or approximating similar nodes with a single clustered node. Different types of minimizations of search networks can be used, if a graph of inverted indices is used for structured sparse inference.

5.5 Tied Document Mixture: A Sparse Generative Model

The sparse inference algorithms described in this chapter provide improved scalability for a variety of structured models. Mixture variable nodes can be introduced at any structural level of a model, while incurring only modest increases in computational requirements. The nodes can be organized in layers, or form any type of a directed acyclic graph. The nodes can correspond to linguistic units, document structure, or any available metadata. Layers of nodes can include: subword units, phrases, sentences, passages, sections, fields, documents, labels, and labelsets.

Considering MNB and multinomial models for text, the most practical inclusion for most modeling purposes is a document node. These models are commonly used on datasets stored in the form of word vectors, where other metadata is not available, has been discarded, or varies between tasks and datasets. Document identifiers, in contrast, are available in every application

of these models. MNB and multinomial models are usually estimated by either averaging the document counts, or treating all documents as a single large document. Both types of modeling are unnatural in the sense that they assume the documents to be identically distributed for a given label. This is a strong assumption, and one that does not generally hold with text data [Puurula and Myaeng, 2013]. Explicitly modeling the document variables avoids this problem.

Introducing a document node to MNB between the label and label conditional probabilities would in a general case increase the computational requirements considerably. Document mixture models [Novovicova and Malik, 2003, Nigam et al.] introduce a mixture over documents and learn the soft assignments of documents to mixture components using the EM algorithm. This requires the estimation of the number of components, the component weights and the component-conditional probabilities. In general, optimizing the number of components has to be done by evaluation on development data, while optimizing the other introduced parameters with EM will produce a local maximum of the model likelihood. The overall estimation is therefore approximate, and the time complexity is multiplied by the number of restarts used to find the number of components and to avoid local minima.

The use of approximate estimation with iterative algorithms can be avoided by constraining the mixture over documents in a suitable way. One such way is to use a mixture with a component assigned for each document. This constrains the number of components to the number of documents for the label, the component assignments to the documents, and the component-conditional probabilities to document-conditional probabilities. This type of extension of MNB was proposed as Tied Document Mixture (TDM) [Puurula and Myaeng, 2013]. In addition to these modeling choices, TDM smoothes the document models hierarchically, and uses a uniform distribution over the document component weights.

The original version of TDM presented used simple hierarchical Jelinek-Mercer smoothing [Puurula and Myaeng, 2013]. With the theory of smoothing presented in Chapter 4, more refined smoothing methods can be attempted. The earlier version was also presented in word vector form, whereas here it can be presented in the word sequence form together with the corresponding directed graphical model. Formally the TDM model takes the form:

$$
\begin{aligned}
p(\underline{\boldsymbol{w}}, l) &= p(l) \sum_{i \in I_l} p_l(i) \prod_j p_{li}(\underline{w}_j) \\
&= p(l) \frac{1}{|I_l|} \sum_{i \in I_l} \prod_j p_{li}(\underline{w}_j) \\
&= p(l) \frac{1}{|I_l|} \sum_{i \in I_l} \prod_j \sum_m p_{li}(m) p_{lim}(\underline{w}_j) \\
&= p(l) \frac{1}{|I_l|} \sum_{i \in I_l} \prod_n \sum_m p_{li}(m) p_{lim}(n)^{w_n}, \qquad (5.5)
\end{aligned}
$$

where $p(l)$, $p_l(i)$, $p_{li}(m)$ and $p_{lim}(\underline{w}_j)$ are all categoricals, and I_l indicates the set of documents corresponding to label l in the collection. Since a single document exists for each document indicator i, $p_l(i) = \frac{1}{|I_l|}$.

The document models $p_{li}(\underline{w}_j) = \sum_m p_{li}(m) p_{lim}(\underline{w}_j)$ are hierarchically tied for smoothing with four levels of nodes: $m = 1$ uniform background model, $m = 2$ collection model, $m = 3$ label-conditional categoricals, and $m = 4$ document-conditional categoricals. The original TDM used hierarchically Jelinek-Mercer smoothed document models, with a uniform distribution for the root-level smoothing [Puurula and Myaeng, 2013]. An extension can be attempted so that both the document- and label-conditional models are smoothed with any of the methods described in Chapter 4. The document models are defined as:

$$
\begin{aligned}
p_{li}(\underline{w}_j) =& \sum_m p_{li}(m) p^u_{lim}(\underline{w}_j) \\
=& p_{li}(m = 4) p^u_{li}(\underline{w}_j) + p_l(m = 3) p^u_l(\underline{w}_j) \\
&+ p(m = 2) p^u(\underline{w}_j) + p(m = 1) U(\underline{w}_j) \qquad (5.6)
\end{aligned}
$$

The label-conditional and collection models are estimated by discounting and normalizing averaged normalized document counts:

$$
p^u_l(\underline{w}_j = n) \propto \max(0, (\sum_{i \in I_l} \frac{w^{(i)}_n}{|\boldsymbol{w}^{(i)}|_1}) - D(l, n)) \qquad (5.7)
$$

$$
p^u(\underline{w}_j = n) \propto \sum_l \max(0, (\sum_i \frac{w^{(i)}_n}{|\boldsymbol{w}^{(i)}|_1}) - D(l, n)), \qquad (5.8)
$$

where $D(l, n)$ is the given by the chosen discounting method for the hierarchy level, if any.

Normalized counts $E(C(l,n)) = \sum_i \frac{w_n^{(i)}}{|\boldsymbol{w}^{(i)}|_1}$ are treated as expected fractional counts for determining the smoothing and discounting values. Length normalization is used for the background models, to reduce the effect of untypically long documents on mean statistics. The weights for the mixture components are computed dynamically from backoff-weights α for the different levels in the hierarchy: $p_{li}(m) = \prod_{m'=m+1}^{M} \alpha_{lim'} - \prod_{m'=m}^{M} \alpha_{lim'}$, where each $\alpha_{lim'}$ is produced by the smoothing method, as described in Chapter 4.

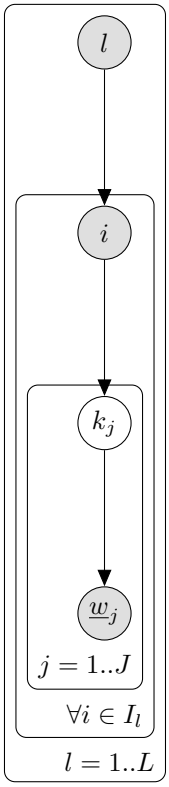


Figure 5.3: Directed graph for the Tied Document Mixture

Figure 5.3 shows a directed graph for TDM. Estimating the smoothed model parameters has $O(\sum_i |\boldsymbol{w}^{(i)}|_0)$ time and space complexity. Using sparse inference, time and space complexity of approximate hierarchical inference for classification is $O(|\boldsymbol{w}|_0 + \sum_l \sum_{i \in I_l} \sum_{n: p_{li}^u(n) \neq 0} 1)$. Complexities of exact inference can be increased a little by the smoothing and type of inference. Computing the prior probabilities $p(l)$ exactly adds a L term to the inference complexities, and using discounting or Dirichlet prior smoothing exactly adds a L term for the label nodes and I for the document nodes.

The uniform distribution over document components can be seen as a type of kernel density model, where hierarchically smoothed multinomials are used

instead of a Parzen kernel density using Gaussians [Parzen, 1962]. The hierarchical smoothing can be formalized as a mean-shift [Fukunaga and Hostetler, 1975] or data sharpening [Choi and Hall, 1999] method, which shifts the document-conditional models closer to label-conditional models. Under this view, the classifier is a form of a Kernel Density Classifier [Specht, 1988, John and Langley, 1995, Pérez et al., 2009]. However, this kernel density formalization is complicated. The smoothed multinomial kernels are discrete, asymmetric, multivariate, bounded kernel functions, as well as local for each class. Each of these properties is treated as a deviation to a standard Parzen kernel density, and there is no known prior work on multinomial or class-smoothed local kernel densities.

Chapter 6

Experiments

This chapter presents a large-scale evaluation of the methods developed in the thesis, showing improvements in both efficiency and scalability. A unified experimental framework is proposed for evaluation of classification and ranked retrieval, as well as optimization of model parameters with Gaussian random searches on development data. The performance measures, baseline methods, and statistical significance measurement across collections are discussed. The set of text collections is described, along the chosen preprocessing, segmentation and statistics. Five sets of experiments are conducted, demonstrating considerable improvements from the methods developed in the thesis.

6.1 Methodology

6.1.1 Experimental Framework

Text mining (TM) applications are commonly decomposed into tasks solved using the methods of machine learning and statistics. The general task types are classification, ranking, clustering, regression and sequence labeling, of which the first two have been most extensively researched. Classification applications include spam classification, sentiment analysis, web page classification, and classification of documents into ontologies. Ranking is mostly applied for information retrieval (IR), where linear models implemented with inverted indices form the basis of modern web search engines. This chapter empirically explores a number of hypotheses drawn from the theory developed in Chapters 4 and 5. Experiments are conducted on ranking and classification datasets using the standard linear models for these tasks as baselines.

The experiments explore the following five research questions:

Common formalization for smoothing Chapter 4 formalized the various

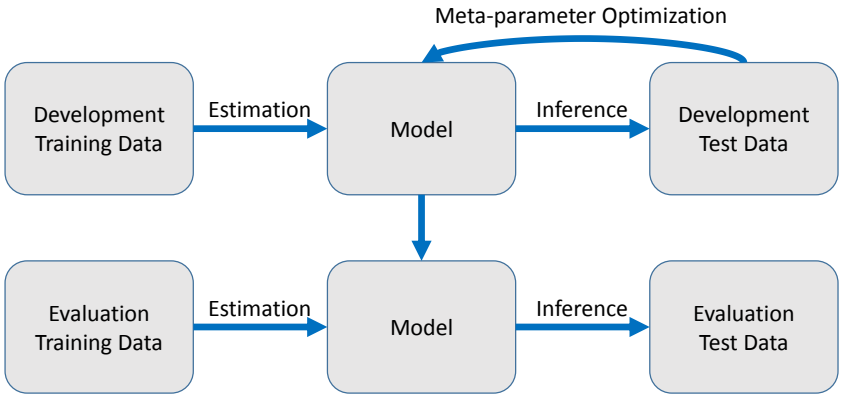


Figure 6.1: The common framework for optimization and evaluation used for the experiments.

smoothing methods for generative models of text. Does this common formalization improve the effectiveness the models used in the basic TM tasks?

Feature weighting and Extended MNB Chapter 4 formalized weighting features for Multinomial Naive Bayes (MNB) and an extension of MNB that includes scaling of the prior and length modeling. Do these types of modifications provide improvements in effectiveness?

Structured generative models Chapter 5 introduced models that add constrained mixture modeling structure into MNB. Do models of this type improve over MNB and Extended MNB models considerably?

Strong linear models baselines Chapters 4 and 5 propose generative models in a common framework for TM tasks. Do the proposed models improve over strong baselines for these tasks?

Scalability of sparse inference Chapter 5 introduced the idea of performing scalable probabilistic inference with generative models using inverted indices. How scalable are generative models utilizing sparse inference, compared to discriminative linear models and generative models implemented without inverted indices?

A common machine learning framework is applied to ranked retrieval and classification, illustrated in Figure 6.1. For both cases datasets are segmented into a development training set D^d, a development test set D^{d*}, an evaluation training set D^e, and an evaluation test set D^{e*}. The development sets are used for random search optimization of the meta-parameters, such as the smoothing

parameters. The evaluation sets are used to produce performance measures for the evaluated models, which are then tested for statistically significant differences between the models across the datasets.

Within this framework, text classification and ad-hoc ranked retrieval tasks have fundamental differences only in how the datasets are organized. Classification operates on documents where both the training $\boldsymbol{w}$ and test set $\boldsymbol{w}^*$ documents are distributed in a similar fashion. Ad-hoc ranked retrieval refers to test set documents as queries, and to training set documents simply as documents. The queries often form word vectors much shorter than the retrieved documents, consisting of only a few keywords, a sentence, or a title indicating the search intent. The labels for classification datasets are distributed similarly between the train and test documents. The labels for ranked retrieval are document identifier variables, each training document corresponding to a single unique identifier, whereas queries correspond to multiple document labels that have been judged relevant to the query. The labels for multi-label classification can be described as binary indicator vectors $\boldsymbol{c}$, $\forall l : c_l \in (0, 1)$, constrained to $\sum_l c_l = 1$ for multi-class classification, and further $L = 2$ for binary-label classification. The labels for ranked retrieval can be described as either binary indicator vectors, or integer vectors when graded judgments are available for the queries. Aside from these fundamental differences in terminology and organization of data, ranked retrieval and text classification can be considered in the same experimental framework.

Multi-label classification tasks are converted into multi-class problems by using the Label Powerset method [Boutell et al., 2004]. This maps each unique label vector seen in training data into a categorical labelset variable, and maps the labelset variables back into label vectors after classification. This simplifies model learning, but also increases the number of possible label variables in learning. Learning and optimizing meta-parameters for discriminative models for the large-scale multi-label datasets used in the experiments is not computationally feasible within practical times with Label Powerset or other basic transformations of multi-label learning, and the results for these have not been computed.

6.1.2 Performance Measures

There are several commonly used performance measures for both ranking and classification. Classification measures need to consider the unbalanced label distributions common with text data: most labels have few associated docu-

ments, while most documents are labeled with one of the most common labels. Ranking measures need to consider the priority of ranking the top ranked labels accurately, compared to ranking all labels accurately. For the experiments conducted in this thesis, Micro-averaged F-score (Micro-F1) is used for evaluating classification, and Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain of top 20 documents (NDCG@20) are used to evaluate ranking. These measures are described in the following.

For many classification tasks, F1-scores form the basis of the common evaluation measures. The F1-score is the harmonic mean of the precision and recall. Given binary label vectors of reference labels $\boldsymbol{c}$ and predicted labels $\hat{\boldsymbol{c}}$, precision is defined as the number of true positives TP divided by the number of predicted positives PP: Precision = TP/PP, where $\mathrm{TP} = \sum_{l:c_l=1 \wedge \hat{c}_l=1} 1$ and $\mathrm{PP} = \sum_l \hat{c}_l$. Recall is defined as TP divided by the number of reference positives RP: Recall = TP/RP, where $\mathrm{RP} = \sum_l c_l$. With these definitions, the F1-score for a single test document can be defined as:

$$\mathrm{F1} = 2\,\frac{\mathrm{Recall} \cdot \mathrm{Precision}}{\mathrm{Recall} + \mathrm{Precision}} \tag{6.1}$$

The Recall, Precision and F1-score measures have been developed in the context of IR, where the measures are commonly averaged across queries to produce corresponding mean measures. Although Mean-F1 averaged across test label vectors $\boldsymbol{c}$ can be used for classification, the label imbalance in classification has made other types of averaged measures popular [Tsoumakas et al., 2010]. With *macro-averaging* a mean F1 is first computed for each label independently, and these are averaged to produce the Macro-F1 measure. With *micro-averaging*, the statistics of Recall and Precision are computed across the labels, and the Micro-F1 measure is then computed from the micro-averaged Recall and Precision statistics:

$$\text{Micro-F1} = 2\,\frac{\text{Micro-recall} \cdot \text{Micro-precision}}{\text{Micro-recall} + \text{Micro-precision}} \tag{6.2}$$

$$\text{Micro-precision} = \frac{\sum_{(\boldsymbol{c},\hat{\boldsymbol{c}})} \sum_{l:c_l=1 \wedge \hat{c}_l=1} 1}{\sum_{(\boldsymbol{c},\hat{\boldsymbol{c}})} \sum_l \hat{c}_l} \tag{6.3}$$

$$\text{Micro-recall} = \frac{\sum_{(\boldsymbol{c},\hat{\boldsymbol{c}})} \sum_{l:c_l=1 \wedge \hat{c}_l=1} 1}{\sum_{(\boldsymbol{c},\hat{\boldsymbol{c}})} \sum_l c_l} \tag{6.4}$$

Micro-F1 is one of the most commonly used measures for multi-label text classification. Unlike Micro-F1, Macro-F1 is strongly affected by label imbalance: a label occurring a thousand times has the same weight as one that occurs a single time. Micro-F1 for binary-label and multi-class tasks is also equivalent to mean Accuracy: the proportion of correct classifications [Manning et al., 2008]. This makes Micro-F1 scores for these tasks directly comparable with much of the earlier research literature.

Precision forms the basis for MAP, the most common measure used for evaluating ranking in IR systems. MAP is computed as a mean of averaged Precision values over possible levels of Recall. Let $\hat{\boldsymbol{y}}$ denote a vector of ranked and sorted scores for predicted labels l and $\boldsymbol{y}$ a matching vector of binary reference scores. MAP is computed as:

$$\text{MAP} = \frac{1}{\sum_k y_k} \sum_{(\boldsymbol{y},\hat{\boldsymbol{y}})} \sum_{k=1}^{L} y_k \text{Precision}(k, \boldsymbol{y}) \tag{6.5}$$

$$\text{Precision}(k, \boldsymbol{y}) = \frac{\sum_{k'=1}^{k} y_{k'}}{k} \tag{6.6}$$

It can be seen that MAP ignores the actual predicted scores $\hat{\boldsymbol{y}}$, and uses only binary reference labels. MAP has since been supplemented with more sophisticated measures, such as the NDCG [Järvelin and Kekäläinen, 2002, Wang et al., 2013]. For NDCG the sorted reference scores $\boldsymbol{y}$ can take graded values. Standard NDCG@k is computed as:

$$\text{NDCG}(k) = \frac{1}{|D^*|} \sum_{(\boldsymbol{y},\hat{\boldsymbol{y}})} Z(k) \sum_{k'}^{k} \frac{2^{y_{k'}} - 1}{\log_2(k'+1)} \tag{6.7}$$

$$Z(k) = \max_{\boldsymbol{y}'} \left(\sum_{k'}^{k} \frac{2^{y'_{k'}} - 1}{\log_2(k'+1)} \right) \tag{6.8}$$

The normalizer $Z(k)$ is the maximum of possible Discounted Cumulative Gains (DCG), giving the DCG for the best possible ranking with k ranked labels, and normalizing NDCG for each test document to a maximum score of 1. Smaller values of k are more effective in discriminating against ranking functions, but become less robust [Wang et al., 2013]. Larger values of k are less brittle, but become less effective in discriminating ranking function performance. The value $k = 20$ is used in the ranking experiments of this thesis.

Absolute differences in basic measures such as Micro-F1, NDCG@20 and MAP are affected by both the baseline model and the dataset. An improvement over a weak baseline or an easy dataset gives larger absolute differences than improvements over strong baselines and hard datasets. In text classification Micro-F1 is close to the perfect score of 1.0 for some tasks, whereas in text retrieval the reverse is the case, with MAP and NDCG results usually below 0.5.

The differences in Micro-F1 are further explored using Relative Error Reduction (RER), which is used in the evaluation of multinomial language models (LM) in the context of speech recognition and machine translation [Olive et al., 2011]. The RER can be defined as $1 - (f^{max} - f^{s_2})/(f^{max} - f^{s_1})$, where f^{max} is the maximum score, f^{s_1} the score for baseline model, and f^{s_2} the score for the new model. Differences in NDCG@20 and MAP are further explored using Relative Improvement (RI), defined as $1 - f^{s_2}/f^{s_1}$. The derived measure RI is often used implicitly for describing the differences in IR ranking performance [Zaragoza et al., 2007, Metzler et al., 2009]. Compared to absolute differences, relative differences are often more stable across varying baselines and datasets, and can give more intuitive measures of improvement.

6.1.3 Baseline Methods

Many of the leading solutions for TM tasks are instances of linear models, including those for classification and ranking tasks. For classification, linear classifiers such as MNB, Support Vector Machines (SVM), and Logistic Regression (LR) have become the most common solutions. For ranking, the linear models Vector Space Model (VSM), Best Match 25 (BM25), and LM are the standard methods for ranked text retrieval in IR. The connections between linear models for TM were discussed in Chapter 2, and Chapters 3 and 4 examined the MNB model in detail, showing that a common framework for generative models covers the LMs used for ranking as a special case of MNB. The experiments conducted in this chapter use SVM, LR and MNB as baselines for classification, and BM25, VSM and MNB as baselines for ranking.

As discussed in Chapter 2, the multi-class linear scoring function takes the form:

$$y(\boldsymbol{\theta}_l, \boldsymbol{w}) = \theta_{l0} + \sum_{n=1}^{N} \theta_{ln} w_n \tag{6.9}$$

The following gives the parameter estimates for each of the baseline models. For most models, results with and without Term Frequency - Inverse Document Frequency (TF-IDF) feature weighting are provided. TF-IDF modifies the training and test documents by applying the generalized TF-IDF presented in Chapter 4: $w_n = \log(1 + \frac{w'_n}{|\boldsymbol{w}'|_0^\phi})/|\boldsymbol{w}|_0^{1-\phi} \log(\max(1, \upsilon + \frac{I}{I_n}))$, where w'_n are the original counts, ϕ is the parameter for length scaling and υ is the parameter for IDF lifting. Depending on the model, these parameters are fixed or optimized. With fixed parameters, $\phi = 0$, and length normalization is done after log normalization of counts. The fixed value $\upsilon = 0$ is used for classification and ranking, producing Robertson-Walker IDF, whereas $\upsilon = -1$ is used for the scalability experiments, producing unsmoothed Croft-Harper IDF and sparsifying the feature vectors to improve scalability.

MNB and LM are generative probabilistic models based on a multinomial or first-order Markov chain distribution of words conditional on each label variable. An unsmoothed MNB or LM baseline model would have the parameter estimates $\theta_{ln} = \log \frac{\sum_{i:l^i=l} w_n^{(i)}}{\sum_i \sum_n w_n^{(i)}}$, while the bias is the log prior probability $\theta_{l0} = \log p(l)$ for MNB classification and uniform in the case of LM retrieval. The Dirichlet prior and Jelinek-Mercer smoothed MNB/LM models are used as basic baselines for classification and ranking. These modify the unsmoothed log label-conditional probability parameters θ_{ln} as described in Chapter 4. For both classification and ranking these baselines are compared with the proposed combinations of smoothing methods and extended generative models.

LR and SVM are discriminative linear classifiers, estimated with iterative algorithms that optimize a chosen loss function. With LR the loss function is derived from an underlying discriminative probabilistic model, whereas with SVM the loss function is non-probabilistic. L1 and L2-regularization with L2-SVM and LR are used as baselines for classification. Regularized LR and SVM models estimate the parameters θ_{ln} by optimizing $\theta_{ln} = \min_{\boldsymbol{\theta}} R(\boldsymbol{\theta}) + C \sum_i L(\boldsymbol{\theta}, D^{(i)})$, where $R(\boldsymbol{\theta})$ is the regularization, $L(\boldsymbol{\theta}, D^{(i)})$ is the cost function, and C is the regularization parameter. The loss function $L(\boldsymbol{\theta}, D^{(i)})$ for L2-SVM is $\max(0, 1 - l^{(i)} \boldsymbol{\theta}^T \boldsymbol{w}^{(i)})^2$, and $\log(1 + \epsilon^{-l^{(i)} \boldsymbol{\theta}^T \boldsymbol{w}^{(i)}})$ for LR. L1-regularization adds the term $R(\boldsymbol{\theta}) = |\boldsymbol{\theta}|_1$, while L2-regularization adds the term $R(\boldsymbol{\theta}) = \frac{1}{2}|\boldsymbol{\theta}|_2^2$. Here L2-SVM is optimized using a coordinate descent algorithm [Hsieh et al., 2008], and LR is optimized using a trust region Newton method [Lin et al., 2008].

The VSM and BM25 methods used for ranking have a number of variants.

VSM was initially defined as a cosine distance on word vector spaces [Rocchio, 1971], and this was later improved by applying TF-IDF feature transforms to training and test documents. Both of these basic models continue to be used in addition to more developed versions. The VSM models have the parameter estimates $\theta_{ln} = \frac{\sum_{i:l^i=l} w_n^{(i)}}{\sqrt{\sum_{n'}^{N}(\sum_{i:l^i=l} w_{n'}^{(i)})^2}}$, and the test document vector is L2-normalized: $w_n = \frac{w'_n}{|\boldsymbol{w}'|_2}$, where $\boldsymbol{w}'$ is the un-normalized vector. BM25 has been derived as an approximation to a probabilistic model, combined with a soft length normalization of counts [Robertson and Zaragoza, 2009]. The BM25 parameter estimates are $\theta_{ln} = IDF(n)\frac{(k_1+1)w_n^{(i=l)}}{LN(i)+w_n^{(i=l)}}$, and the test document counts are normalized $w_n = \frac{(k_3+1)w'_n}{k_3+w'_n}$, where w'_n are the original un-normalized counts [Manning et al., 2008, Robertson and Zaragoza, 2009]. The training document length normalization is given by $LN(i) = k_1((1-b) + b|\boldsymbol{w}^{(i=l)}|_1/A)$, with the average length $A = \sum_i |\boldsymbol{w}^{(i)}|_1/I$. The IDF for this standard BM25 is given by the smoothed Croft-Harper $IDF(n) = (I - I_n + 0.5)/(I_n + 0.5)$ [Manning et al., 2008]. BM25 does not benefit from a further TF-IDF feature transform, as it includes an IDF function and has implicit count and document length normalization.

6.1.4 Parameter Optimization

The meta-parameters required by the models and TF-IDF are optimized on a development set for each dataset. A common practice is to either use a grid search of parameter estimates, or heuristic values [Robertson and Zaragoza, 2009]. Neither of these is guaranteed to provide optimal performance, and the results produced by unoptimized models can be misleading. The experiments shown in this thesis use random search optimization of the meta-parameters, an approach that makes few assumptions about the optimized function and is efficient for the small-dimensional optimization problems encountered when optimizing TM linear model meta-parameters.

Grid search works by defining a grid of permissible parameter ranges ($\min_q$, $\max_q$) with small constant steps Δ_q for each meta-parameter q, such as increments of 0.1 from 0.0 to 1.0. There are two main problems with this. First, the number of points to sample in the parameter space is an exponential function the dimension Q of the parameter vector. With $Q = 2$ parameters, a grid with range from 0.0 to 1.0 and increments of 0.1 would require evaluation of 11^2 points, while with $Q = 5$ parameters the number of points would increase to $11^5 = 161051$. This makes grid search efficient only when there are few parameters. Second, if the steps for any of the parameters do not cover the

optimum value, the optimization fails. With new models and data the permissible ranges and steps are not well known, and grid search can miss the optimal values.

Direct search optimization [Powell, 1998], also known as metaheuristics [Luke, 2009] and black-box optimization, is a more complex method of parameter optimization. This seeks to optimize a function f using a limited number of point evaluations $f(\boldsymbol{a})$, when very little is known about the properties of the function, such as smoothness, unimodality or convexity. Direct search problems of different types are encountered in a number of scientific disciplines, and as a result hundreds of methods have been extensively investigated. Some commonly known cases are genetic algorithms and simulated annealing [Luke, 2009].

Random search [Favreau and Franks, 1958, White, 1971] offers a type of direct search algorithm that is well suited for the small-dimensional, non-smooth and multi-modal functions encountered with the linear models in TM. A random search operates by improving the currently best point $\boldsymbol{a}$ by randomly generating new points $\boldsymbol{d} = \boldsymbol{a} + \boldsymbol{\Delta}$ with steps $\boldsymbol{\Delta}$, bounding the points within the permissible ranges $\max_q$ and $\min_q$, and replacing $\boldsymbol{a}$ by $\boldsymbol{d}$ if the new point is good or better, i.e. $f(\boldsymbol{d}) \geq f(\boldsymbol{a})$. Generating the steps by a Gaussian distribution produces the Random Optimization algorithm [Matyas, 1965], which can be improved by several commonly used heuristics:

- Decreasing step sizes
  The step sizes can be gradually reduced by modifying the variance of the Gaussian distribution. The variance for each parameter can be initialized to be half the permissible range $0.5 \cdot (\max_q - \min_q)$, and multiplied by 0.9 after each iteration. This produces a log-curve decrease in step sizes, sampling most of the permissible ranges initially and searching locally later

- Multiple parallel steps
  The point evaluation can be parallelized, evaluating a subiteration of Q points simultaneously and choosing the best point $\boldsymbol{d}_q = \boldsymbol{a} + \boldsymbol{\Delta}_q$ as the new best point. This enables direct use of multiple processors for optimization, without any parallelization overhead

- Multiple best points
  In case of a tie, the best X points from each subiteration can be used to replace the current best point, and sampling can be done uniformly from the this set $\boldsymbol{a}_x$: $\boldsymbol{d}_q = \boldsymbol{a}_{q\%X} + \boldsymbol{\Delta}_q$. This enables the search to spread out

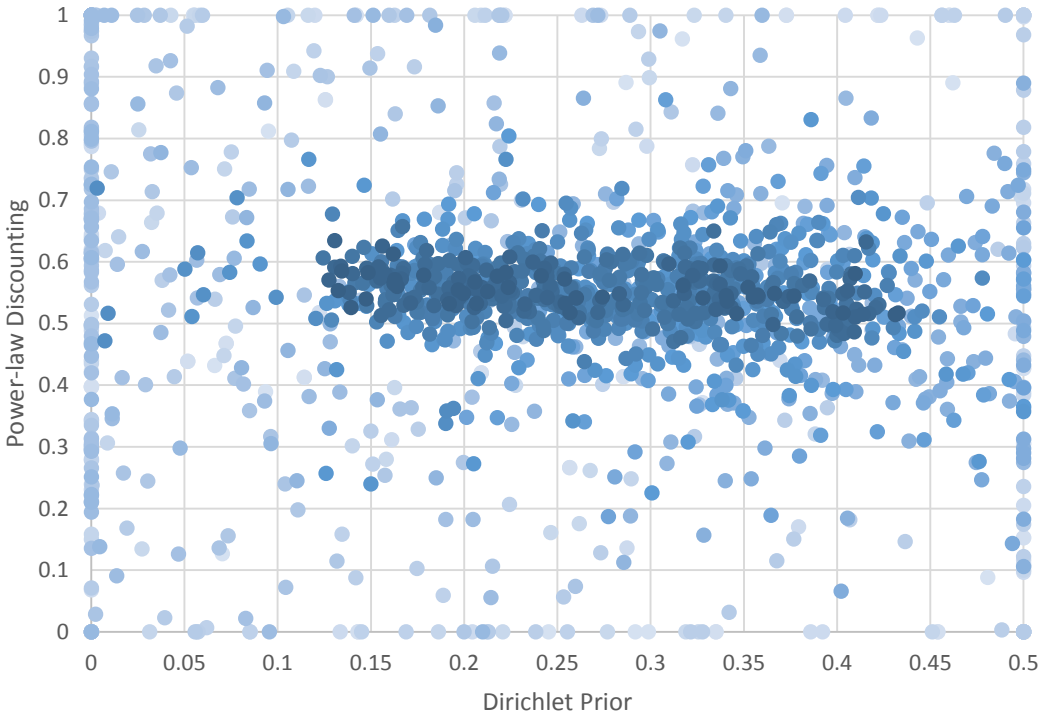


Figure 6.2: Evaluated points for a 40x40 random search, optimizing the Dirichlet prior and power-law discounting parameter for a MNB model on the webkb text classification dataset. Points color-coded from light to dark in the order of iterations. Light points show the global sampling done by the initial iterations, dark points in the center show the local search done by the last iterations. Dirichlet prior parameter is normalized by L for optimization, adding up to 0.5 to each count

to X different locations, in case a plateau is reached

All of the above modifications are common variations to random searches, and generally improve search efficiency without introducing additional flaws into random search, such as vulnerability to local optima. A parallelized random search of 50 iterations and 8 subiterations can be denoted a 50x8 random search. Figure 6.2 visualizes the evaluated points for a 40x40 search using this Gaussian random search. The models for retrieval were optimized with 50x8 searches, the models for classification with 40x40 searches. LR and SVM models were optimized with 40x8 searches, due to longer estimation times and the use of only up to three meta-parameters. For all model types, the random searches were iterated a few times with different permissible ranges.

6.1.5 Significance Tests

The parameters optimized on the development sets are used for measuring performance on the evaluation sets. Comparison of performance can be done within a dataset or between datasets. Within-dataset comparisons are more common when a limited number of datasets are available, as used to be the case in IR [Hull, 1993, Sanderson and Zobel, 2005, Smucker et al., 2007, Cormack and Lynam, 2007, Smucker et al., 2009]. Across-dataset comparisons are more common in fields where standardized datasets are publicly available, as is the case in machine learning [Dietterich, 1998, Demšar, 2006]. These have the important advantage of measuring performance on a group of datasets, avoiding problems encountered when using folds of the same dataset for testing significance [Demšar, 2006]. If the chosen datasets are distributed in the same way as a typical dataset for the task, then the discovered effects will hold on new datasets of the same task. Here a number of datasets have been segmented for both classification and ranking, and the methods are compared across the datasets.

Statistical significance of the evaluation set performance measures is assessed by performing one-tailed paired t-tests on the absolute between-dataset differences. The paired Student's t-test [Gosset, 1908] is a basic test for comparing differences, recently advocated for evaluation of IR results [Sanderson and Zobel, 2005, Smucker et al., 2007, Cormack and Lynam, 2007, Smucker et al., 2009]. It compares the means and variances of two groups of observations, and assumes both groups have a Gaussian distribution. The null-hypothesis for a t-test posits that the difference between the means of the two groups is the result of the Gaussian variance. If the difference exceeds what the variance allows, it is considered to be statistically significant. Significance is computed by comparing the t-value of the test to a t-distribution, and discarding the null-hypothesis if the t-statistic deviates too far from the t-distribution for the chosen p-value of statistical significance.

A paired t-test compares observations from matched pairs, so that instead of the difference between means, the mean of the paired differences is compared. This makes the test more powerful, because the variance caused by datasets is subtracted from the comparison. A one-tailed t-test compares only the difference in the t-statistic in a single direction, instead of deviation in both directions. This makes the test twice as powerful in p-value, with the prior assumption that variance in one direction will not be significant. Paired one-tailed t-tests are conducted with two significance levels on the absolute differences in the measures: 0.005 (†) and 0.05 (‡).

A problem with large-scale statistical testing is that the risk of false positive results is multiplied by the number of conducted comparisons. Modifying test statistics to penalize for the number of comparisons in turn weakens the significances of the individual comparisons. The strategy adopted here is to constrain significance tests only on the differences that attempt to answer the research questions set out in advance, at the beginning of the chapter. This both simplifies the description of findings, and reduces the risk of false positives. Nevertheless, the datasets in question have been used to iteratively develop the models, and a prior literature exists on the performance of the baseline linear models on these datasets. For these reasons the experiments and significance tests are *exploratory*, and *confirmatory* evaluation of the discovered effects is left for future research on new datasets. For further exploration and confirmation of the findings, the full evaluation set results are provided as tables in Appendix A.

6.2 Datasets

6.2.1 Dataset Overview

A total of 13 datasets are used for ranking and 16 for classification. Additional experiments are conducted on a large Wikipedia dataset, which allows the scalability of linear classifiers to be assessed when the numbers of documents, features and labels are each scaled up to a million. Aside from the TREC1-8 datasets for ranking, all datasets are publicly available for research use, and the pre-processing scripts and pre-processed datasets are made available[1].

The ranking datasets consist of the TREC 1-8[2] collections split according to data source into 11 datasets, OHSU-TREC[3] [Hersh et al., 1994, Robertson and Hull, 2001] and FIRE 2008-2011[4] datasets. TREC1-8 [Voorhees and Harman, 1999] contains the ad-hoc retrieval collections that were used in the 1990s to establish modern ranking functions and performance measures, most notably the BM25 ranking function [Robertson et al., 1996] and the MAP evaluation measure [Voorhees and Harman, 1999]. TREC was followed by other programs of for evaluating IR technology, such as NTCIR, INEX and FIRE. While the TREC ad-hoc track was suspended in 1999 in favor of more diverse

[1] http://www.cs.waikato.ac.nz/~asp12/
[2] http://trec.nist.gov/data/test_coll.html
[3] http://trec.nist.gov/data/t9_filtering.html
[4] http://www.isical.ac.in/~clia/

tasks, some of these can be considered collections for ad-hoc ranked retrieval. The OHSU-TREC collection is a publicly available dataset of medical articles from PubMed, used for TREC9. The FIRE 2008-2011 English collections were constructed to evaluate IR in the major languages spoken in India, following the TREC ad-hoc evaluation paradigm.

Six binary-label, five multi-class, and five multi-label datasets are used. The binary-label datasets are TREC06[5] [Cormack, 2006], ECUE1 and ECUE2[6] [Delany et al., 2006] for spam classification, and ACL-IMDB[7] [Maas et al., 2011], TripAdvisor12[8] [Bespalov et al., 2012], and Amazon12[8] [Blitzer et al., 2007] for sentiment analysis. These have been made available in the last 10 years, during which period spam classification and sentiment analysis became popular topics. The multi-class datasets are R8, R52, WebKb, 20Ng and Cade[9] [Cardoso-Cachopo, 2007]. These are older datasets, and versions of the first four provided the benchmarks used for early comparisons of text classification algorithms [Lewis, 1992, Joachims, 1998, McCallum and Nigam, 1998]. The multi-label datasets, which are more recent and large-scale, are RCV1-v2-Ind[10], EUR-Lex[11] [Mencía and Fürnkranz, 2010], OHSU-TREC[12] [Robertson and Hull, 2001], DMOZ2[13] and WikipMed2[13].

The large-scale Wikipedia dataset WikipLarge[13] was used in the Large Scale Hierarchical Text Classification (LSHTC) evaluation of scalable multi-label classification of articles into the Wikipedia categorization hierarchy. It has word vector features with millions of words, documents and labels. The scalable probabilistic models developed in this thesis were used to win the 2014 LSHTC competition [Puurula et al., 2014] on this data. For the systematic comparisons of scalability, the original dataset is pruned in terms of features, documents and labels, as described in the following subsections.

6.2.2 Preprocessing

Datasets were pre-processed from the original form into word vectors; however, some datasets are provided in word vector forms, making it impossible

[5]http://plg.uwaterloo.ca/~gvcormac/treccorpus06/
[6]http://www.dit.ie/computing/staff/sjdelany/Dataset.htm
[7]http://ai.stanford.edu/~amaas/data/sentiment/
[8]http://www.cs.virginia.edu/yanjun/paperA14/ecml12-cikm11-deepSC.htm
[9]http://web.ist.utl.pt/~acardoso/datasets/
[10]http://www.daviddlewis.com/resources/testcollections/rcv1/
[11]http://www.ke.tu-darmstadt.de/resources/eurlex
[12]http://trec.nist.gov/data/t9_filtering.html
[13]http://lshtc.iit.demokritos.gr

to perform identical processing steps. In all cases text was first lowercased. Next, stop-words were removed on most datasets, as well as short words (< 3 characters) and long words (> 20 characters). This was followed by Porter-stemming [Porter, 1980] of the remaining words. The scripts used for both pre-processing and segmentation are publicly available[14].

The binary-label datasets ecue1 and ecue2 are provided as pre-processed integer word vectors. Stop-word removal or stemming is not performed on the text [Delany et al., 2006], but documentation for any other pre-processing is not available. The trec06 dataset is provided as raw emails, including the metadata header. The header was removed, the remaining text was Porter-stemmed, and stop-words, short words, long words, non-words, and numbers were removed. The aclimdb dataset is provided in pre-processed form, in lowercase with numbers removed, but without stemming or removal of stop-words or non-words [Delany et al., 2006]. Opinion grades 1-4 and 1-7 were mapped to negative and positive labels, respectively [Maas et al., 2011]. The tripa12 and amazon12 datasets are provided in pre-processed form, with numbers replaced by "NUMBER", but without stemming, stop-word, short-word or non-word removal. Opinion grades 1-2 were mapped to negative label, 4-5 to positive [Bespalov et al., 2012].

The single-label datasets 20ng, cade, r52, r8, and webkb are provided as pre-processed word vectors. These use Porter-stemming, removal of 524 SMART stop-words, removal of short and truncation of long words [Cardoso-Cachopo, 2007]. Where available, titles of documents are concatenated to text bodies. Here no further processing was done aside from format conversion of the files.

The multi-label dataset rcv1 is provided as pre-processed word vectors, with punctuation removal, Porter-stemming and SMART stop word removal [Lewis et al., 2004]. The eurlex dataset is provided with lowercasing, Porter-stemming, stop-word removal and number removal [Mencía and Fürnkranz, 2010]. The ohsu-trec dataset is provided in the original OHSU-MED format [Hersh et al., 1994]. Here we use the MEDLINE subject field as labels, and concatenate the title and description fields to form the word vectors, with Porter-stemming and short word removal as pre-processing. The DMOZ2, wikip_med2 and wikip_large datasets are provided in pre-processed word vector forms, and no further pre-processing was made.

[14] http://www.cs.waikato.ac.nz/~asp12/

The TREC1-8 collections "trec_*" for ranked retrieval were pre-processed by stop-word, xml-tag, non-word, number, and short word removal, followed by Porter-stemming. For queries the description fields of queries were used to form the query word vectors, and relevance judgements were converted into binary label vectors of relevant document identifiers. The fire_en dataset was processed identically to the TREC datasets. For ohsu_trec the pre-processing was performed identically, but the queries were concatenated from title and description fields, and the provided graded relevance judgments were preserved as integer-weighted label vectors with relevance judgement grades 0, 1 and 2.

6.2.3 Segmentation

The segmentation scripts first partitioned the datasets into a development set for optimizing parameters and an evaluation set for computing the performance measures of the optimized models. Dataset-dependent segmentation choices were made to make the parameter optimization reliable, and keep processing complexity within practical bounds. Pre-existing dataset partitions were used where available. Otherwise random sampling was used to further segment the data. All datasets were mapped into the same framework of segmentation, with the development sets used for optimizing parameters and the evaluation sets used for conducting the evaluated experiment results. Both sets were further divided into a training sets (D^d, D^e) for learning the linear models, and test sets (D^{d*}, D^{e*}) for measuring performance.

Many of the classification datasets are provided with existing development and evaluation partitions. The original partitions for ohsu-trec and rcv1 were swapped, as this provided more data for learning models. The evaluation training set for all classification datasets was concatenated from the development training and test sets, while the evaluation test set composed of a held-out portion. The single-label and binary-label datasets ecue1 and ecue2 had insufficient data to form reliable development sets. For these datasets, 5-fold cross-validation over the development partition was used, with 200 documents preserved for each fold as the development test set, and the rest as the development train set. The classification datasets amazon12, rcv1, ohsu-trec, wikip_med2, and DMOZ2 had the lowest and highest document frequency words pruned, to reduce the total number of counts to 8 million per dataset, enabling efficient experimentation with less memory use.

The TREC datasets are commonly divided by combinations of data source, year, TIPSTER disk, and query number to form smaller segments for experi-

ments. Here the datasets were segmented by data source to form the 11 trec_* datasets. The trec_* and fire_en datasets were further segmented according to queries to form the development and evaluation sets, so that the first 20 queries from each year were concatenated to form the development test set, and the remaining 30 queries from each year were concatenated to form the evaluation test set. The ohsu_trec dataset was segmented according to the existing document partition, reserving ohsumed.87 for development and ohsumed.88-91 for evaluation. For all of the retrieval datasets, documents not given a relevance judgement for any of the queries in the test set were removed from the training set, greatly improving the efficiency for performing experiments.

The wikip_large dataset was segmented different from the general framework. The original training dataset provided for LSHTC4 was segmented by random sampling to reserve 1% of the data as an evaluation test set, and the remaining 99% was used as the evaluation training set. These evaluation sets were then further pruned by documents, features and labelsets so that each of these dimensions scaled up to a million. Documents were pruned in the order they occurred in a shuffled training dataset, with the number of preserved documents varied with the thresholds (10, 100, 1000, 10000, 100000, 1000000). Features were pruned to preserve the most frequent words, with the number or preserved features varied with the thresholds (10, 100, 1000, 10000, 100000, 1000000). Labelsets were similarly pruned to preserve the most common labelsets with the thresholds (1, 10, 1000, 1000000). Overall, these pruning choices resulted in 144 pruned versions of wikip_large for testing scalability of the models. The scalability experiments were then conducted on the evaluation sets using fixed parameters, and no development sets were constructed for optimization.

6.2.4 Dataset Statistics

The common framework for classification and ranking tasks enables direct comparison of the dataset properties. Table 6.1 shows the basic dataset statistics of numbers of documents, features and labels for the development and evaluation sets. Table 6.2 shows the mean numbers of features and labels per document. For the datasets that use 5-fold cross-validation for development, the first fold is used to compute the statistics for the tables. For the retrieval datasets, label variables for labeled non-relevant documents are not included for showing the number of document labels per query, as these are assumed to be as relevant as non-labeled documents.

Table 6.1: Basic statistics of the pre-processed and segmented experiment datasets. D^d is the development training set, D^{d*} the development test set, D^e the evaluation training set and D^{e*} the evaluation test set. Statistics for the number of label variables L and the number of word features N are denoted correspondingly

dataset	$\|D^d\|$	$\|D^{d*}\|$	$\|D^e\|$	$\|D^{e*}\|$	L^d	L^e	N^d	N^e
20ng	11093	200	11293	7528	20	20	54112	54580
cade	27122	200	27322	13661	12	12	156751	157483
r52	6332	200	6532	2568	52	52	15882	16145
r8	5285	200	5485	2189	8	8	14334	14575
webkb	2585	200	2785	1396	4	4	7287	7287
ecue1	9778	200	9978	1000	2	2	100000	100000
ecue2	10665	200	10865	1000	2	2	159579	161155
trec06	34039	1000	34039	2783	2	2	797772	797772
tripa12	55299	4999	55299	10077	2	2	76364	76364
aclimdb	45000	2000	45000	3000	2	2	89527	89527
amazon12	257877	9998	257877	100556	2	2	86914	86914
rcv1	342117	1000	342117	8644	350	350	160281	160281
eurlex	16381	1000	16381	1933	3828	3828	172928	172928
ohsu-trec	196555	1000	196555	35890	14373	14373	290117	290117
DMOZ2	390809	2000	390809	1947	27874	27874	111939	111939
wikip_med2	452318	2000	452318	2568	36463	36463	47021	47021
wikip_large	NA	NA	2341782	23654	NA	324634	NA	1608946
fire_en	21919	90	16075	60	21919	16075	103551	91089
ohsu_trec	36890	63	196555	63	36890	196555	77994	220256
trec_ap	47172	150	33474	100	47172	33474	201591	162284
trec_cr	5063	60	4006	40	5063	4006	198170	188513
trec_doe	10053	89	7717	59	10053	7717	32352	28569
trec_fbis	23207	90	17315	60	23207	17315	202033	175660
trec_fr	25185	240	20581	160	25185	20581	252577	242648
trec_ft	41452	120	30549	80	41452	30549	228547	187797
trec_la	25944	90	17834	60	25944	17834	162531	129299
trec_pt	1635	30	1792	20	1635	1792	111883	106147
trec_sjmn	9160	30	6469	20	9160	6469	74447	59992
trec_wsj	57117	150	45078	100	57117	45078	247771	215497
trec_zf	19901	150	13763	99	19901	13763	192489	158042

Table 6.2: Mean statistics of the pre-processed and segmented experiment datasets. $|\boldsymbol{w}^d|_0$ is the mean number of unique words and $|\boldsymbol{c}^d|_0$ the mean number of labels per document in the training set. The corresponding statistics are given for the development test set ($\boldsymbol{w}^{d*}$, $\boldsymbol{c}^{d*}$), evaluation training set ($\boldsymbol{w}^{e}$, $\boldsymbol{c}^{e}$) and evaluation test set ($\boldsymbol{w}^{e*}$, $\boldsymbol{c}^{e*}$)

dataset	$\lvert\boldsymbol{w}^d\rvert_0$	$\lvert\boldsymbol{w}^{d*}\rvert_0$	$\lvert\boldsymbol{w}^e\rvert_0$	$\lvert\boldsymbol{w}^{e*}\rvert_0$	$\lvert\boldsymbol{c}^d\rvert_0$	$\lvert\boldsymbol{c}^{d*}\rvert_0$	$\lvert\boldsymbol{c}^e\rvert_0$	$\lvert\boldsymbol{c}^{e*}\rvert_0$
20ng	84.20	90.62	84.32	83.14	1.00	1.00	1.00	1.00
cade	62.24	74.68	62.33	59.83	1.00	1.00	1.00	1.00
r52	43.08	44.08	43.11	39.71	1.00	1.00	1.00	1.00
r8	41.34	37.49	41.20	37.28	1.00	1.00	1.00	1.00
webkb	76.86	79.79	77.07	79.03	1.00	1.00	1.00	1.00
ecue1	186.54	165.13	186.11	211.28	1.00	1.00	1.00	1.00
ecue2	144.02	145.66	144.05	132.91	1.00	1.00	1.00	1.00
trec06	107.21	85.01	107.21	93.97	1.00	1.00	1.00	1.00
tripa12	105.25	106.85	105.25	104.61	1.00	1.00	1.00	1.00
aclimdb	136.51	129.71	136.51	131.53	1.00	1.00	1.00	1.00
amazon12	30.69	30.20	30.69	30.63	1.00	1.00	1.00	1.00
rcv1	22.44	21.02	22.44	21.84	1.60	1.58	1.60	1.57
eurlex	271.42	252.74	271.42	271.84	5.32	5.29	5.32	5.32
ohsu-trec	40.12	37.00	40.12	37.35	12.39	11.66	12.39	11.93
DMOZ2	20.44	20.03	20.44	23.72	1.03	1.02	1.03	1.03
wikip_med2	17.27	18.38	17.27	15.52	1.84	2.01	1.84	1.67
wikip_large	NA	NA	42.54	42.27	NA	NA	3.26	3.27
fire_en	145.84	6.92	148.56	6.57	1.00	52.48	1.00	41.02
ohsu_trec	50.87	6.41	53.25	6.41	1.00	10.63	1.00	50.87
trec_ap	164.77	8.71	165.26	8.62	1.00	82.78	1.00	81.66
trec_cr	473.82	7.33	531.94	8.50	1.00	14.75	1.00	7.00
trec_doe	43.65	9.62	45.36	9.97	1.00	9.63	1.00	22.93
trec_fbis	205.15	6.78	216.20	7.97	1.00	30.07	1.00	28.40
trec_fr	233.84	7.98	253.04	8.38	1.00	6.50	1.00	7.38
trec_ft	155.63	6.91	158.34	7.71	1.00	29.03	1.00	37.53
trec_la	202.05	6.78	205.21	7.97	1.00	23.04	1.00	24.35
trec_pt	563.39	7.30	517.50	6.50	1.00	0.83	1.00	0.75
trec_sjmn	178.74	7.30	179.12	6.50	1.00	28.53	1.00	22.05
trec_wsj	188.03	8.71	194.51	8.62	1.00	66.42	1.00	69.81
trec_zf	216.07	8.71	219.00	8.66	1.00	17.65	1.00	18.35

Table 6.3: Modification affixes and reference pages in the thesis

affix	modification	reference pages
u	uniform background model	77
c	collection background model	77
uc	uniform-smoothed collection background model	77
dp	Dirichlet prior smoothing	78
jm	Jelinek-Mercer smoothing	78
ad	absolute discounting	78
pd	power-law discounting	78
kdp	kernel Dirichlet prior smoothing	78, 111
kjm	kernel Jelinek-Mercer smoothing	78, 111
kpd	kernel power-law discounting	78, 111
po	Poisson document length modeling	97
ps	prior scaling	97
qidf	query IDF weighting	87
qidfX	query IDF weighting with IDF lifting	87
ti	TF-IDF weighting	87
tXi	TF-IDF weighting with length scaling	87
tiX	TF-IDF weighting with IDF lifting	87
tXiX	TF-IDF weighting with length scaling and IDF lifting	87
l1r	L1 regularization for LR/SVM	121
l2r	L2 regularization for LR/SVM	121

When compared under the same framework, it can be seen from Table 6.2 that the training and test set documents in retrieval have very different properties. The test set documents are queries, which are often 20 times shorter than the training documents, and have a large number of labels; whereas the retrieved documents have each one document identifier label. In stark contrast, text classification training and test set documents are generally drawn from the same type of data. Both types of tasks can have significant differences between development and evaluation conditions, although generally these are defined to be similar enough for meta-parameter optimization to be possible.

6.3 Experiments and Results

6.3.1 Evaluated Linear Model Modifications

The experiments compare a number of models across datasets and measures. Modifications to models are denoted by acronym affixes separated by underscores, for example “jm” for Jelinek-Mercer smoothed models, and “u_dp” for uniform-background Dirichlet prior smoothed models. Throughout the experiments only a subset of possible modifications is attempted, because there are 39 combinations for the basic smoothing methods, and far more when feature weighting and structural models are included. Table 6.3 summarizes the modifications used in the experiments, with references to descriptions. None of the MNB or TDM modifications increase the time or space complexities of

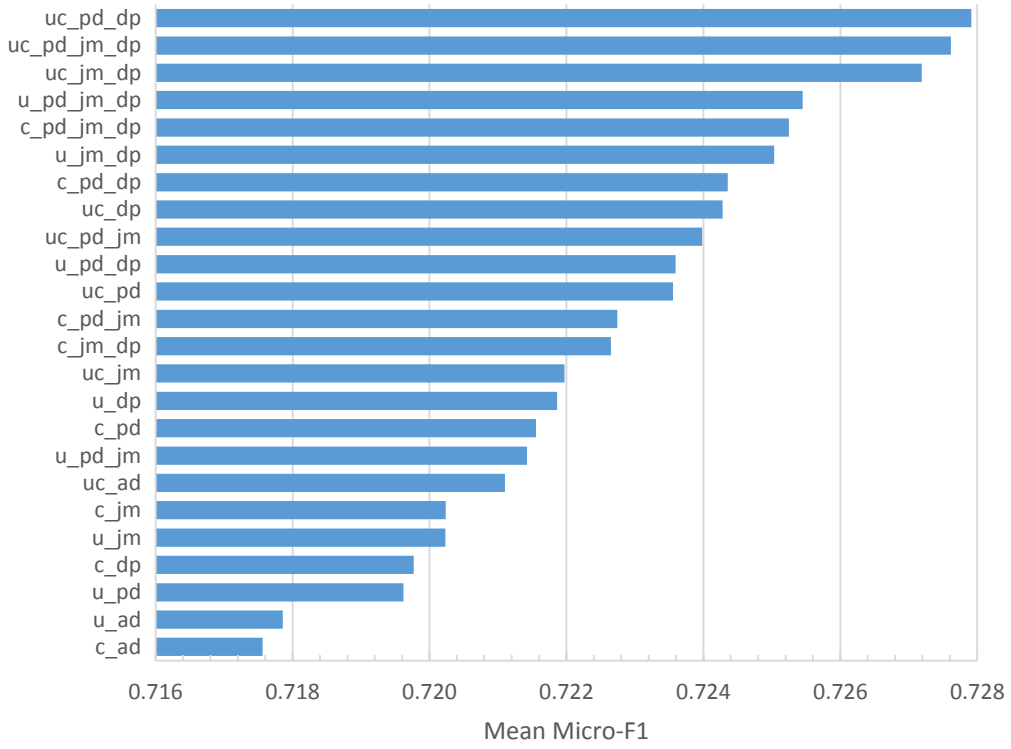


Figure 6.3: Mean Micro-F1 across the text classification datasets with the different multinomial smoothing methods

estimation or inference, or introduce considerable additional constants to processing requirements. IDF lifting can decrease the complexities, sparsifying documents further by weighting frequently occurring words to 0.

6.3.2 Smoothing Methods

The first set of experiments evaluates the usefulness of the common framework for multinomial models of text presented in Chapter 4. Four methods are used for discounting and smoothing: Dirichlet prior (dp), Jelinek-Mercer (jm), absolute discounting (ad), and power-law discounting (pd). These are combined with three choices for background distribution: uniform (u), collection (c), and uniform-smoothed collection (uc). Absolute discounting proved early in the experiments to be inferior to power-law discounting, and further combinations with the other methods are not shown. This was expected from the literature [Huang and Renals, 2010]. Combinations for the other models were explored based on the initial performance of the collection-smoothed models.

Figure 6.3 shows the mean Micro-F1 across the text classification datasets for the different smoothing methods. Figures 6.4 and 6.5 shows the mean

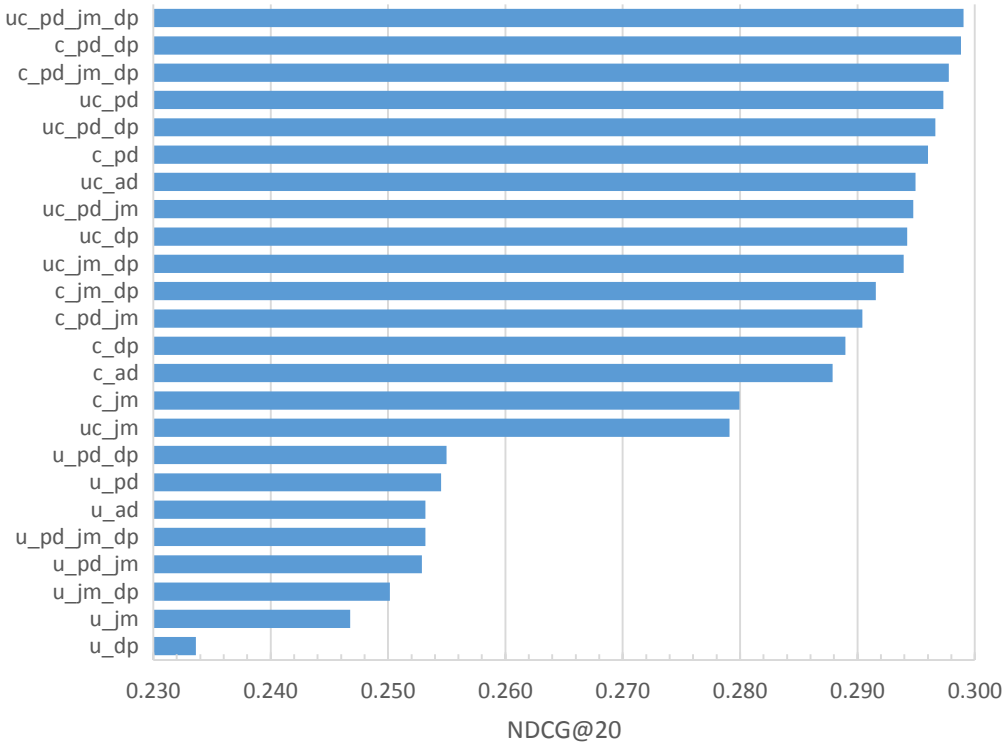


Figure 6.4: Mean NDCG@20 across the text retrieval datasets with the different multinomial smoothing methods

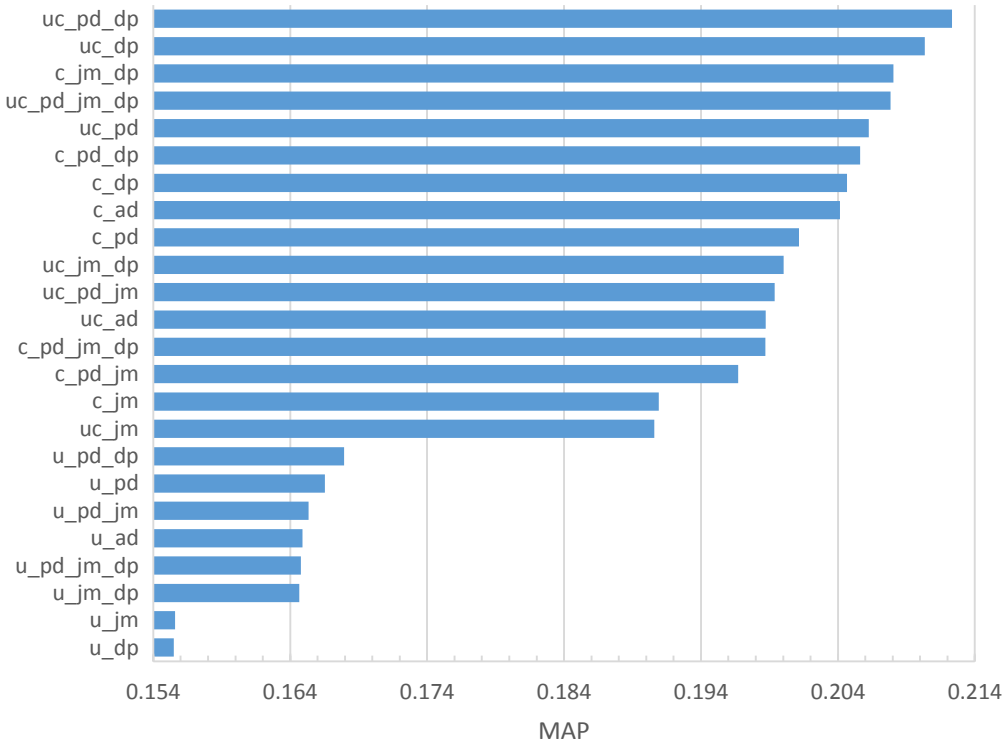


Figure 6.5: Mean MAP across the text retrieval datasets with the different multinomial smoothing methods

NDCG@20 and MAP across the text retrieval datasets for the different smoothing methods, respectively. The first visible effect is the overall improvement from the combinations, compared to the individual smoothing methods in standard use, such as c_dp and c_jm. Overall, the differences are small but consistent.

The main hypothesis to test is whether the generalized smoothing method (uc_pd_jm_dp) improves over the baselines Dirichlet prior (c_dp) and Pitman-Yor process smoothing (c_pd_dp). Comparing the combined smoothing method uc_pd_jm_dp to the baseline c_dp, a relative reduction of 2.80%† in Micro-F1 and relative improvements of 3.37% in NDCG@20 and 1.53% in MAP are seen. Comparing to the stronger baseline of c_pd_dp, the corresponding reductions are 1.03%‡, 0.03%, and 0.28%.

6.3.3 Feature Weighting and the Extended MNB

The second set of experiments compares feature weighting and the Extended MNB model that adds document length modeling and scaling of the label prior. Feature weighting in text retrieval has been proposed in the form of query weighting [Smucker and Allan, 2006, Momtazi et al., 2010], whereas in text classification both training and test set documents are weighted similarly [Rennie et al., 2003]. Document length modeling and prior scaling are applicable for the text classification experiments, where test documents can be assumed to have the same length distributions as training documents, and the prior probabilities of labels have varying degrees of usefulness for prediction.

The experiments first compared the usefulness of Poisson length modeling (po) and prior scaling (ps). Parameters for both were allowed to vary within the permissible range from 0.0 to 2.0. Prior scaling was noticed to improve classification considerably on most datasets. Poisson length modeling gave no significant improvement on average, and no additional gain was observed in combination with prior scaling. The following experiments on text classification used prior scaling, but not length modeling.

Feature weighting was attempted with both query idf weighting (qidf) and TF-IDF training and test document weighting (ti). Parameterized versions used IDF lifting (tiX, qidfX) in the range -1.0 to 50.0, length scaling (tXi) in the range -1.0 to 2.0, or both (tXiX). A limited selection of the best-performing smoothing models were chosen for these experiments, with uniform or uniform-smoothed collection distributions for background models, and prior scaling for

the text classification datasets.

Figures 6.6, 6.7 and 6.8 show the results for the second set of experiments, averaged across the datasets. For reference, the baselines c_dp and c_pd_dp are included in the figures. Compared to the smoothing method variants, the improvements from feature weighting and prior scaling are substantial. Averaged over the datasets, uc_pd_dp_tXiX_ps produces a relative error reduction of 7.50%† in Micro-F1 over c_pd_dp in text classification, and u_pd_qidf produces relative improvements over c_pd_dp of 8.26%‡ in NDCG@20 and 11.03%‡ in MAP.

6.3.4 Tied Document Mixture

The third set of experiments explored the Tied Document Mixture (TDM) model proposed in Chapter 5 for text classification. Smoothing the TDM kernel densities was done with Jelinek-Mercer (kjm), power-law discounting (kpd), and Dirichlet prior (kdp). Due to longer processing times on the largest datasets, as well as a much larger number of possible combinations for smoothing, a small number of combinations successful on single-label datasets were chosen for a full set of experiments. For simplifying the comparisons, the feature weighting and prior scaling combinations were also excluded from the TDM experiments, and left for future experimentation.

Figure 6.9 shows the Micro-F1 results averaged across the text classification datasets. For comparison, MNB baselines from the previous sets of experiments have been included, and results for the two models are separated by the "tdm" and "mnb" affixes. The best performing model tdm_uc_jm_dp_kpd_kdp produces a relative Micro-F1 improvement of 2.55% over mnb_uc_pd_dp_tXiX_ps and 8.67%† over mnb_uc_pd_dp.

6.3.5 Comparison with Strong Linear Model Baselines

The fourth set of experiments compared strong linear model baselines to the results from the first three sets of experiments. For ad-hoc text retrieval a strong baseline is the BM25 model (bm25), while results from the earlier VSMs can be included for comparison. For text classification tasks the strong baselines are LR (lr) and l2-SVM (l2svm) models, combined with the parameterized TF-IDF feature weighting used with the Extended MNB models.

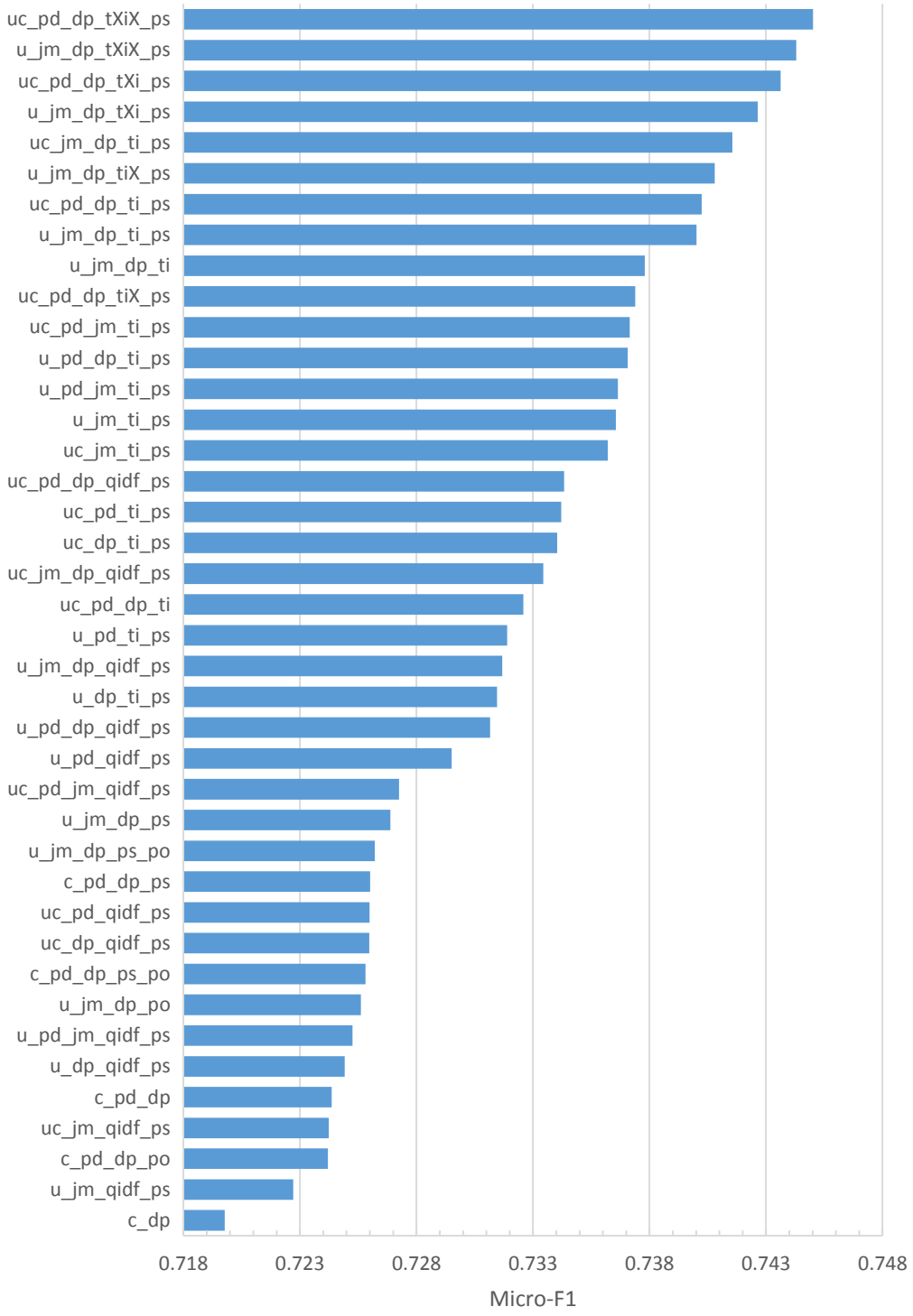


Figure 6.6: Mean Micro-F1 across the text classification datasets with the Extended MNB models. Baseline models c_dp and c_pd_dp included for comparison

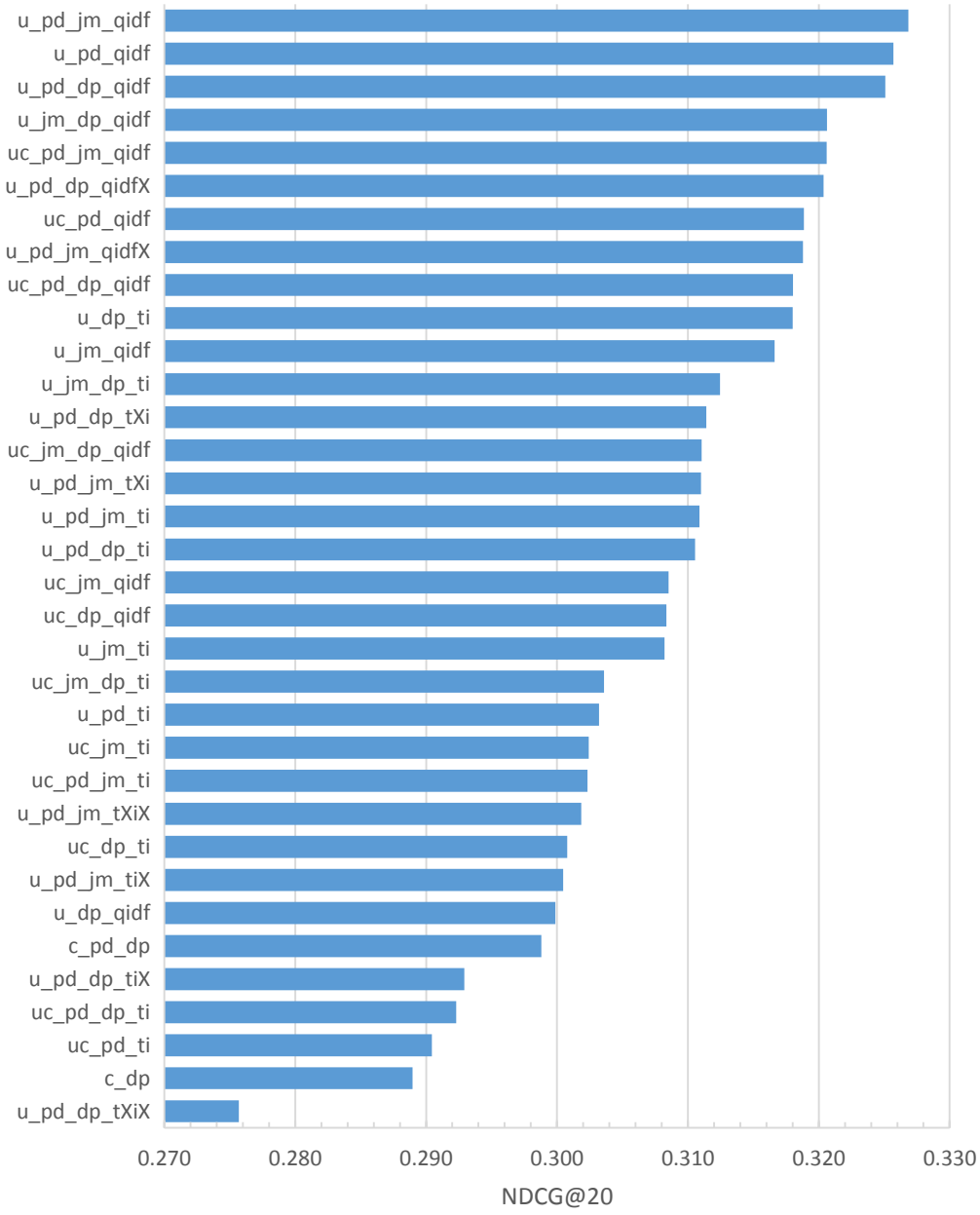


Figure 6.7: Mean NDCG@20 across the text retrieval datasets with the Extended MNB models. Baseline models c_dp and c_pd_dp included for comparison

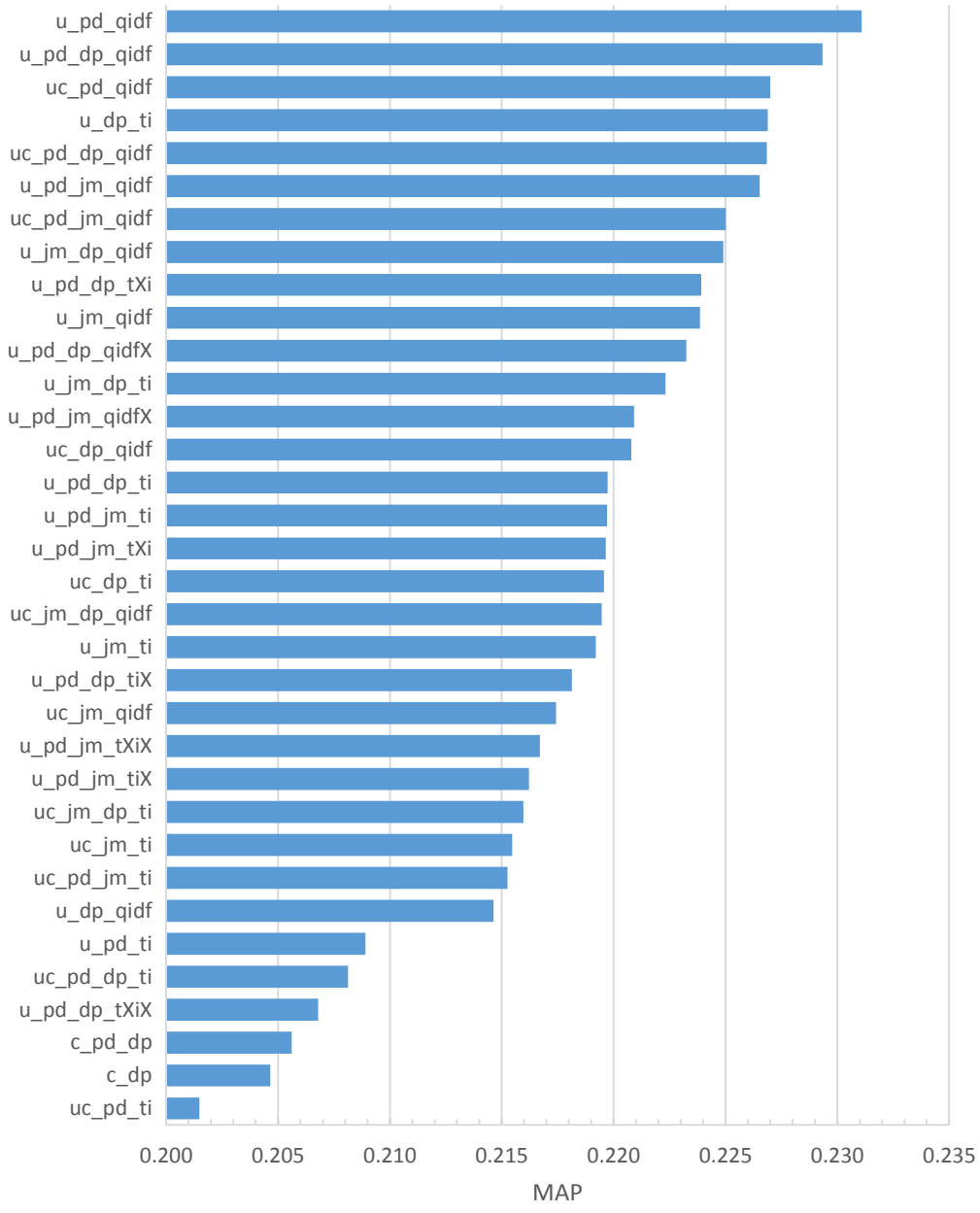


Figure 6.8: Mean MAP across the text retrieval datasets with the Extended MNB models. Baseline models c_dp and c_pd_dp included for comparison

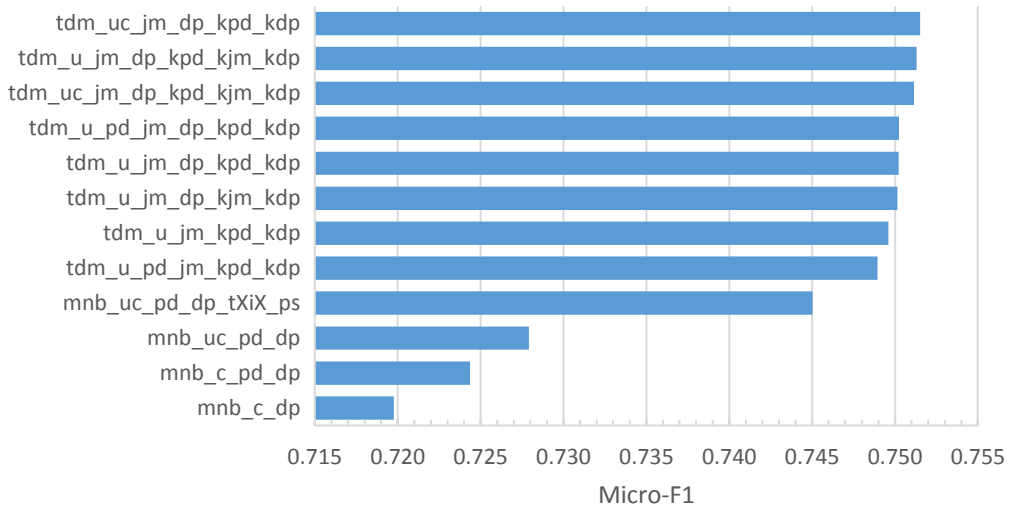


Figure 6.9: Mean Micro-F1 across the text classification datasets with different TDM models. MNB baseline models included for comparison

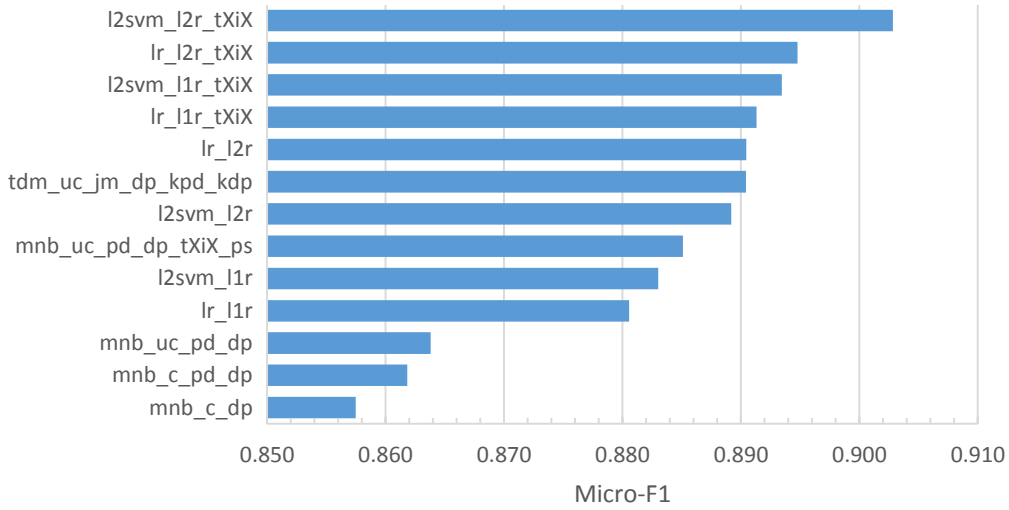


Figure 6.10: Mean Micro-F1 across the binary-label and multi-class text classification datasets for the baseline LR and SVM models, compared to MNB and TDM models

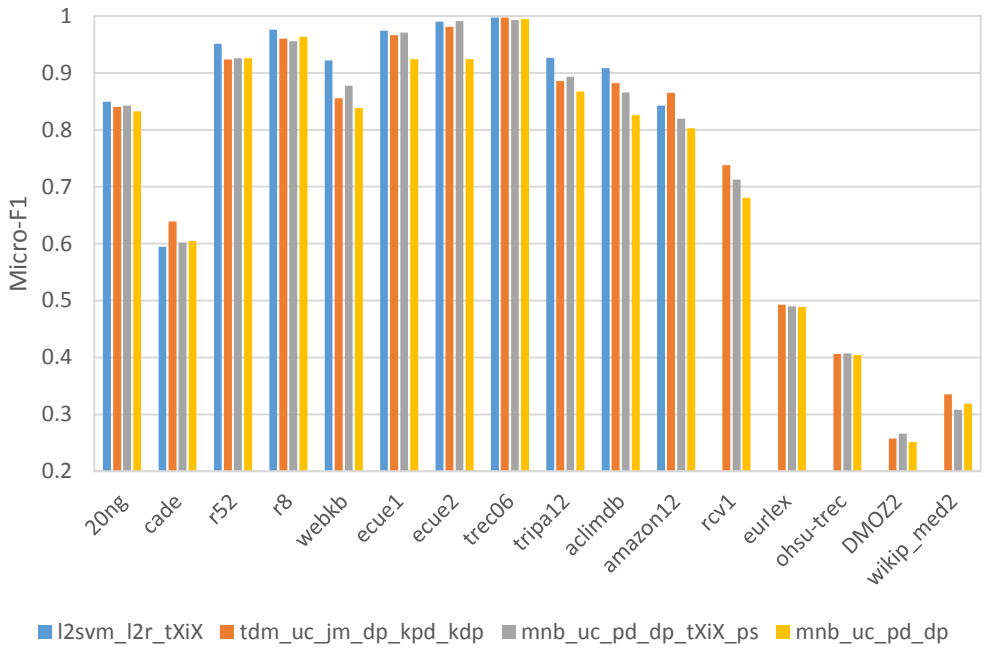


Figure 6.11: Micro-F1 on each text classification dataset. TDM outperforms the L2-regularized SVM with parameterized TF-IDF on cade and amazon12

Figure 6.10 summarizes the Micro-F1 results for the LR and SVM baselines compared to MNB and TDM, averaged across the binary-label and multi-class datasets. Overall, the feature weighted SVM and LR models seem to outperform the generative models, l2svm_l2r_tXiX in particular showing exceptionally high performance. The difference of l2svm_l2r_tXiX is 11.32% to tdm_uc_jm_dp_kpd_kdp, 15.43%† to mnb_uc_pd_dp_tXiX_ps, and 28.66%† to mnb_uc_pd_dp. Despite the difference in mean scores, none of the LR and SVM models significantly improve over TDM, when tested across datasets.

The differences within each dataset can be further examined. Figure 6.11 shows the differences within each dataset. On both cade and amazon12, tdm_uc_jm_dp_kpd_kdp outperforms l2svm_l2r_tXiX, while mnb_uc_pd_dp_tXiX_ps never outperforms l2svm_l2r_tXiX. The most likely reason for this is that both feature weighted MNB and SVM models are linear models, and the parameter estimation for learned linear models provides more accurate classifiers in linearly separable classification problems. Datasets such as cade and amazon12 are possibly more non-linear, and TDM outperforms in these cases. With large-scale multi-label datasets SVM and LR are not directly usable in reasonable processing time, and on rcv1 and wikip_med2 TDM improves on MNB by a small margin.

Figures 6.12 and 6.13 show comparisons of the MNB models to VSM and

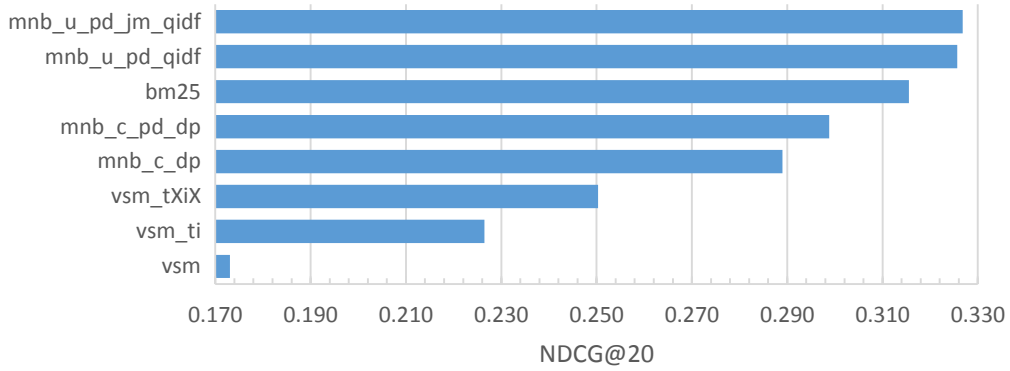


Figure 6.12: Mean NDCG@20 across the text retrieval datasets with the baseline VSM and BM25 models, compared to MNB models

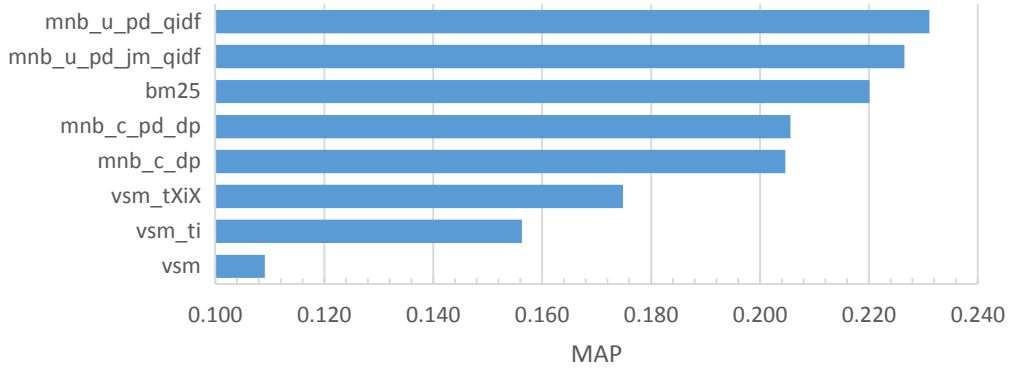


Figure 6.13: Mean MAP across the text retrieval datasets with the baseline VSM and BM25 models, compared to MNB models

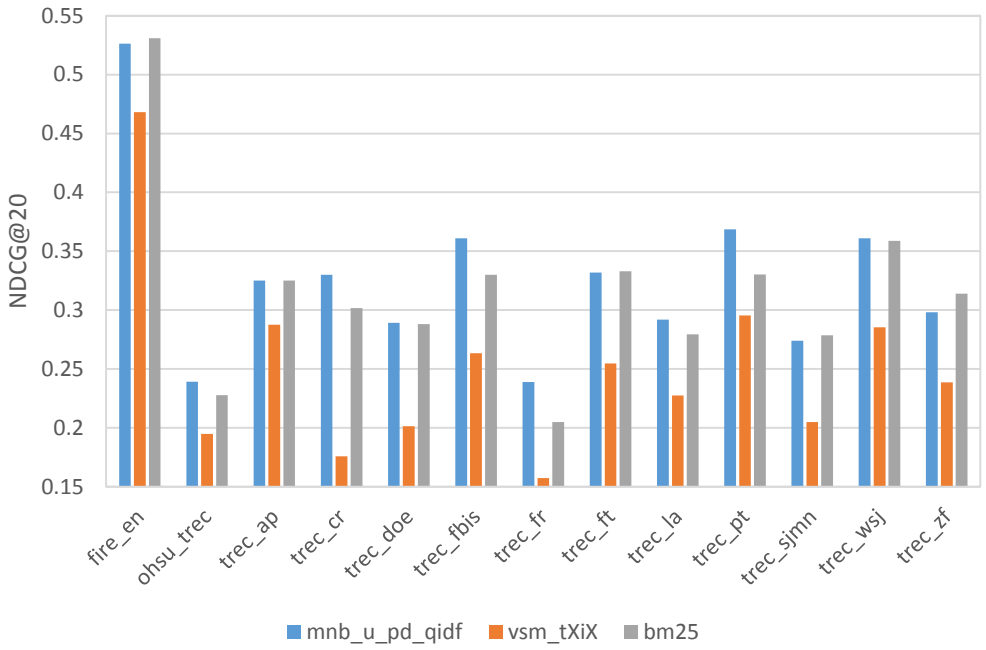


Figure 6.14: NDCG@20 on each text retrieval dataset, comparing MNB with VSM and BM25

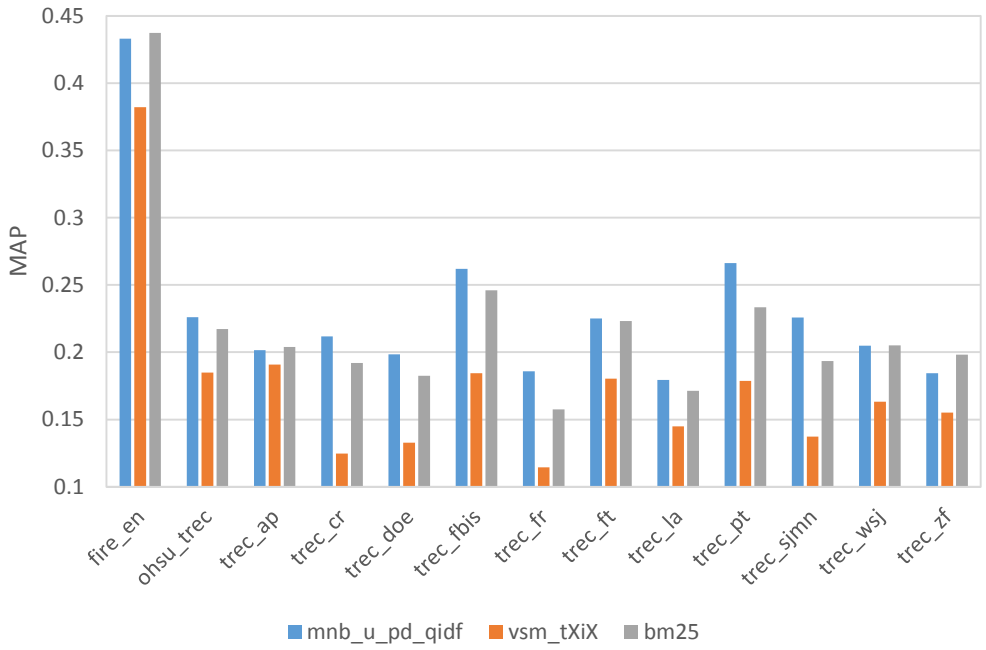


Figure 6.15: MAP on each text retrieval dataset, comparing MNB with VSM and BM25

BM25 models in mean NDCG@20 and MAP across the datasets. The results are nearly identical under both measures. VSM models fall behind MNB models, while BM25 outperforms basic MNB models, but not the MNB models using query weighting. The improvement of mnb_u_pd_qidf over vsm_tXiX is 23.42%† in NDCG@20 and 24.33%† in MAP. The improvement over bm25 is 3.12%‡ in NDCG@20 and 4.76%‡ in MAP.

The NDCG@20 and MAP differences within each retrieval dataset are illustrated in Figures 6.14 and 6.15. While mnb_u_pd_qidf and bm25 have similar performance, on several datasets mnb_u_pd_qidf outperforms bm25 by a margin on both measures, resulting in the significant differences across the datasets.

6.3.6 Scalability and Efficiency

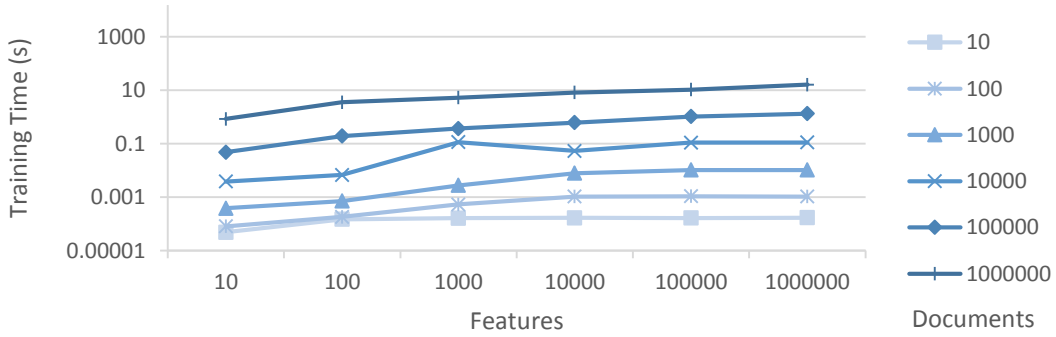
The fifth set of experiments explored the scalability of linear models for classification in estimation and inference. The large Wikipedia dataset from the LSHTC4 competition was pruned to scale the number documents, features, and labelsets, each up to a million. The Label Powerset method [Boutell et al., 2004] was used to map the multi-label learning problems into a multi-class problem directly learnable by the LR and SVM models. For improving scalability, all evaluated models used TF-IDF feature weighting with the un-

smoothed Croft-Harper IDF, further pruning the words occurring in more than half of the training set documents. Each model configuration was allowed to run for four hours on a 3.40GHz i7-2600 processor with 16GB of RAM memory, and runs taking longer were terminated. The learned models lr_l2r_-tXiX, l2svm_l2r_tXiX, l2svm_l1r_tXiX, lr_l1r_tXiX were evaluated, as well as mnb_u_jm_tXiX and tdm_u_jm_kjm_tXiX. Hash table implementations of the generative models instead of an inverted index were tested to evaluate the significance of the sparse posterior inference. These perform MNB inference by updating the conditional probabilities for each label, for each word in the test document. This gives the commonly considered "optimal" time complexity for MNB [Manning et al., 2008].

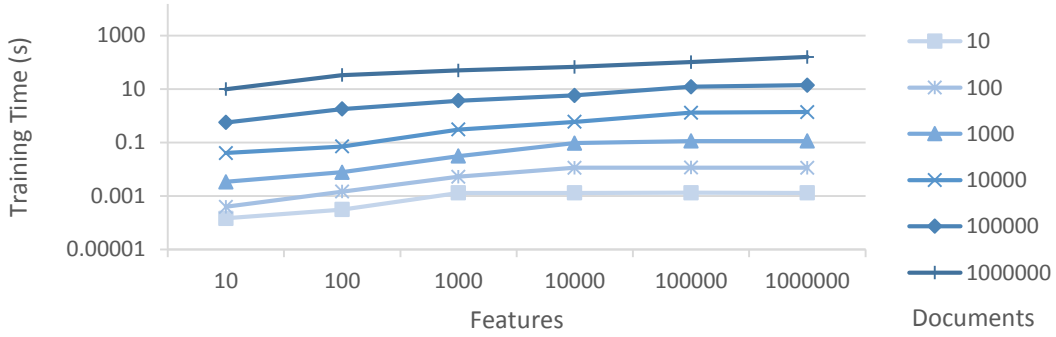
Figure 6.16 shows the estimation times for l2svm_l2r_tXiX and Figure 6.17 for mnb_u_jm_tXiX. Figure 6.16 for the SVM model shows exponentially scaling estimation times, with the times increasing rapidly with more documents and labelsets. Figure 6.17 for the MNB model shows linear estimation times, unaffected by the number of labels, and only marginally affected by the number of features. The SVM model does not complete the task in the allowed time when the allowed documents and labelsets both number over 10000, while the MNB model completes in all but few of the largest configurations. The estimation times of the other learner linear models behave similarly to l2svm_-l2r_tXiX, while estimation for tdm_u_jm_kjm_tXiX behaves similarly to mnb_-u_jm_tXiX. The constant difference in small numbers of documents in favor of SVM is due to a pre-processing difference: the LR and SVM models were implemented in C with LibLinear, with Python feature reading times subtracted from the training times, while the generative models were implemented in Java with SGMWeka and the training times include a constant from reading the feature files.

The inference times for the models depend on both sparsity of the parameters and their representation. Application of sparse inference reduces the inference complexities for both linear and non-linear models according to sparsity of the parameters. Figure 6.18 compares the inference times for mnb_-u_jm_tXiX and tdm_u_jm_kjm_tXiX using an inverted index and a hash table, with labelsets pruned to 1000000.

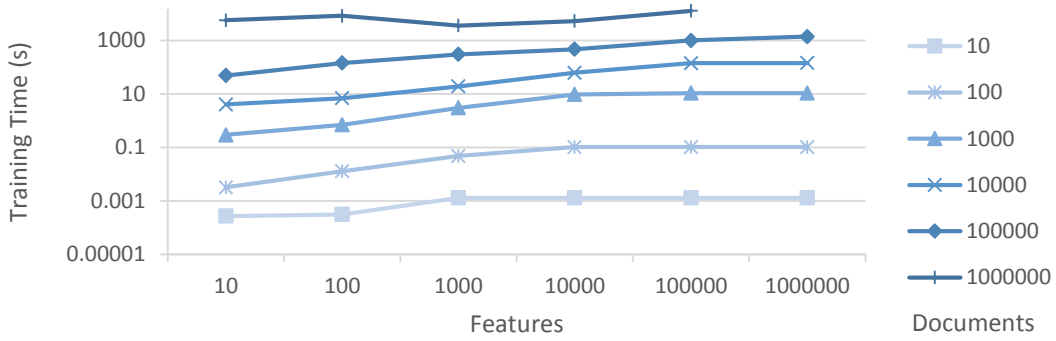
Figure 6.18 exhibits several overlapping effects. With a hash table, inference for MNB and TDM has similar complexity, producing nearly identical scaling. With an inverted index, the inference becomes more scalable for both models. While most of the configurations with over 100000 training documents



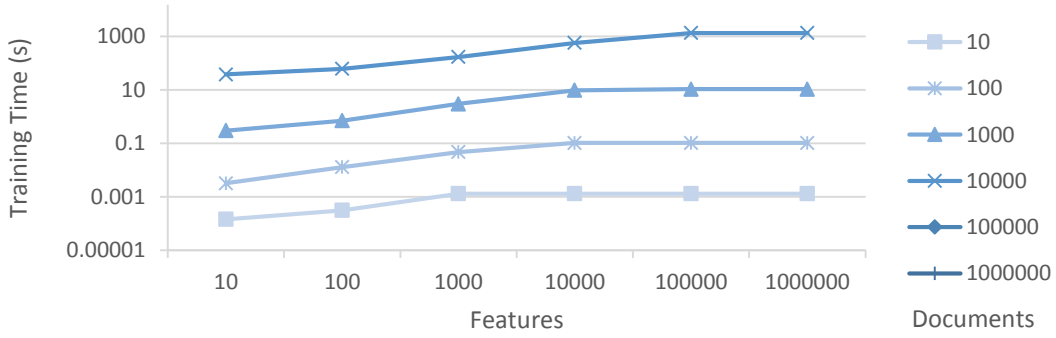
(a) L2-regularized L2-Support Vector Machines, labelsets pruned to 1



(b) L2-regularized L2-Support Vector Machines, labelsets pruned to 10

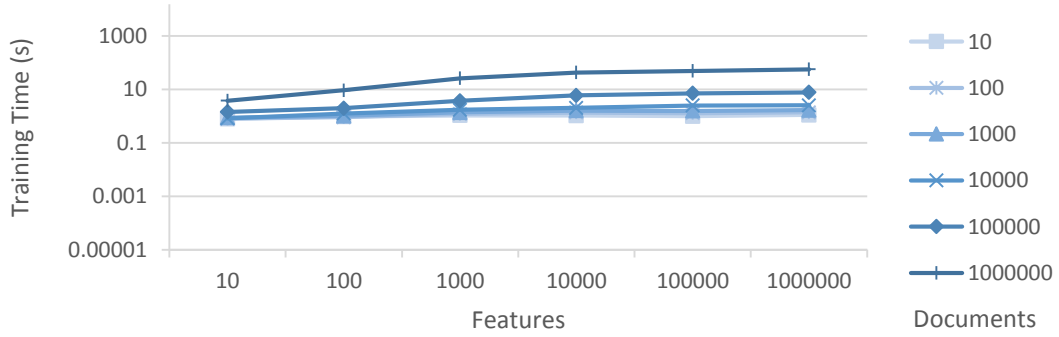


(c) L2-regularized L2-Support Vector Machines, labelsets pruned to 1000

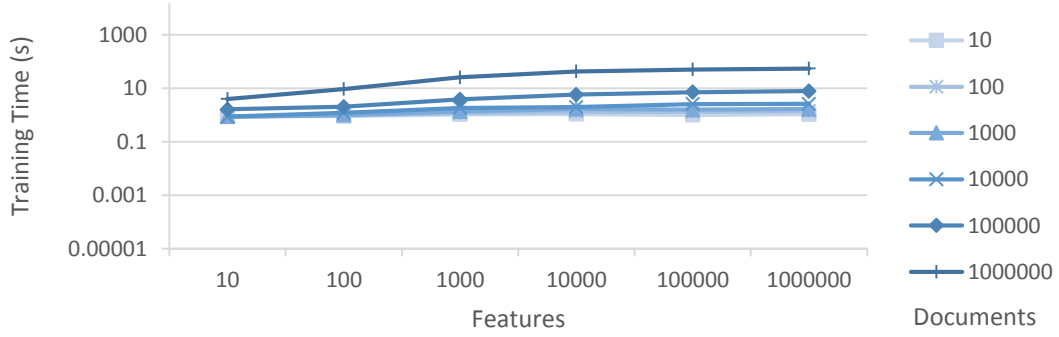


(d) L2-regularized L2-Support Vector Machines, labelsets pruned to 1000000

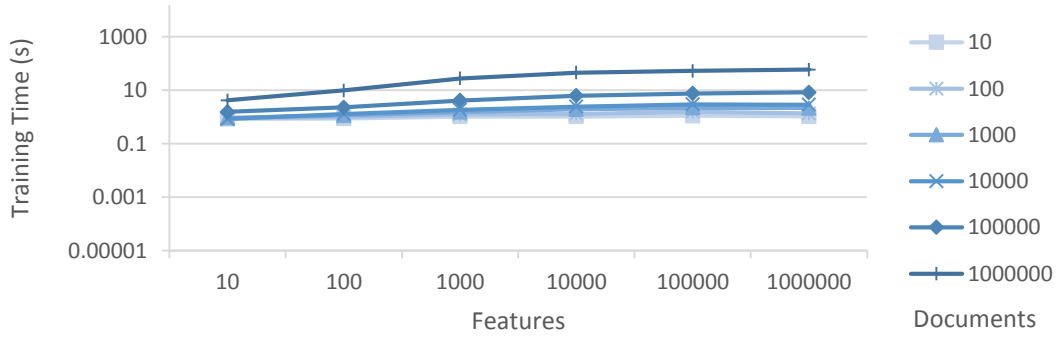
Figure 6.16: Estimation times for L2-regularized L2-Support Vector Machines on wikip_large, with different pruning of documents, features and labelsets



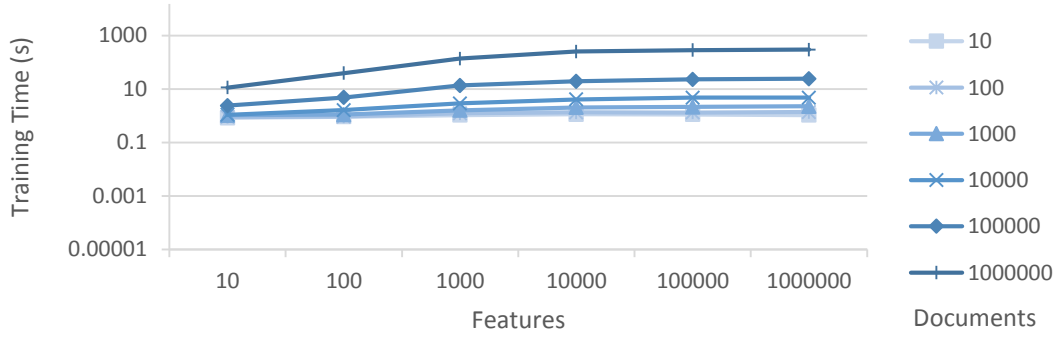
(a) Multinomial Naive Bayes with an inverted index, labelsets pruned to 1



(b) Multinomial Naive Bayes with an inverted index, labelsets pruned to 10

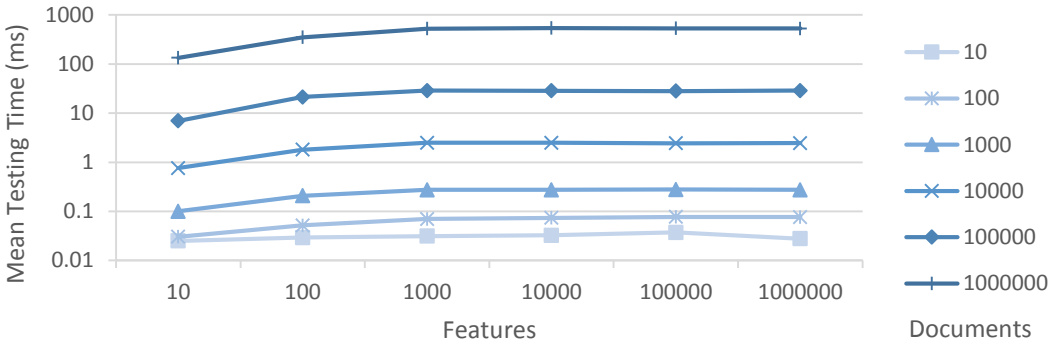


(c) Multinomial Naive Bayes with an inverted index, labelsets pruned to 1000

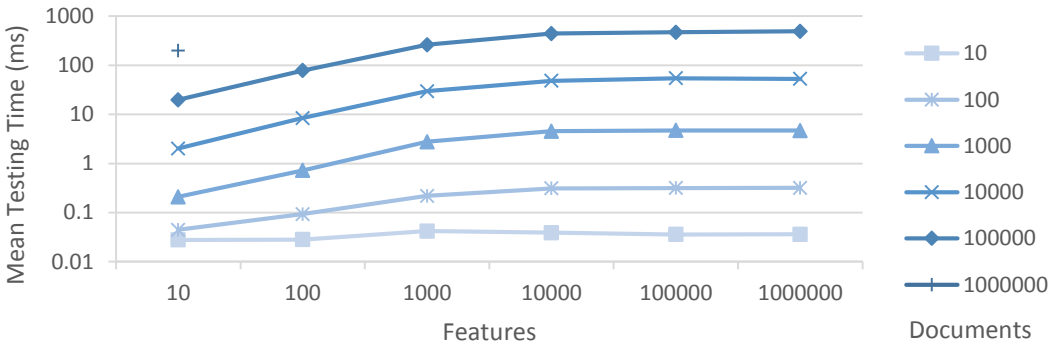


(d) Multinomial Naive Bayes with an inverted index, labelsets pruned to 1000000

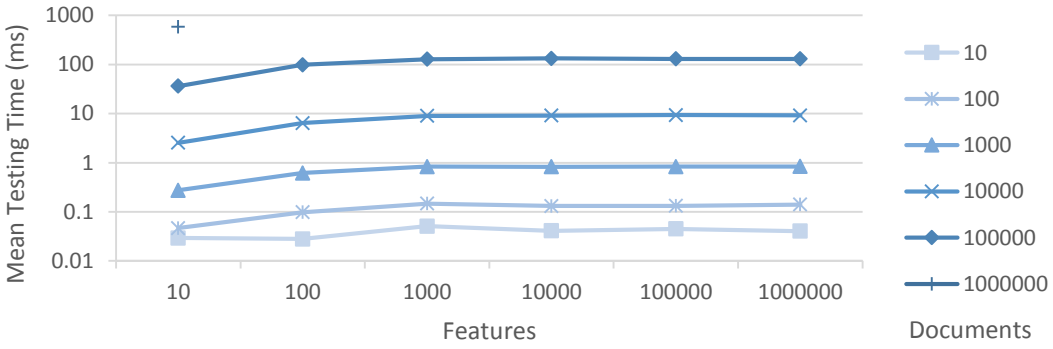
Figure 6.17: Estimation times for Multinomial Naive Bayes with an inverted index on wikip_large, with different pruning of documents, features and labelsets



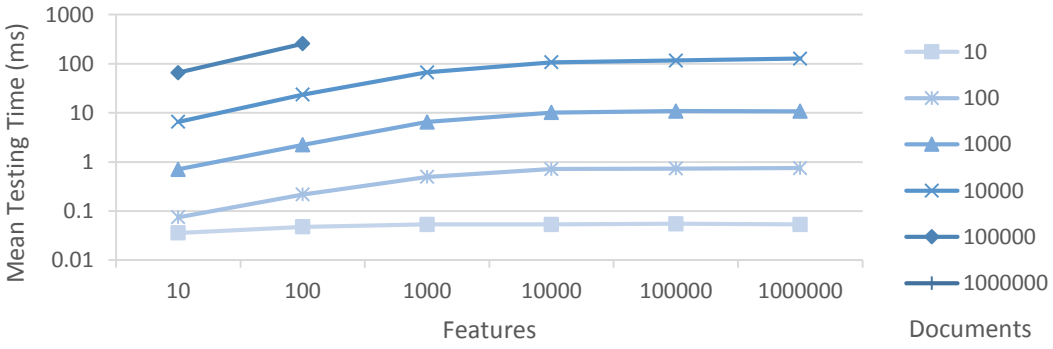
(a) Multinomial Naive Bayes with an inverted index



(b) Multinomial Naive Bayes with a hash table



(c) Tied Document Mixture with an inverted index



(d) Tied Document Mixture with a hash table

Figure 6.18: Mean inference times per document on wikip_large with 1000000 labelsets. Multinomial Naive Bayes and Tied Document Mixture compared, with inverted index and hash table implementations

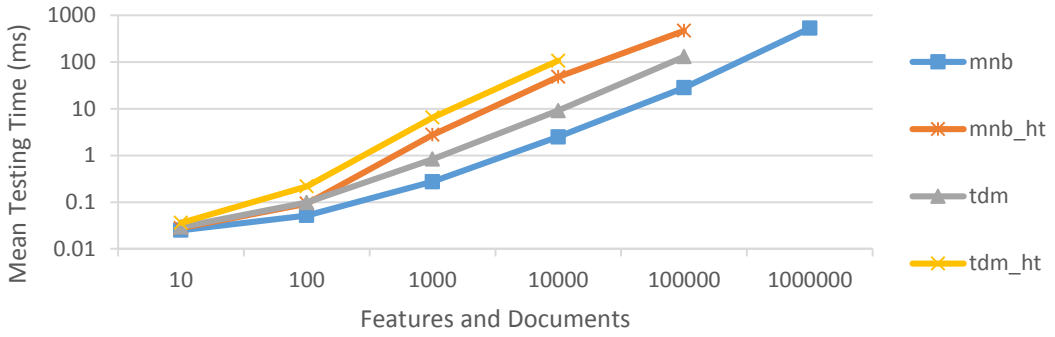


Figure 6.19: Mean inference times per document on wikip_large with 1000000 labelsets. Inverted index and hash table (_ht) implementations compared, with both the number of features and documents increased

do not complete in time with a hash table, the inverted index implementations scale more easily to 1000000 training documents. For both MNB and TDM, the sparse inference times scale linearly with the number of training documents, since this also increases the number of seen labelsets closer to 1000000. For both models, with high numbers of documents the number of features induces an exponential growth with the hash table, whereas with inverted index the growth becomes less exponential. Figure 6.19 shows this effect in more detail. The highest-dimensional TDM model (10000 features, 10000 documents) to complete the task within four hours with the hash table implementation took 106.8 ms per classification. The corresponding inverted index implementation took 9.1 ms per classification, an order of magnitude reduction in mean classification times. The gap between a common hash table implementation and the sparse inference with an inverted index will only increase from this in higher dimensional tasks.

Chapter 7

Conclusion

This chapter concludes the thesis with a discussion. First a summary of the thesis results and implications of the findings are discussed. The thesis statement is revisited, arguing that models extending Multinomial Naive Bayes offer both a versatile solution in terms of both effectiveness and scalability. Limitations of the thesis and future work are discussed, considering current developments related to text mining.

7.1 Summary of Results

This thesis proposed generative models of text using sparse computation as a general solution for text mining applications. The problems of fragmentation of research and scalability of models were identified as central problems for text mining. A solution based on modified generative multinomial models of text combined with a novel type of exact sparse inference was proposed as a solution for a variety of text mining tasks.

Building on overviews of both text mining and generative multinomial models for text, the thesis showed the connection of the Multinomial Naive Bayes (MNB) models to linear models and directed generative graphical models. Modifications and extensions to MNB such as smoothing and feature weighting were formalized as constrained graphical models estimated with the maximum likelihood principle.

Inference using inverted indices was shown to reduce the complexity of inference with linear models according to the sparsity of the model representation. This sparse inference was shown to be equally applicable to structured extensions of linear models. A hierarchical extension of MNB called Tied Document Mixture (TDM) was proposed as a basic extension of MNB with

document-level nodes. Empirical evaluation of the TDM and modified MNB models showed that the models offer highly competitive performance across text classification and ranking tasks, and sparse inference reduced the inference times by an order of magnitude in the largest considered experiments that completed within the allowed time.

7.2 Implications of Findings

Current research in machine learning considers generative Bayes models for classification generally inferior to discriminative models. The formalization of model modifications and the experiment results show that the Bayes model framework is considerably more flexible and effective than thought. Moreover, these findings directly extend to other types of text mining tasks, such as ranked retrieval of documents. Structured generative models such as the proposed TDM were shown to further improve modeling effectiveness, leading to new types of scalable generative models for text mining.

The generalized smoothing function presents virtually all of the commonly used smoothing functions for multinomial and n-gram models of text in a common mixture model framework. The methods differ only in the chosen discounting method, and how the smoothing coefficient is chosen. This describes the decades of statistical language modeling research in a concise form, as well as simplifies the development and analysis of new smoothing methods. Formalizing all the smoothing methods as approximate maximum likelihood estimation on a Hidden Markov Model re-establishes a probabilistic formulation for the functions. The formalization of feature transforms and weighted words as inference over probabilistic data similarly re-establishes the use of these methods in a probabilistic framework. Feature transforms were shown to greatly improved MNB performance for classification and ranking, and has potential implications for other types of generative text models.

Scalability limits the range of applications for probabilistic models. The naive inference used here as baseline is widely considered to be optimal: *"Because we have to look at the data at least once, NB can be said to have optimal time complexity."* [Manning et al., 2008]. The presented sparse inference enables improved scaling of linear models and structured extensions to different types of tasks. Unlike parallelization or approximation, sparse inference reduces the total required computation non-linearly and provides exact results. The inference can be further combined with parallelization and many other

efficiency improvements used in information retrieval and machine learning. Especially the application of structured sparse models becomes more scalable compared to naive inference.

The experimental results show that the commonly used generative models for text classification and ranking are comparatively weak baselines, whereas the modified and extended generative models have performance on par with strong task-specific methods. Among the possible models, the combination of TF-IDF weighting with uniform Jelinek-Mercer smoothing is one single-parameter option that performs well in both types of tasks. With more parameters and optimization for the specific dataset, a number of stronger models can be learned. The results show that the models developed in the thesis provide improved solutions for a variety of common applications, such as classification of sentiment, spam, e-mails, news, web-pages and Wikipedia articles, and different types of ranked retrieval of text documents. The obtained improvements should extend naturally to other tasks that process text with generative models, as well as to future text mining applications.

7.3 Revisiting the Thesis Statement

The thesis statement was posed in Chapter 1 as: *Generative models of text combined with inference using inverted indices provide sparse generative models for text mining that are both versatile and scalable, providing state-of-the-art effectiveness and high scalability for various text mining tasks.*

The theory and experiment results provide strong support to the thesis statement. Modifications and extensions to MNB models of text were formalized as well-defined graphical models. The experiments showed high effectiveness of the developed models for a variety of text classification and clustering tasks, and the obtained improvements should hold in many current and future applications of generative text modeling. The idea of Naive Bayes models as "punching bags of machine learning" should therefore be reconsidered. The theory for sparse inference was developed to produce scalability as a function of model sparsity to linear models and their structural extensions. In practice this reduced the processing times of the largest completed experiments by an order of magnitude. Given the theory and results of the thesis, the "curse of dimensionality" of high-dimensional sparse text data should perhaps be considered a useful property.

7.4 Limitations of the Thesis

The experiments in the thesis were restricted to the main applications of text classification and ad-hoc text retrieval, where performance of modified MNB models were shown to be competitive with high-performing task-specific solutions. These are by far the most successful applications of text mining, but do not cover all of the possible types of current and future text mining tasks. Some applications are less suited for the assumptions of generative multinomial models than others. Generative models estimated to maximize likelihood are neither guaranteed to give sufficient performance, if the perfomance measure is very different from maximum likelihood. For example, discriminative models directly optimizing posterior probabilities can be a more suitable choice, if high precision of posterior probabilities is required.

Over the last years a number of research directions have been proposed as widely applicable solutions for machine learning. Deep learning combining modern parallel computing hardware with developments in optimization has brought a resurgence of interest in multi-layer neural networks [Bengio, 2009, Poon and Domingos, 2011, Mikolov et al., 2011, Collobert et al., 2011]. Probabilistic programming combined with factor graphs is enabling flexible development of complex graphical model architectures [McCallum et al., 2009, Minka et al., 2012, Andres et al., 2012]. Gaussian processes are developing into a high-performing solution to a wide variety of applications [Rasmussen, 2003, Preotiuc-Pietro and Cohn, 2013]. Connections to these research directions is outside the scope of the thesis. Research in these topics will continue, but none of these frameworks currently form a paradigm for performing a variety of text processing mining tasks with high scalability.

Scalability has become an increasingly prominent topic in machine learning during the time used for conducting the thesis research. Particularly, parallelized and online stream learning algorithms have become popular in the context of the "Big Data" and data science movement. Online algorithms such as adaptive stochastic gradient descent [Duchi et al., 2011, McMahan et al., 2013] are used to learn deep neural networks, and parallelization frameworks such as MapReduce [Dean and Ghemawat, 2008] combined with cloud computing are enabling new types of applications. Both types of improvements are increasingly popular for text mining. Linear models combining parallelized online learning with approximations [Li et al., 2013, McMahan et al., 2013, Agarwal et al., 2014] have been proposed as one highly scalable solution. Extensive comparison to these developments is outside the scope of this thesis. As discussed in Chapters 4 and 5, both parallelization and stream processing can be

trivially combined in estimation and inference with the algorithms presented in this thesis. Furthermore, unlike the sparse inference developed in this thesis, these developments do not reduce the computational complexity of scaling to large numbers of labels and other latent variable nodes.

7.5 Future Work

Experiments in text mining tasks other than ranking and classification should prove the models developed in this thesis useful across text mining applications. Clustering uses the posterior probabilities from generative models in the same way as classification and ranking, and there is no reason to doubt that the shown performance improvements extend to clustering. Regression with Bayes models [Frank et al., 1998] is likely improved substantially, since TDM enables both modeling of each continuous variable value using a document-conditional distribution, and hierarchical smoothing of the value-conditional models.

One highly useful application of the models is in n-gram language models that extend the multinomial models with context variables. These have a variety of uses such as speech recognition, machine translation, text compression, text prediction and optical character recognition. The generalized smoothing and feature weighting combined with random search for metaparameters could provide superior models to basic n-gram models using Kneser-Ney smoothing. Sparse inference could be applied to language model decoding, although the most common operation with language models is the query of individual conditional probabilities of words given the context, not the computation of marginals for the Bayes rule.

The use of weighted words from feature transforms was shown to improve classification and ranking performance, but the use of arbitrary weighted words for generative models can have other uses. Topic models are commonly estimated from word count or sequence data, but the derivation presented in Chapter 5 allows the use of fractional counts for these models as well. Alternatively, the posterior probabilities of a topic model or outputs of a Non-Negative Matrix Factorization [Paatero and Tapper, 1994] could be used as features for a generative model. Non-probabilistic topic models such as Latent Semantic Analysis [Deerwester, 1988] have traditionally used TF-IDF to improve topic separation, and the same should work for generative topic models.

Sparse inference as well as the modifications to generative models were provided for the case of MNB. However, sparse inference is applicable whenever the parameters of any linear model can be represented with sparse vectors for each label and less sparse vectors for back-off nodes, with the most basic case being a single background distribution node. Inference of kernel densities over sparse data is particularly scalable. As shown with structured extensions of MNB, many types of graphs for the back-off nodes can be efficiently computed. It would therefore be valuable to know which types of linear and non-linear models can be represented in sparse forms, and how much the scalability of inference is improved.

Sparse inference algorithms reduce the complexity of inference with large numbers of labels, and more generally with large numbers of hidden variables in structured models. The TDM model examined in this thesis was an elementary expansion of a linear model with a mixture over document-conditional models. Extension of these findings into more structured cases can yield rich models that both have higher performance and offer the possibility to make different types of inference using the joint probabilities of hidden variables. For example, adding layers of topic variables could enable highly scalable topic modeling.

Combining the sparse inference with other improvements for scalability was outlined, but empirical experimentation was left for future work. The combination with parallelization should provide further linear improvements in processing speeds, but in practice parallelization of tasks across networks introduces many complications. Combining the algorithm with the other efficiency improvements for text mining mentioned in Chapter 5 can provide more than linear reductions, but the exact scale of these improvements depend on the data and structure of the model. Tree-based searches can provide considerable reductions, if the node-conditional models separate into different clusters [Ram and Gray, 2012]. Search network minimization can produce considerably smaller models, if nodes can be merged or removed with minimal loss in performance [Aubert, 2002]. A considerable literature exists in information retrieval to improve the efficiency of inverted indices, that can be directly leveraged for sparse inference [Zobel and Moffat, 2006].

Extensions of the TDM model were used by the author for the 2014 Kaggle competitions LHSTC4[1] [Puurula et al., 2014] and WISE[2] [Tsoumakas et al.]. Both competitions were large-scale multi-label text classification tasks with over a hundred competing teams. The submission to LSHTC4 won the com-

[1] http://www.kaggle.com/c/lshtc/
[2] http://www.kaggle.com/c/wise-2014

petition, while the submission for WISE placed narrowly second. TDM extended with label hierarchy nodes proved to be the most useful model, and an ensemble of sparse generative models proved to be a high-performing solution. Future work will apply models such as TDM to other cases where groundbreaking effectiveness and scalability is required.

Appendix A

Tables of Results

This appendix contains the tables of results for the experiments summarized in Chapter 6. Tables A.1 to A.9 show the results for the MNB smoothing methods. Tables A.10 to A.22 show the results for the Extended MNB models. Table A.23 shows the results for the TDM model. Tables A.24 to A.25 shows the results for the strong linear model baselines. Tables A.26 to A.33 show the training times for the scalability experiments, and tables A.34 to A.41 show the corresponding testing times. In tables A.26 to A.41 the modifier appendix "ht" indicates a hash table implementation for inference instead of an inverted index.

Table A.1: Evaluation set Micro-F1 results of smoothed MNB models on text classification datasets, optimized with 40x40 Gaussian random searches on the development set

Model	MNB							
Background	c							
Smoothing	jm	dp	ad	pd	jm_dp	pd_dp	pd_jm	pd_jm_dp
20ng	83.04	82.90	82.93	**83.22**	83.06	**83.22**	**83.22**	**83.22**
cade	60.70	59.90	60.44	60.45	**60.74**	60.46	60.68	**60.74**
r52	91.51	90.69	89.68	93.14	91.19	93.18	93.18	**93.26**
r8	96.25	94.83	95.43	95.75	96.29	95.93	**96.84**	**96.84**
webkb	83.16	83.09	83.30	83.09	83.16	83.09	83.16	**83.38**
ecue1	91.10	91.30	91.30	**91.60**	91.10	91.40	91.50	91.10
ecue2	**92.20**	**92.20**	**92.20**	**92.20**	**92.20**	**92.20**	**92.20**	**92.20**
trec06	99.31	**99.38**	**99.38**	**99.38**	**99.38**	**99.38**	99.24	99.28
tripa12	86.59	86.56	**86.71**	86.69	86.59	86.69	86.69	86.69
aclimdb	82.30	82.36	82.36	**82.40**	82.30	82.33	82.33	**82.40**
amazon12	**80.16**	79.97	79.96	80.11	**80.16**	80.11	80.12	80.08
rcv1	65.24	67.32	64.75	64.60	**67.37**	67.35	65.22	**67.37**
eurlex	49.11	48.57	48.88	48.99	49.18	48.99	49.16	**49.25**
ohsu-trec	40.30	40.31	40.11	**40.36**	40.30	**40.36**	40.34	**40.36**
DMOZ2	22.48	23.84	22.05	22.90	24.28	**24.83**	22.90	24.54
wikip_med2	28.88	28.34	28.56	29.54	28.88	29.39	29.54	**29.64**
mean	72.02	71.97	71.75	72.15	72.26	72.43	72.27	**72.52**

Table A.2: Evaluation set Micro-F1 results of smoothed MNB models on text classification datasets, optimized with 40x40 Gaussian random searches on development sets

Model	MNB							
Background	c							
Smoothing	jm	dp	ad	pd	jm_dp	pd_dp	pd_jm	pd_jm_dp
20ng	82.08	82.66	82.50	82.46	**82.67**	**82.67**	82.05	82.66
cade	**59.90**	59.76	59.83	59.83	59.74	59.84	59.83	59.83
r52	**91.90**	91.08	91.54	91.78	91.54	91.78	91.86	**91.90**
r8	**96.57**	96.43	96.43	96.02	96.39	96.39	96.07	96.48
webkb	84.02	84.02	83.73	83.73	**84.09**	84.02	83.81	**84.09**
ecue1	90.80	92.40	91.80	**92.60**	92.40	92.40	92.40	92.40
ecue2	**92.40**	**92.40**	**92.40**	**92.40**	**92.40**	**92.40**	**92.40**	**92.40**
trec06	99.28	**99.46**	99.42	99.42	**99.46**	**99.46**	99.38	99.28
tripa12	86.76	86.79	86.78	86.78	86.78	**86.80**	86.78	86.79
aclimdb	82.50	82.56	**82.60**	82.56	82.53	82.56	82.56	82.56
amazon12	**80.28**	80.21	80.21	80.24	**80.28**	80.27	80.26	80.27
rcv1	64.99	67.95	64.61	64.61	67.86	**67.96**	65.03	67.95
eurlex	48.04	45.59	45.41	45.39	48.03	45.39	47.99	**48.06**
ohsu-trec	40.19	39.85	39.54	40.06	40.19	40.06	**40.35**	40.04
DMOZ2	22.28	24.62	21.97	22.54	24.64	**24.80**	22.54	24.05
wikip_med2	30.35	29.14	29.73	30.89	30.99	30.89	30.91	**31.90**
mean	72.02	72.18	71.78	71.96	72.50	72.35	72.14	**72.54**

Table A.3: Evaluation set Micro-F1 results of smoothed MNB models on text classification datasets, optimized with 40x40 Gaussian random searches on development sets

Model	MNB							
Background	uc							
Smoothing	jm	dp	ad	pd	jm_dp	pd_dp	pd_jm	pd_jm_dp
20ng	83.06	83.03	82.79	83.22	**83.39**	83.22	83.22	83.22
cade	**60.77**	59.71	60.45	60.43	60.70	60.46	60.70	60.74
r52	91.66	91.31	91.47	92.60	**92.79**	92.56	92.56	92.67
r8	**96.57**	96.43	96.39	96.29	96.48	96.34	**96.57**	96.16
webkb	83.66	**84.02**	83.45	83.73	83.88	83.81	83.95	83.95
ecue1	91.10	92.40	91.90	**92.60**	91.60	92.40	92.40	92.40
ecue2	92.20	**92.40**	92.20	92.20	**92.40**	**92.40**	92.20	**92.40**
trec06	99.24	**99.42**	99.38	99.38	99.24	**99.42**	99.17	99.20
tripa12	86.75	86.79	**86.80**	86.74	86.77	86.73	86.74	**86.80**
aclimdb	82.43	82.53	**82.60**	82.56	82.53	82.56	82.56	82.53
amazon12	**80.27**	80.22	80.20	80.25	**80.27**	80.25	80.25	**80.27**
rcv1	65.10	**68.28**	64.61	64.60	68.03	68.07	65.00	68.12
eurlex	48.93	48.80	48.88	49.12	49.08	48.89	**49.27**	49.26
ohsu-trec	**40.60**	40.34	40.10	40.42	40.59	40.42	40.19	40.44
DMOZ2	22.40	23.93	22.05	22.50	24.52	**25.16**	22.56	25.02
wikip_med2	30.35	29.18	30.43	30.98	31.17	**31.91**	30.97	30.93
mean	72.19	72.42	72.10	72.35	72.71	**72.79**	72.39	72.76

Table A.4: Evaluation set NDCG@20 results of smoothed MNB models on text retrieval datasets, optimized with 50x8 Gaussian random searches on development sets

Model	MNB							
Background	c							
Smoothing	jm	dp	ad	pd	jm_dp	pd_dp	pd_jm	pd_jm_dp
fire_en	49.06	50.43	50.47	50.11	50.27	50.71	50.10	**50.90**
ohsu_trec	23.18	22.98	**23.75**	23.57	23.03	22.79	23.26	22.95
trec_ap	30.38	29.86	29.79	30.79	30.67	**31.48**	30.90	30.97
trec_cr	26.80	26.52	26.73	26.02	27.61	26.56	25.93	**27.76**
trec_doe	26.84	28.66	28.24	29.16	28.95	28.84	28.88	**30.38**
trec_fbis	31.90	**33.96**	31.64	33.17	31.63	33.17	32.05	32.95
trec_fr	19.68	22.43	21.68	23.16	22.88	23.41	23.09	**24.25**
trec_ft	27.74	33.18	31.87	31.62	32.87	32.86	31.42	**33.97**
trec_la	25.99	25.45	**27.74**	26.67	25.83	26.67	26.61	25.64
trec_pt	15.65	14.24	18.24	**23.23**	17.30	**23.23**	17.33	17.30
trec_sjmn	27.66	**29.53**	26.81	25.97	27.66	26.14	26.65	25.98
trec_wsj	32.31	33.94	33.08	33.62	34.18	34.88	33.59	**35.15**
trec_zf	26.70	24.42	24.12	27.64	26.09	27.64	27.64	**28.84**
mean	27.99	28.89	28.78	29.59	29.15	**29.88**	29.04	29.77

Table A.5: Evaluation set MAP results of smoothed MNB models on text retrieval datasets, optimized with 50x8 Gaussian random searches on development sets

Model	MNB							
Background	c							
Smoothing	jm	dp	ad	pd	jm_dp	pd_dp	pd_jm	pd_jm_dp
fire_en	40.72	41.95	41.45	41.55	**42.05**	41.55	41.55	41.54
ohsu_trec	20.95	20.40	21.54	**21.74**	21.41	21.60	**21.74**	20.41
trec_ap	19.19	18.40	18.36	19.22	19.24	19.25	19.15	**19.26**
trec_cr	**17.39**	16.07	17.17	17.30	16.49	17.30	17.30	17.32
trec_doe	17.24	20.19	19.37	20.33	20.05	**20.57**	17.29	17.21
trec_fbis	22.47	25.93	22.92	23.09	**26.08**	25.86	22.72	23.58
trec_fr	14.06	17.17	12.26	16.59	17.17	**18.05**	16.54	17.09
trec_ft	19.95	22.78	20.96	21.78	**22.88**	22.59	21.73	21.89
trec_la	16.41	16.13	16.49	16.37	16.63	16.51	16.37	**16.99**
trec_pt	11.06	11.96	**18.44**	11.00	11.07	11.00	11.06	11.08
trec_sjmn	14.72	20.68	**23.16**	16.73	20.86	16.73	14.48	14.41
trec_wsj	18.55	19.33	18.27	19.06	19.58	19.52	19.07	**19.72**
trec_zf	15.42	15.00	14.93	16.69	16.87	16.69	16.70	**17.75**
mean	19.09	20.46	20.41	20.11	**20.80**	20.56	19.67	19.87

Table A.6: Evaluation set NDCG@20 results of smoothed MNB models on text retrieval datasets, optimized with 50x8 Gaussian random searches on development sets

Model	MNB							
Background	u							
Smoothing	jm	dp	ad	pd	jm_dp	pd_dp	pd_jm	pd_jm_dp
fire_en	47.84	46.55	48.64	48.48	49.17	**49.25**	48.54	48.61
ohsu_trec	20.88	21.01	**22.25**	21.10	21.15	21.37	21.13	21.18
trec_ap	25.23	24.67	24.72	**26.56**	25.26	**26.56**	26.07	26.53
trec_cr	26.54	24.62	27.54	**27.77**	26.56	27.27	25.91	25.91
trec_doe	22.62	22.22	23.22	**23.40**	22.62	23.14	22.62	22.81
trec_fbis	20.34	24.02	**25.54**	24.98	21.74	24.99	23.56	24.48
trec_fr	12.97	12.68	**14.18**	12.63	13.17	12.63	12.63	12.62
trec_ft	26.66	24.42	27.32	28.05	27.31	27.33	**28.09**	28.04
trec_la	23.45	20.82	22.51	24.56	24.71	24.62	**24.94**	24.64
trec_pt	25.50	22.72	**26.40**	24.66	23.01	24.66	24.66	25.62
trec_sjmn	26.78	26.80	**28.74**	28.52	27.80	28.52	28.45	28.48
trec_wsj	27.42	22.75	26.83	**28.43**	28.26	**28.43**	28.08	27.61
trec_zf	**14.52**	10.36	11.20	11.67	14.38	12.65	14.02	12.55
mean	24.67	23.36	25.31	25.45	25.01	**25.49**	25.28	25.31

Table A.7: Evaluation set MAP results of smoothed MNB models on text retrieval datasets, optimized with 50x8 Gaussian random searches on development sets

Model	MNB							
Background	u							
Smoothing	jm	dp	ad	pd	jm_dp	pd_dp	pd_jm	pd_jm_dp
fire_en	36.71	37.29	37.97	38.06	**38.60**	38.03	38.02	38.05
ohsu_trec	17.13	17.26	**18.47**	17.77	17.44	17.85	17.79	17.85
trec_ap	14.70	13.11	13.78	14.87	14.70	14.88	14.91	**14.92**
trec_cr	15.25	14.53	16.36	16.97	15.25	**16.99**	15.62	15.34
trec_doe	14.72	13.27	14.62	**15.59**	14.72	**15.59**	14.75	14.76
trec_fbis	11.87	**16.75**	16.37	16.51	13.47	16.45	16.49	15.04
trec_fr	9.01	8.90	**10.12**	8.60	9.55	8.62	8.60	9.88
trec_ft	16.52	14.18	16.09	16.84	16.80	16.84	16.89	**16.95**
trec_la	13.46	11.90	12.82	14.16	14.01	14.17	14.20	**14.22**
trec_pt	8.84	15.49	**15.93**	13.90	15.51	15.71	13.90	14.07
trec_sjmn	20.37	20.88	20.95	**21.20**	20.37	**21.20**	**21.20**	**21.20**
trec_wsj	14.08	11.76	13.40	14.63	**14.82**	14.63	14.69	14.57
trec_zf	**9.53**	6.77	7.45	7.31	8.74	7.30	7.81	7.30
mean	15.55	15.55	16.49	16.65	16.46	**16.79**	16.53	16.47

Table A.8: Evaluation set NDCG@20 results of smoothed MNB models on text retrieval datasets, optimized with 50x8 Gaussian random searches on development sets

Model	MNB							
Background	uc							
Smoothing	jm	dp	ad	pd	jm_dp	pd_dp	pd_jm	pd_jm_dp
fire_en	49.14	50.83	50.68	50.91	51.80	**51.89**	50.96	51.64
ohsu_trec	23.08	23.32	**23.71**	23.48	23.13	23.09	23.17	23.33
trec_ap	30.42	29.70	29.75	30.92	**31.02**	30.79	30.87	30.86
trec_cr	26.80	26.52	26.71	26.04	26.45	**28.22**	25.88	26.39
trec_doe	26.84	28.84	28.42	29.16	27.73	**29.45**	28.88	28.64
trec_fbis	31.93	**33.77**	31.67	33.17	31.90	33.17	32.05	32.71
trec_fr	20.26	22.35	21.64	**23.15**	22.43	22.47	23.10	22.94
trec_ft	26.31	32.89	31.86	31.29	33.01	**33.71**	30.28	32.99
trec_la	25.94	25.37	**27.74**	26.62	25.83	26.41	26.30	25.70
trec_pt	15.65	19.06	**26.54**	22.19	22.34	15.47	22.31	22.22
trec_sjmn	27.67	**29.51**	27.55	26.93	22.68	26.83	27.42	28.13
trec_wsj	32.31	34.10	33.07	33.53	34.33	**35.27**	33.62	34.38
trec_zf	26.41	26.19	24.03	29.07	**29.39**	28.80	28.27	28.76
mean	27.90	29.42	29.49	29.73	29.39	29.66	29.47	**29.90**

Table A.9: Evaluation set MAP results of smoothed MNB models on text retrieval datasets, optimized with 50x8 Gaussian random searches on development sets

Model	MNB							
Background	uc							
Smoothing	jm	dp	ad	pd	jm_dp	pd_dp	pd_jm	pd_jm_dp
fire_en	40.81	41.93	41.41	41.89	**42.25**	41.84	41.84	42.06
ohsu_trec	20.91	21.52	21.54	**21.74**	20.48	21.15	21.61	20.44
trec_ap	19.18	18.98	18.23	18.97	19.16	**19.22**	19.06	19.03
trec_cr	**17.34**	16.80	17.17	17.30	16.56	17.30	17.12	17.30
trec_doe	17.23	20.46	19.38	20.33	20.37	**20.58**	17.87	20.45
trec_fbis	22.45	25.61	22.91	23.10	24.01	25.98	22.58	**26.17**
trec_fr	13.98	17.62	12.26	16.65	17.16	17.34	16.69	**17.63**
trec_ft	19.54	22.79	20.95	21.78	22.43	22.74	21.16	**22.85**
trec_la	16.38	16.13	16.50	16.36	16.66	16.18	16.38	**16.82**
trec_pt	11.12	15.11	**15.72**	13.25	11.08	15.69	12.64	15.70
trec_sjmn	14.69	20.76	19.07	21.12	14.69	**21.74**	16.60	15.72
trec_wsj	18.61	19.30	18.27	19.08	19.24	**19.69**	18.96	19.22
trec_zf	15.46	16.39	14.86	16.52	15.87	16.52	16.62	**16.76**
mean	19.05	21.03	19.87	20.62	20.00	**21.23**	19.93	20.78

Table A.10: Evaluation set Micro-F1 results of Extended MNB models and MNB baselines on text classification datasets, optimized with 40x40 Gaussian random searches on development sets

Model	MNB							
Background	c				u			
Smoothing	pd_dp				jm_dp			
Factors	po	ps	po_ps	-	po	ps	po_ps	-
20ng	**83.23**	83.22	83.20	83.22	82.71	82.67	82.75	82.67
cade	60.46	60.85	**60.88**	60.46	60.44	59.68	60.44	59.74
r52	**93.18**	92.71	92.67	**93.18**	91.90	91.90	91.62	91.54
r8	95.70	96.29	95.84	95.93	**96.61**	96.57	**96.61**	96.39
webkb	83.23	83.09	83.38	83.09	84.02	**84.09**	84.02	**84.09**
ecue1	91.10	91.50	91.50	91.40	**92.70**	92.40	91.70	92.40
ecue2	92.20	92.20	92.20	92.20	**92.40**	**92.40**	**92.40**	**92.40**
trec06	99.38	99.38	99.38	99.38	**99.46**	99.42	99.20	**99.46**
tripa12	86.68	86.69	86.69	86.69	**86.80**	86.79	86.79	86.78
aclimdb	82.40	82.40	82.30	82.33	**82.56**	82.53	82.53	82.53
amazon12	80.11	80.17	80.16	80.11	**80.33**	80.28	**80.33**	80.28
rcv1	67.35	67.66	67.77	67.35	68.70	68.03	**68.71**	67.86
eurlex	48.86	**49.06**	48.94	48.99	48.10	47.97	47.83	48.03
ohsu-trec	40.36	40.36	40.36	40.36	**40.43**	**40.43**	40.07	40.19
DMOZ2	24.85	**25.44**	**25.44**	24.83	23.87	24.77	25.21	24.64
wikip_med2	29.56	30.52	30.51	29.39	**31.92**	30.98	31.64	30.99
mean	72.42	72.60	72.58	72.43	**72.68**	72.56	72.62	72.50

Table A.11: Evaluation set Micro-F1 results of Extended MNB models on text classification datasets, optimized with 40x40 Gaussian random searches on development sets

Model	MNB							
Background	uc		u		uc		u	
Smoothing	pd_dp				jm			
Weighting	qidf	ti	qidf	ti	qidf	ti	qidf	ti
Factors	ps							
20ng	82.53	82.81	82.59	82.30	83.06	**83.62**	82.77	83.32
cade	**61.12**	59.26	59.05	57.10	60.10	58.26	58.71	57.50
r52	**93.06**	92.60	92.17	91.51	90.14	92.60	90.10	92.60
r8	**96.52**	96.20	96.48	96.25	96.48	**96.52**	95.70	**96.52**
webkb	82.95	84.74	82.59	85.17	82.30	85.53	82.52	**85.67**
ecue1	95.10	96.70	95.10	**96.90**	95.10	94.80	95.10	95.80
ecue2	**99.30**	98.90	99.20	99.00	**99.30**	98.50	**99.30**	98.60
trec06	**99.38**	99.35	**99.38**	99.20	**99.38**	99.31	**99.38**	**99.38**
tripa12	88.08	87.16	88.08	87.38	88.14	87.65	**88.20**	87.67
aclimdb	83.93	**85.80**	83.93	85.76	84.20	85.23	84.20	85.30
amazon12	79.53	81.37	79.53	81.37	79.67	**81.51**	79.67	81.46
rcv1	68.75	**71.29**	68.76	71.25	63.03	68.93	63.36	69.08
eurlex	49.56	49.55	47.92	47.90	49.23	50.12	49.41	**50.27**
ohsu-trec	40.32	40.22	40.31	40.20	**40.60**	40.47	**40.60**	40.43
DMOZ2	24.70	**27.05**	25.08	26.73	21.45	24.55	20.87	24.55
wikip_med2	30.03	**31.35**	29.62	31.22	26.52	30.28	26.38	30.26
mean	73.43	**74.02**	73.11	73.70	72.42	73.62	72.27	73.65

Table A.12: Evaluation set Micro-F1 results of Extended MNB models on text classification datasets, optimized with 40x40 Gaussian random searches on development sets

Model	MNB							
Background	uc		u		uc		u	
Smoothing	jm_dp				pd			
Weighting	qidf	ti	qidf	ti	qidf	ti	qidf	ti
Factors	ps							
20ng	83.03	**83.51**	82.65	83.35	82.57	82.26	82.46	82.27
cade	59.74	59.05	59.62	57.52	**61.40**	58.92	58.70	57.08
r52	92.87	93.02	91.08	92.44	**93.18**	92.52	92.91	91.51
r8	**96.71**	96.48	96.16	96.52	96.48	96.34	96.25	96.25
webkb	82.37	**85.45**	82.30	84.88	82.30	84.88	82.44	85.17
ecue1	95.20	95.30	95.20	95.50	95.10	96.60	95.10	**96.80**
ecue2	99.20	99.00	**99.30**	98.70	**99.30**	98.40	99.20	98.80
trec06	99.38	99.28	99.38	**99.46**	99.35	99.10	99.35	99.17
tripa12	88.10	87.66	**88.18**	87.66	88.17	87.12	88.16	87.13
aclimdb	84.20	**85.86**	84.20	85.83	83.86	85.06	83.93	85.06
amazon12	79.67	**81.52**	79.67	**81.52**	79.52	81.34	79.54	81.34
rcv1	68.54	**71.51**	68.54	71.22	62.42	69.66	71.29	69.54
eurlex	49.36	**50.08**	49.48	49.85	49.59	49.49	49.55	47.84
ohsu-trec	**40.60**	40.00	**40.60**	40.49	40.32	39.29	40.31	39.31
DMOZ2	24.91	**26.99**	24.95	26.70	20.97	24.93	20.97	24.95
wikip_med2	29.55	31.69	29.32	**32.32**	26.99	28.76	26.99	28.74
mean	73.34	**74.15**	73.16	74.00	72.59	73.42	72.95	73.18

Table A.13: Evaluation set Micro-F1 results of Extended MNB models on text classification datasets, optimized with 40x40 Gaussian random searches on development sets

Model	MNB							
Background	uc		u		uc		u	
Smoothing	pd				dp			
Weighting	qidf	ti	qidf	ti	qidf	ti	qidf	ti
Factors	ps							
20ng	82.99	**83.67**	82.63	83.35	82.66	83.18	82.94	83.24
cade	**61.15**	58.72	58.71	58.14	59.46	59.70	59.21	57.71
r52	**93.14**	92.56	92.95	91.51	88.51	88.74	88.31	87.73
r8	**96.61**	96.29	96.29	96.48	93.92	94.24	93.92	94.01
webkb	82.16	84.74	82.44	**85.31**	82.30	84.52	82.37	84.52
ecue1	95.20	**96.80**	95.20	96.70	93.50	93.90	93.50	95.80
ecue2	99.20	98.50	**99.30**	98.80	**99.30**	98.30	99.20	98.50
trec06	**99.38**	99.31	**99.38**	**99.38**	**99.38**	99.28	**99.38**	99.17
tripa12	88.19	87.66	**88.20**	87.65	88.11	87.40	88.12	87.41
aclimdb	84.00	85.46	84.03	85.46	83.96	**85.83**	83.96	**85.83**
amazon12	79.60	**81.52**	79.66	81.51	79.45	81.47	79.45	81.47
rcv1	63.34	69.40	63.35	69.38	68.91	**71.12**	68.56	71.00
eurlex	49.63	49.77	49.48	**49.89**	49.22	49.46	48.14	45.31
ohsu-trec	40.44	40.39	**40.61**	40.37	39.99	40.03	40.05	40.17
DMOZ2	21.51	24.82	21.12	24.88	24.53	**26.99**	24.58	26.74
wikip_med2	26.99	29.76	26.98	29.76	28.27	30.23	28.09	**31.63**
mean	72.72	**73.71**	72.52	73.66	72.59	73.40	72.49	73.14

Table A.14: Evaluation set Micro-F1 results of Extended MNB models on text classification datasets, optimized with 40x40 Gaussian random searches on development sets

Model	MNB							
Background	uc	u	uc	u	uc	u	uc	u
Smoothing	pd_dp	jm_dp	pd_dp	jm_dp	pd_dp	jm_dp	pd_dp	jm_dp
Weighting	ti		tXi		tiX		tXiX	
Factors	ps							
20ng	83.04	83.26	83.87	83.74	83.31	83.42	**84.23**	83.55
cade	59.25	57.55	59.62	57.56	59.55	57.96	**60.17**	57.78
r52	88.43	90.84	91.90	**92.83**	87.92	92.64	92.60	92.56
r8	93.96	95.70	96.34	96.39	94.79	**96.52**	95.56	96.34
webkb	82.37	83.59	86.17	87.03	85.67	85.88	87.75	**88.25**
ecue1	96.40	96.30	96.80	94.30	96.90	96.00	**97.10**	95.30
ecue2	97.80	98.90	98.50	98.20	98.90	98.50	**99.10**	98.80
trec06	99.35	99.24	**99.46**	99.38	99.35	99.42	99.28	99.42
tripa12	87.15	87.65	89.06	89.20	87.63	87.81	**89.33**	**89.33**
aclimdb	85.36	85.73	**86.80**	86.73	85.90	85.80	86.53	86.50
amazon12	81.36	81.46	81.95	81.96	81.50	81.56	81.95	**82.06**
rcv1	71.45	71.51	71.22	71.06	70.73	70.85	71.25	**71.56**
eurlex	49.05	48.84	49.63	**50.11**	49.08	50.06	48.96	50.04
ohsu-trec	40.13	40.47	40.34	40.41	40.31	40.21	**40.74**	40.52
DMOZ2	26.26	26.76	**27.17**	27.07	26.28	26.81	26.60	26.52
wikip_med2	30.68	**32.60**	30.93	32.21	31.94	31.79	30.82	32.29
mean	73.25	73.77	74.36	74.26	73.73	74.08	**74.50**	74.43

Table A.15: Evaluation set NDCG@20 results of Extended MNB models on text retrieval datasets, optimized with 50x8 Gaussian random searches on development sets

Model	MNB							
Background	uc		u		uc		u	
Smoothing	pd_dp				jm			
Weighting	qidf	ti	qidf	ti	qidf	ti	qidf	ti
fire_en	**53.91**	49.37	52.22	51.67	51.30	50.60	51.30	50.66
ohsu_trec	23.75	23.76	23.76	**23.78**	23.21	**23.78**	23.28	**23.78**
trec_ap	31.68	29.64	**32.50**	31.95	31.71	30.74	31.39	31.33
trec_cr	27.15	26.02	**32.98**	27.54	27.91	26.47	32.86	29.69
trec_doe	29.28	25.91	28.90	27.08	**29.53**	25.18	28.44	26.34
trec_fbis	34.41	26.21	**36.51**	30.00	33.34	32.73	32.39	33.29
trec_fr	23.54	23.10	23.88	**23.93**	23.22	21.96	22.17	21.51
trec_ft	32.69	28.25	**33.17**	32.50	29.87	30.09	30.99	31.94
trec_la	28.31	26.12	**28.75**	28.55	28.26	27.79	28.48	27.80
trec_pt	36.79	31.09	**36.85**	32.94	33.25	35.95	36.65	35.95
trec_sjmn	26.52	24.45	27.59	27.57	26.78	27.44	**30.34**	26.87
trec_wsj	35.51	35.33	**35.72**	35.43	34.52	33.44	34.52	33.52
trec_zf	29.82	30.67	29.71	**30.70**	28.11	26.92	28.75	27.93
mean	31.80	29.23	**32.50**	31.05	30.85	30.24	31.66	30.82

Table A.16: Evaluation set MAP results of Extended MNB models on text retrieval datasets, optimized with 50x8 Gaussian random searches on development sets

Model	MNB							
Background	uc		u		uc		u	
Smoothing	pd_dp				jm			
Weighting	qidf	ti	qidf	ti	qidf	ti	qidf	ti
fire_en	43.27	42.15	**43.28**	42.98	42.05	42.45	42.42	42.45
ohsu_trec	**22.73**	22.63	22.70	22.44	22.05	22.39	22.08	22.31
trec_ap	**20.55**	18.93	20.20	20.40	20.15	20.30	19.90	20.29
trec_cr	17.31	15.88	**21.17**	18.90	17.02	16.61	19.70	19.18
trec_doe	19.39	17.41	**19.84**	18.47	18.99	16.82	19.30	18.36
trec_fbis	26.21	22.46	26.18	**26.48**	24.97	24.38	23.38	24.21
trec_fr	**19.33**	17.01	17.48	17.64	17.35	16.68	16.44	16.00
trec_ft	23.06	20.50	22.50	**23.53**	22.90	22.68	21.84	22.89
trec_la	17.92	15.56	17.90	**17.98**	17.19	17.09	17.62	17.32
trec_pt	25.59	23.54	25.22	22.43	25.51	**26.81**	26.70	**26.81**
trec_sjmn	18.42	16.27	22.57	18.45	16.97	17.45	**23.44**	17.27
trec_wsj	**20.94**	20.63	20.51	20.52	19.78	19.71	19.91	19.70
trec_zf	**20.11**	17.54	18.53	15.37	17.66	16.67	18.22	18.11
mean	22.68	20.81	**22.93**	21.97	21.74	21.54	22.38	21.92

Table A.17: Evaluation set NDCG@20 results of Extended MNB models on text retrieval datasets, optimized with 50x8 Gaussian random searches on development sets

Model	MNB							
Background	uc		u		uc		u	
Smoothing	jm_dp				pd			
Weighting	qidf	ti	qidf	ti	qidf	ti	qidf	ti
fire_en	51.96	50.55	**53.12**	50.59	52.88	49.44	52.63	49.48
ohsu_trec	23.91	23.78	23.77	23.78	23.75	**23.99**	23.90	23.75
trec_ap	31.32	30.79	32.49	31.33	32.16	29.86	**32.50**	30.07
trec_cr	27.91	26.51	32.22	29.56	26.34	25.89	**32.98**	27.69
trec_doe	**29.47**	25.18	28.26	28.23	29.15	28.87	28.90	29.27
trec_fbis	35.09	34.71	32.39	33.18	34.42	26.25	**36.08**	29.98
trec_fr	22.96	23.40	22.17	21.17	**25.32**	22.33	23.88	22.42
trec_ft	32.63	29.02	**33.41**	32.25	31.90	27.61	33.17	29.38
trec_la	26.17	26.70	28.08	27.62	28.07	25.90	**29.18**	25.71
trec_pt	34.09	35.95	**38.78**	35.95	36.92	31.13	36.85	35.18
trec_sjmn	26.13	27.62	26.99	27.07	**27.84**	24.46	27.38	27.28
trec_wsj	34.54	33.52	**36.26**	35.30	35.93	31.09	36.09	32.46
trec_zf	28.11	26.85	28.81	30.11	29.76	30.71	29.80	**31.41**
mean	31.10	30.35	32.06	31.24	31.88	29.04	**32.56**	30.32

Table A.18: Evaluation set MAP results of Extended MNB models on text retrieval datasets, optimized with 50x8 Gaussian random searches on development sets

Model	MNB							
Background	uc		u		uc		u	
Smoothing	jm_dp				pd			
Weighting	qidf	ti	qidf	ti	qidf	ti	qidf	ti
fire_en	42.60	42.46	**43.93**	42.45	43.29	42.57	43.31	42.66
ohsu_trec	**22.69**	22.38	22.43	22.31	22.64	22.54	22.60	22.22
trec_ap	20.12	20.22	20.20	20.30	**20.53**	17.70	20.15	18.63
trec_cr	17.01	16.59	19.35	19.18	17.32	15.63	**21.17**	17.60
trec_doe	19.65	16.82	19.30	19.14	19.41	18.79	**19.84**	19.11
trec_fbis	26.21	24.42	23.39	24.21	**26.39**	18.40	26.18	20.13
trec_fr	17.36	17.49	17.58	18.17	**19.12**	16.91	18.57	16.61
trec_ft	**23.50**	22.74	21.83	22.78	22.95	21.10	22.50	21.39
trec_la	17.47	17.09	17.37	17.32	17.78	15.61	**17.93**	15.91
trec_pt	24.81	**26.81**	24.81	**26.81**	26.61	23.37	26.61	24.73
trec_sjmn	16.61	15.72	**23.44**	18.45	18.42	15.50	22.57	18.00
trec_wsj	19.57	19.70	20.22	19.70	**20.79**	17.52	20.48	18.44
trec_zf	17.66	18.26	18.45	18.12	**19.78**	16.25	18.44	16.08
mean	21.94	21.59	22.49	22.23	22.70	20.14	**23.10**	20.89

Table A.19: Evaluation set NDCG@20 results of Extended MNB models on text retrieval datasets, optimized with 50x8 Gaussian random searches on development sets

Model	MNB							
Background	uc		u		uc		u	
Smoothing	pd_jm				dp			
Weighting	qidf	ti	qidf	ti	qidf	ti	qidf	ti
fire_en	52.58	50.57	**52.73**	50.67	51.62	51.62	51.63	51.62
ohsu_trec	23.60	23.84	**23.86**	23.72	23.27	23.02	23.24	23.38
trec_ap	32.00	30.79	**32.53**	31.02	31.16	31.92	30.79	31.90
trec_cr	**34.24**	28.50	34.23	29.69	28.28	16.00	27.69	28.44
trec_doe	29.04	25.97	28.90	**29.58**	27.12	24.52	26.29	28.03
trec_fbis	34.34	33.21	35.59	32.86	**36.27**	35.58	34.92	34.99
trec_fr	24.00	21.86	23.63	21.80	21.66	**24.46**	21.17	24.17
trec_ft	31.81	28.25	**33.21**	31.98	32.05	32.42	30.28	32.43
trec_la	28.39	27.80	**29.26**	27.79	25.20	28.69	24.28	28.55
trec_pt	34.09	35.95	**38.48**	35.99	36.87	30.04	37.33	36.29
trec_sjmn	27.38	25.88	27.39	26.88	26.39	26.94	**30.67**	27.29
trec_wsj	**36.08**	33.41	35.11	33.41	32.30	35.32	30.18	35.47
trec_zf	29.16	26.92	29.92	28.70	28.61	30.40	21.32	**30.77**
mean	32.06	30.23	**32.68**	31.08	30.83	30.07	29.98	31.80

Table A.20: Evaluation set MAP results of Extended MNB models on text retrieval datasets, optimized with 50x8 Gaussian random searches on development sets

Model	MNB							
Background	uc		u		uc		u	
Smoothing	pd_jm				dp			
Weighting	qidf	ti	qidf	ti	qidf	ti	qidf	ti
fire_en	**43.34**	42.45	43.26	42.46	42.25	43.00	42.25	43.04
ohsu_trec	22.64	22.64	22.58	22.44	**22.65**	22.25	22.38	22.27
trec_ap	20.32	20.30	20.20	20.31	19.05	20.36	18.41	**20.38**
trec_cr	17.23	16.72	**21.42**	19.18	17.58	16.12	17.20	18.16
trec_doe	19.31	16.89	19.84	18.97	**20.30**	19.55	18.96	19.13
trec_fbis	26.34	24.42	25.63	24.21	27.01	21.98	26.81	**27.74**
trec_fr	**19.43**	17.01	18.57	16.00	18.32	18.16	16.30	17.71
trec_ft	22.95	22.76	22.36	22.78	22.62	**23.42**	20.89	23.38
trec_la	**17.97**	17.10	17.91	17.31	17.24	17.89	15.14	17.96
trec_pt	25.07	26.80	25.45	26.81	25.29	25.88	25.29	**26.94**
trec_sjmn	17.88	16.69	18.22	17.28	17.08	17.06	**24.38**	18.45
trec_wsj	**20.82**	19.71	20.51	19.70	19.37	20.61	16.32	20.67
trec_zf	**19.16**	16.28	18.47	18.11	18.20	19.08	14.62	19.08
mean	22.50	21.52	22.65	21.97	22.07	21.95	21.46	**22.68**

Table A.21: Evaluation set NDCG@20 results of Extended MNB models on text retrieval datasets, optimized with 50x8 Gaussian random searches on development sets

Model	MNB							
Background	u							
Smoothing	pd_dp	pd_jm	pd_dp	pd_jm	pd_dp	pd_jm	pd_dp	pd_jm
Weighting	qidfX		tXi		tiX		tXiX	
fire_en	51.57	**52.35**	51.76	50.68	49.82	49.61	42.63	50.06
ohsu_trec	23.77	23.97	23.89	23.96	23.66	**24.16**	23.74	23.64
trec_ap	**32.87**	32.84	31.87	31.04	31.79	31.12	29.46	30.05
trec_cr	30.73	31.08	28.99	29.86	30.01	30.91	28.73	**31.51**
trec_doe	28.90	**28.96**	27.55	26.69	28.77	26.96	27.31	26.63
trec_fbis	**36.23**	36.04	30.24	33.18	30.19	31.30	36.16	32.16
trec_fr	22.75	**24.35**	24.01	21.63	24.02	22.42	11.21	22.12
trec_ft	32.90	**32.93**	31.45	32.05	30.86	30.14	30.99	30.35
trec_la	29.12	**29.50**	28.56	27.87	26.52	28.30	26.79	28.04
trec_pt	34.18	28.95	32.99	**38.68**	14.42	27.50	14.71	28.41
trec_sjmn	27.40	27.55	27.24	26.91	28.01	26.02	**28.19**	26.75
trec_wsj	35.24	35.11	**35.48**	33.40	32.31	34.33	32.48	34.25
trec_zf	30.74	**30.75**	30.72	28.28	30.35	27.77	25.95	28.39
mean	**32.03**	31.88	31.13	31.09	29.29	30.04	27.56	30.18

Table A.22: Evaluation set MAP results of Extended MNB models on text retrieval datasets, optimized with 50x8 Gaussian random searches on development sets

Model	MNB							
Background	u							
Smoothing	pd_dp	pd_jm	pd_dp	pd_jm	pd_dp	pd_jm	pd_dp	pd_jm
Weighting	qidfX		tXi		tiX		tXiX	
fire_en	51.57	**52.35**	51.76	50.68	49.82	49.61	42.63	50.06
ohsu_trec	23.77	23.97	23.89	23.96	23.66	**24.16**	23.74	23.64
trec_ap	**32.87**	32.84	31.87	31.04	31.79	31.12	29.46	30.05
trec_cr	30.73	31.08	28.99	29.86	30.01	30.91	28.73	**31.51**
trec_doe	28.90	**28.96**	27.55	26.69	28.77	26.96	27.31	26.63
trec_fbis	**36.23**	36.04	30.24	33.18	30.19	31.30	36.16	32.16
trec_fr	22.75	**24.35**	24.01	21.63	24.02	22.42	11.21	22.12
trec_ft	32.90	**32.93**	31.45	32.05	30.86	30.14	30.99	30.35
trec_la	29.12	**29.50**	28.56	27.87	26.52	28.30	26.79	28.04
trec_pt	34.18	28.95	32.99	**38.68**	14.42	27.50	14.71	28.41
trec_sjmn	27.40	27.55	27.24	26.91	28.01	26.02	**28.19**	26.75
trec_wsj	35.24	35.11	**35.48**	33.40	32.31	34.33	32.48	34.25
trec_zf	30.74	**30.75**	30.72	28.28	30.35	27.77	25.95	28.39
mean	**32.03**	31.88	31.13	31.09	29.29	30.04	27.56	30.18

Table A.23: Evaluation set Micro-F1 results of TDM models on text classification datasets, optimized with 40x40 Gaussian random searches on development sets

Model	TDM							
Background	u	u	u	uc	u	u	u	uc
Smoothing	jm dp	jm dp	jm dp	dp dp	jm	pd pd	pd jm dp	jm dp
Kernel	pd dp	jm dp	pd jm dp	pd jm dp	pd dp	pd dp	pd dp	pd dp
20ng	83.96	83.90	83.95	83.75	83.56	83.90	83.67	**84.01**
cade	62.62	61.95	62.15	62.80	62.33	62.66	62.71	**63.86**
r52	91.16	92.48	92.01	91.97	**93.38**	92.17	92.28	92.36
r8	**97.12**	95.97	96.02	96.02	97.03	95.97	97.03	96.02
webkb	85.95	**86.03**	85.74	85.60	85.95	85.02	85.53	85.53
ecue1	96.10	95.90	**97.00**	96.10	96.40	96.80	96.40	96.60
ecue2	98.90	**99.00**	**99.00**	98.70	98.60	98.70	98.70	98.10
trec06	99.67	**99.71**	**99.71**	**99.71**	**99.71**	99.67	**99.71**	**99.71**
tripa12	**88.65**	88.46	88.63	88.55	88.59	88.53	88.49	88.58
aclimdb	**88.36**	88.23	88.26	88.20	88.23	88.16	88.26	88.16
amazon12	86.61	**86.77**	86.62	86.61	86.63	86.62	86.61	86.50
rcv1	73.89	73.89	**73.91**	**73.91**	73.51	73.51	73.79	73.79
eurlex	48.20	48.20	**49.50**	**49.50**	48.55	48.55	49.29	49.29
ohsu-trec	40.11	40.11	**40.59**	**40.59**	40.37	40.37	**40.59**	**40.59**
DMOZ2	26.53	26.38	26.33	**26.66**	25.60	25.60	25.93	25.75
wikip_med2	32.47	33.21	32.58	33.12	30.84	32.00	31.33	**33.50**
mean	75.02	75.01	75.12	75.11	74.96	74.89	75.02	**75.15**

Table A.24: Evaluation set Micro-F1 results of LR and SVM models on text classification datasets, optimized with 40x8 Gaussian random searches on development sets

Model	lr	l2svm	l2svm	lr	lr	l2svm	l2svm	lr
Regularizer	l2r	l2r	l1r	l1r	l2r	l2r	l1r	l1r
Weighting	-				tXiX			
20ng	80.28	80.73	76.58	78.87	83.79	**84.92**	81.30	80.67
cade	57.86	57.79	56.89	51.05	**59.74**	59.46	59.21	58.89
r52	93.06	93.22	93.38	93.30	92.28	**95.13**	94.19	94.19
r8	97.03	97.03	96.93	96.84	97.03	**97.57**	96.80	96.80
webkb	91.33	91.26	89.18	90.40	91.97	**92.19**	91.83	91.69
ecue1	95.20	95.30	96.20	95.50	95.50	**97.40**	97.10	96.20
ecue2	98.50	97.50	97.40	97.60	97.60	**99.00**	96.90	96.50
trec06	99.42	99.38	99.64	99.60	99.49	**99.71**	99.64	99.56
tripa12	92.10	91.91	91.31	91.70	92.51	**92.62**	91.98	91.91
aclimdb	90.36	89.93	89.50	89.23	89.93	**90.83**	89.80	89.86
amazon12	84.32	84.01	84.26	**84.47**	84.35	84.26	84.00	84.14
rcv1	NA	NA	NA	NA	NA	NA	NA	NA
eurlex	NA	NA	NA	NA	NA	NA	NA	NA
ohsu-trec	NA	NA	NA	NA	NA	NA	NA	NA
DMOZ2	NA	NA	NA	NA	NA	NA	NA	NA
wikip_med2	NA	NA	NA	NA	NA	NA	NA	NA
mean	89.04	88.91	88.30	88.05	89.47	**90.28**	89.34	89.13

Table A.25: Evaluation set NDCG@20 and MAP results of VSM and BM25 modes on text classification datasets, optimized with 50x8 Gaussian random searches on development sets

Measure	NDCG@20				MAP			
Model	vsm			bm25	vsm			bm25
Weighting	-	ti	tXiX	-	-	ti	tXiX	-
fire_en	39.42	45.59	46.80	**53.08**	28.84	36.08	38.22	**43.73**
ohsu_trec	13.87	18.04	19.46	**22.76**	9.76	15.00	18.48	**21.71**
trec_ap	20.50	24.27	28.74	**32.50**	11.56	15.89	19.07	**20.39**
trec_cr	14.35	18.51	17.56	**30.16**	9.53	12.75	12.45	**19.20**
trec_doe	18.67	21.58	20.14	**28.79**	11.23	12.40	13.27	**18.26**
trec_fbis	13.95	24.14	26.33	**32.99**	8.46	17.38	18.43	**24.60**
trec_fr	8.25	14.88	15.72	**20.49**	4.72	9.74	11.44	**15.75**
trec_ft	18.59	23.79	25.46	**33.29**	11.55	16.58	18.04	**22.32**
trec_la	16.86	18.22	22.73	**27.94**	10.76	12.75	14.49	**17.12**
trec_pt	12.19	20.65	29.55	**33.01**	6.91	14.73	17.87	**23.33**
trec_sjmn	19.03	14.78	20.49	**27.85**	12.03	11.14	13.73	**19.34**
trec_wsj	19.15	25.39	28.53	**35.88**	9.50	14.03	16.31	**20.50**
trec_zf	10.09	24.50	23.84	**31.39**	6.94	14.67	15.50	**19.81**
mean	17.30	22.64	25.03	**31.55**	10.91	15.63	17.48	**22.00**

Table A.26: Training times in seconds for lr_l2r_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
	100	0.0001	0.0002	0.0004	0.0008	0.0008	0.0008
	1000	0.0009	0.0013	0.0020	0.0040	0.0052	0.0053
	10000	0.0117	0.0159	0.0281	0.0509	0.0801	0.1044
	100000	0.1683	0.4092	0.7425	0.8779	0.9760	1.0540
	1000000	2.0355	6.2743	9.0974	9.4415	11.883	14.129
10	10	0.0002	0.0003	0.0010	0.0010	0.0010	0.0010
	100	0.0007	0.0011	0.0037	0.0085	0.0085	0.0086
	1000	0.0084	0.0134	0.0213	0.0530	0.0628	0.0628
	10000	0.1167	0.1676	0.3172	0.5188	0.8549	0.8525
	100000	1.7954	4.3427	6.8359	8.7140	11.208	12.095
	1000000	22.667	78.952	118.013	118.47	143.11	172.67
1000	10	0.0002	0.0003	0.0010	0.0010	0.0009	0.0009
	100	0.0065	0.0100	0.0328	0.0773	0.0773	0.0769
	1000	0.7587	1.2218	2.0641	5.0078	6.5650	6.5551
	10000	11.689	16.710	30.521	57.114	89.748	89.574
	100000	174.73	370.49	510.40	742.66	988.79	1081.3
	1000000	2189.1	4495.1	7999.8	9713.2	12722	NA
1000000	10	0.0004	0.0003	0.0010	0.0339	0.0009	0.0009
	100	0.0057	0.0100	0.0327	0.0773	0.0773	0.0769
	1000	0.7595	1.2238	1.8876	5.0495	6.6172	6.5731
	10000	111.56	154.99	278.18	530.67	848.82	849.07
	100000	NA	NA	NA	NA	NA	NA
	1000000	NA	NA	NA	NA	NA	NA

Table A.27: Training times in seconds for l2svm_l2r_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.0000	0.0001	0.0002	0.0002	0.0002	0.0002
	100	0.0001	0.0002	0.0005	0.0011	0.0011	0.0011
	1000	0.0004	0.0007	0.0028	0.0079	0.0104	0.0104
	10000	0.0038	0.0068	0.1135	0.0531	0.1095	0.1101
	100000	0.0478	0.1920	0.3682	0.6134	1.0333	1.3279
	1000000	0.8341	3.5295	5.2274	8.1350	10.373	16.083
10	10	0.0001	0.0003	0.0013	0.0013	0.0013	0.0013
	100	0.0004	0.0015	0.0053	0.0115	0.0115	0.0115
	1000	0.0034	0.0078	0.0312	0.0953	0.1141	0.1141
	10000	0.0404	0.0716	0.3067	0.5911	1.2975	1.3744
	100000	0.5661	1.8004	3.6789	5.7895	12.073	13.939
	1000000	9.9024	33.144	49.833	66.929	100.74	157.52
1000	10	0.0003	0.0003	0.0013	0.0013	0.0013	0.0013
	100	0.0032	0.0129	0.0478	0.1039	0.1037	0.1039
	1000	0.2965	0.7037	3.0223	9.6650	10.660	10.675
	10000	4.0634	6.8802	19.159	61.966	141.37	142.23
	100000	48.736	142.70	302.79	470.85	998.70	1398.0
	1000000	5675.4	8281.2	3578.4	5277.5	12685	NA
1000000	10	0.0001	0.0003	0.0013	0.0013	0.0013	0.0013
	100	0.0032	0.0129	0.0470	0.1039	0.1034	0.1034
	1000	0.2984	0.7063	2.9793	9.6680	10.563	10.626
	10000	37.778	60.488	169.58	570.82	1332.4	1327.6
	100000	NA	NA	NA	NA	NA	NA
	1000000	NA	NA	NA	NA	NA	NA

Table A.28: Training times in seconds for l2svm_l1r_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.0000	0.0000	0.0001	0.0001	0.0001	0.0001
	100	0.0001	0.0005	0.0003	0.0005	0.0006	0.0006
	1000	0.0002	0.0026	0.0077	0.0099	0.0106	0.0103
	10000	0.0016	0.0064	0.0857	0.1177	0.1315	0.1333
	100000	0.0110	0.0966	0.4699	0.7891	1.5672	0.9556
	1000000	0.2419	1.3368	5.6083	16.567	19.312	21.434
10	10	0.0001	0.0001	0.0004	0.0004	0.0004	0.0004
	100	0.0005	0.0032	0.0037	0.0056	0.0056	0.0057
	1000	0.0016	0.0295	0.1083	0.1081	0.1196	0.1199
	10000	0.0195	0.0810	1.2605	1.6538	1.5993	1.6816
	100000	0.2344	2.9972	27.844	17.851	19.906	22.160
	1000000	2.6495	12.147	72.906	261.68	281.47	277.37
1000	10	0.0003	0.0001	0.0004	0.0004	0.0004	0.0004
	100	0.0027	0.0300	0.0372	0.0490	0.0484	0.0463
	1000	0.1319	2.9088	11.645	11.153	11.341	11.388
	10000	2.3629	8.0287	158.97	207.07	203.13	202.94
	100000	9.8809	607.77	3135.7	2831.2	2643.5	2511.2
	1000000	413.15	1427.4004	NA	NA	NA	NA
1000000	10	0.0001	0.0001	0.0004	0.0004	0.0004	0.0004
	100	0.0026	0.0300	0.0369	0.0496	0.0483	0.0482
	1000	0.1325	2.8892	11.651	11.101	11.404	11.446
	10000	22.279	77.448	1564.9	2087.4	1962.6	1963.9
	100000	NA	NA	NA	NA	NA	NA
	1000000	NA	NA	NA	NA	NA	NA

Table A.29: Training times in seconds for lr_l1r_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.0000	0.0001	0.0001	0.0001	0.0001	0.0001
	100	0.0001	0.0002	0.0001	0.0002	0.0002	0.0002
	1000	0.0006	0.0010	0.0019	0.0053	0.0070	0.0069
	10000	0.0065	0.0124	0.1104	0.0414	0.0665	0.0896
	100000	0.0667	0.1451	0.3758	0.4872	0.6406	0.7311
	1000000	0.9081	2.8916	11.386	16.454	15.465	17.160
10	10	0.0001	0.0001	0.0003	0.0003	0.0003	0.0003
	100	0.0010	0.0012	0.0009	0.0017	0.0018	0.0017
	1000	0.0067	0.0132	0.0219	0.0616	0.0764	0.0764
	10000	0.0765	0.1518	0.3038	0.4541	0.7475	0.7326
	100000	0.9863	2.2069	5.0263	6.3274	8.2237	9.0659
	1000000	11.118	37.730	131.64	183.16	191.48	213.20
1000	10	0.0003	0.0002	0.0003	0.0003	0.0003	0.0003
	100	0.0080	0.0107	0.0071	0.0146	0.0146	0.0141
	1000	0.6058	1.2823	2.1988	5.9547	7.3667	7.2402
	10000	7.8575	16.184	33.175	46.843	86.436	86.163
	100000	102.46	278.76	636.04	756.73	914.16	1033.0
	1000000	1277.8	4043.6	13973	NA	NA	NA
1000000	10	0.0001	0.0001	0.0003	0.0003	0.0003	0.0003
	100	0.0098	0.0107	0.0071	0.0147	0.0147	0.0147
	1000	0.6064	1.2817	2.1301	5.9632	7.3422	7.2268
	10000	75.097	155.78	323.41	450.20	813.93	819.39
	100000	NA	NA	NA	NA	NA	NA
	1000000	NA	NA	NA	NA	NA	NA

Table A.30: Training times in seconds for mnb_c_jm_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.7960	0.9570	1.0570	1.0500	0.9890	1.1000
	100	0.7980	0.9110	1.1990	1.3600	1.2840	1.3700
	1000	0.8730	1.0230	1.3720	1.5950	1.5670	1.6630
	10000	0.8240	1.2430	1.7280	2.0370	2.5070	2.5170
	100000	1.4380	1.9760	3.7140	5.9010	7.0570	7.6850
	1000000	3.7440	9.1760	25.912	42.388	49.062	56.350
10	10	0.8940	0.9170	1.0590	1.0830	1.0080	1.0760
	100	0.8460	0.9730	1.1890	1.3260	1.3220	1.3900
	1000	0.8750	0.9980	1.3530	1.6610	1.5720	1.6780
	10000	0.8650	1.2190	1.8310	1.9720	2.5530	2.5900
	100000	1.6300	2.0190	3.8380	5.8080	7.1380	7.8690
	1000000	3.9580	9.3590	25.688	42.614	50.119	54.836
1000	10	0.8410	0.8690	1.0290	1.0370	1.1070	1.0450
	100	0.8620	0.9980	1.2330	1.2940	1.5510	1.3480
	1000	0.9300	1.1400	1.5260	1.9270	2.1850	2.1280
	10000	0.8400	1.2820	1.8400	2.4180	2.9430	2.8430
	100000	1.5380	2.2710	4.1040	6.2100	7.4630	8.2480
	1000000	4.1660	9.8050	27.356	45.013	52.388	58.735
1000000	10	0.8440	0.9190	1.0670	1.1420	1.1150	1.0600
	100	0.8940	0.9310	1.2280	1.3340	1.3180	1.3750
	1000	1.0450	1.1030	1.6000	2.0590	2.1420	2.2600
	10000	1.0920	1.6320	2.9020	4.1110	4.7820	4.7640
	100000	2.3750	4.8250	13.553	19.339	22.848	24.043
	1000000	11.346	39.257	137.97	252.56	284.44	297.33

Table A.31: Training times in seconds for mnb_ht_c_jm_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.7960	0.9570	1.0570	1.0500	0.9890	1.1000
	100	0.7980	0.9110	1.1990	1.3600	1.2840	1.3700
	1000	0.8730	1.0230	1.3720	1.5950	1.5670	1.6630
	10000	0.8240	1.2430	1.7280	2.0370	2.5070	2.5170
	100000	1.4380	1.9760	3.7140	5.9010	7.0570	7.6850
	1000000	3.7440	9.1760	25.912	42.388	49.062	56.350
10	10	0.8940	0.9170	1.0590	1.0830	1.0080	1.0760
	100	0.8460	0.9730	1.1890	1.3260	1.3220	1.3900
	1000	0.8750	0.9980	1.3530	1.6610	1.5720	1.6780
	10000	0.8650	1.2190	1.8310	1.9720	2.5530	2.5900
	100000	1.6300	2.0190	3.8380	5.8080	7.1380	7.8690
	1000000	3.9580	9.3590	25.688	42.614	50.119	54.836
1000	10	0.8410	0.8690	1.0290	1.0370	1.1070	1.0450
	100	0.8620	0.9980	1.2330	1.2940	1.5510	1.3480
	1000	0.9300	1.1400	1.5260	1.9270	2.1850	2.1280
	10000	0.8400	1.2820	1.8400	2.4180	2.9430	2.8430
	100000	1.5380	2.2710	4.1040	6.2100	7.4630	8.2480
	1000000	4.1660	9.8050	27.356	45.013	52.388	58.735
1000000	10	0.8440	0.9190	1.0670	1.1420	1.1150	1.0600
	100	0.8940	0.9310	1.2280	1.3340	1.3180	1.3750
	1000	1.0450	1.1030	1.6000	2.0590	2.1420	2.2600
	10000	1.0920	1.6320	2.9020	4.1110	4.7820	4.7640
	100000	2.3750	4.8250	13.553	19.339	22.848	24.043
	1000000	11.346	39.257	137.97	252.56	284.44	297.33

Table A.32: Training times in seconds for tdm_c_jm_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.8710	0.9900	1.1090	1.1300	1.1230	1.1400
	100	0.8790	1.1550	1.2540	1.5090	1.3910	1.4310
	1000	0.9560	1.1320	1.7020	2.2030	2.3380	2.4050
	10000	1.2020	1.8080	3.3090	4.7660	5.2520	5.3160
	100000	2.8890	6.0580	16.262	23.959	30.482	29.778
	1000000	12.928	NA	NA	NA	NA	NA
10	10	0.8990	1.0770	1.1380	1.1240	1.0780	1.1280
	100	0.8940	1.0350	1.2340	1.4130	1.4390	1.4930
	1000	0.9390	1.2580	1.7520	2.2090	2.1880	2.2960
	10000	1.1340	1.7800	3.3070	4.8550	5.3250	5.3390
	100000	2.9950	6.2540	14.819	26.155	30.157	29.596
	1000000	11.686	NA	NA	NA	NA	NA
1000	10	0.9070	0.9860	1.1080	1.0900	1.1890	1.1410
	100	0.9640	1.0790	1.2250	1.4590	1.4410	1.4470
	1000	1.0090	1.2650	1.7410	2.3730	2.3080	2.3230
	10000	1.2440	2.0500	3.3750	4.9760	5.6250	5.6510
	100000	2.8590	6.2570	15.028	24.879	30.490	30.236
	1000000	12.949	NA	NA	NA	NA	NA
1000000	10	0.9310	0.9450	1.1510	1.1980	1.1460	1.1230
	100	1.0110	1.0710	1.2790	1.4270	1.4470	1.4390
	1000	0.9990	1.2510	1.7090	2.5010	2.5760	2.3480
	10000	1.4370	2.2830	4.3430	6.1070	7.0820	7.1310
	100000	3.7330	9.1130	23.223	39.236	42.128	43.396
	1000000	21.207	NA	NA	NA	NA	NA

Table A.33: Training times in seconds for tdm_ht_c_jm_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.9170	1.0090	1.1980	1.2200	1.1280	1.1960
	100	0.9260	1.1380	1.4110	1.5050	1.5130	1.4470
	1000	0.9450	1.2260	1.5930	2.1850	2.3790	2.1340
	10000	1.1440	1.7140	3.3050	4.9150	5.1990	5.1930
	100000	2.8150	5.6730	14.9470	23.4950	NA	NA
	1000000	12.9140	NA	NA	NA	NA	NA
10	10	0.9280	1.0530	1.1830	1.1660	1.2070	1.1500
	100	0.9980	1.1130	1.3230	1.4630	1.4720	1.5410
	1000	1.0040	1.2290	1.6500	2.0790	2.4190	2.5340
	10000	1.1520	1.7460	3.4040	4.7840	5.2020	5.2730
	100000	2.7420	5.7550	14.9800	23.5480	NA	NA
	1000000	12.8880	NA	NA	NA	NA	NA
1000	10	0.9290	0.9800	1.1970	1.2270	1.2490	1.1910
	100	1.0780	1.1240	1.3380	1.4720	1.4470	1.4830
	1000	1.0600	1.1980	1.7890	2.4960	2.2760	2.2940
	10000	1.1910	1.9680	3.4280	5.2450	5.4840	5.4810
	100000	2.8110	6.2980	15.3180	23.9250	NA	NA
	1000000	11.7400	NA	NA	NA	NA	NA
1000000	10	1.0010	1.0590	1.1250	1.1250	1.1090	1.1940
	100	1.0120	1.0660	1.3480	1.5150	1.5530	1.5380
	1000	1.1350	1.3500	1.7670	2.3450	2.2720	2.2760
	10000	1.4920	2.4200	4.4770	6.2660	6.9170	7.2160
	100000	3.6570	8.6960	NA	NA	NA	NA
	1000000	NA	NA	NA	NA	NA	NA

Table A.34: Mean test times in milliseconds for lr_l2r_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.0201	0.0347	0.0694	0.0705	0.0698	0.0709
	100	0.0218	0.0475	0.1031	0.1439	0.1461	0.1460
	1000	0.0262	0.0474	0.1189	0.1862	0.1965	0.1927
	10000	0.0238	0.0492	0.1239	0.2089	0.2235	0.2190
	100000	0.0268	0.0552	0.1334	0.2069	0.2306	0.2356
	1000000	0.0536	0.0881	0.1744	0.2772	0.3131	0.3229
10	10	0.0206	0.0384	0.0707	0.0717	0.0732	0.0708
	100	0.0248	0.0434	0.1047	0.1471	0.1484	0.1487
	1000	0.0242	0.0516	0.1207	0.1878	0.1950	0.1977
	10000	0.0250	0.0491	0.1249	0.2019	0.2250	0.2267
	100000	0.0297	0.0536	0.1360	0.2082	0.2356	0.2354
	1000000	0.0560	0.0835	0.1710	0.2711	0.3019	0.3177
1000	10	0.0209	0.0350	0.0716	0.0714	0.0716	0.0705
	100	0.0279	0.0491	0.1106	0.1526	0.1543	0.1586
	1000	0.0740	0.1033	0.1903	0.2766	0.2793	0.2841
	10000	0.0871	0.1163	0.2052	0.2988	0.3215	0.3230
	100000	0.0742	0.1091	0.2062	0.2934	0.3189	0.3196
	1000000	0.0983	0.1301	0.2313	0.3322	0.3711	NA
1000000	10	0.0205	0.0349	0.0700	0.0756	0.0701	0.0702
	100	0.0274	0.0486	0.1111	0.1529	0.1545	0.1538
	1000	0.0742	0.1029	0.1915	0.2718	0.2799	0.2804
	10000	0.6095	0.6746	0.8917	1.1102	1.1576	1.1520
	100000	NA	NA	NA	NA	NA	NA
	1000000	NA	NA	NA	NA	NA	NA

Table A.35: Mean test times in milliseconds for l2svm_l2r_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.0225	0.0354	0.0698	0.0718	0.0695	0.0733
	100	0.0240	0.0431	0.1046	0.1460	0.1462	0.1484
	1000	0.0239	0.0496	0.1207	0.1868	0.1966	0.1945
	10000	0.0239	0.0491	0.1252	0.2051	0.2195	0.2248
	100000	0.0250	0.0521	0.1304	0.2072	0.2313	0.2343
	1000000	0.0532	0.0836	0.1798	0.2707	0.3067	0.3162
10	10	0.0203	0.0352	0.0708	0.0701	0.0711	0.0704
	100	0.0230	0.0437	0.1052	0.1452	0.1467	0.1451
	1000	0.0247	0.0481	0.1203	0.1867	0.1953	0.1956
	10000	0.0247	0.0494	0.1242	0.2050	0.2226	0.2267
	100000	0.0271	0.0575	0.1325	0.2063	0.2318	0.2360
	1000000	0.0546	0.0816	0.1719	0.2743	0.3054	0.3198
1000	10	0.0205	0.0347	0.0729	0.0724	0.0702	0.0702
	100	0.0276	0.0485	0.1114	0.1535	0.1529	0.1527
	1000	0.0746	0.1021	0.1963	0.2716	0.2825	0.2786
	10000	0.0870	0.1162	0.2079	0.2986	0.3261	0.3196
	100000	0.0757	0.1099	0.1952	0.2893	0.3181	0.3222
	1000000	0.1002	0.1325	0.2317	0.3354	0.3678	NA
1000000	10	0.0204	0.0351	0.0722	0.0706	0.0707	0.0709
	100	0.0270	0.0484	0.1106	0.1541	0.1533	0.1527
	1000	0.0749	0.1024	0.1938	0.2708	0.2782	0.2826
	10000	0.6079	0.6730	0.8910	1.1057	1.1551	1.1572
	100000	NA	NA	NA	NA	NA	NA
	1000000	NA	NA	NA	NA	NA	NA

Table A.36: Mean test times in milliseconds for l2svm_l1r_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.0204	0.0351	0.0704	0.0724	0.0701	0.0708
	100	0.0246	0.0435	0.1053	0.1450	0.1441	0.1457
	1000	0.0238	0.0470	0.1202	0.1844	0.1946	0.1928
	10000	0.0248	0.0495	0.1268	0.1994	0.2243	0.2254
	100000	0.0257	0.0529	0.1286	0.2109	0.2322	0.2360
	1000000	0.0517	0.0815	0.1749	0.2831	0.3148	0.3291
10	10	0.0213	0.0377	0.0707	0.0712	0.0730	0.0705
	100	0.0229	0.0440	0.1036	0.1449	0.1448	0.1473
	1000	0.0254	0.0481	0.1197	0.1888	0.1937	0.1959
	10000	0.0246	0.0496	0.1280	0.2059	0.2251	0.2230
	100000	0.0253	0.0560	0.1331	0.2085	0.2366	0.2339
	1000000	0.0526	0.0831	0.1726	0.2811	0.3077	0.3261
1000	10	0.0208	0.0353	0.0712	0.0722	0.0705	0.0701
	100	0.0275	0.0490	0.1120	0.1520	0.1546	0.1566
	1000	0.0752	0.1023	0.1891	0.2744	0.2833	0.2828
	10000	0.0891	0.1189	0.2068	0.2989	0.3239	0.3214
	100000	0.0760	0.1102	0.1983	0.2938	0.3284	0.3209
	1000000	0.1019	0.1322	NA	NA	NA	NA
1000000	10	0.0210	0.0348	0.0735	0.0733	0.0711	0.0716
	100	0.0274	0.0486	0.1106	0.1526	0.1535	0.1519
	1000	0.0769	0.1028	0.1891	0.2677	0.2841	0.2814
	10000	0.6142	0.6750	0.8909	1.1085	1.1605	1.1565
	100000	NA	NA	NA	NA	NA	NA
	1000000	NA	NA	NA	NA	NA	NA

Table A.37: Mean test times in milliseconds for lr_l1r_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.0204	0.0389	0.0699	0.0728	0.0703	0.0707
	100	0.0256	0.0442	0.1051	0.1482	0.1464	0.1449
	1000	0.0268	0.0474	0.1187	0.1863	0.1918	0.1940
	10000	0.0259	0.0486	0.1264	0.2031	0.2238	0.2213
	100000	0.0275	0.0527	0.1299	0.2032	0.2323	0.2371
	1000000	0.0528	0.0829	0.1764	0.2795	0.3139	0.3277
10	10	0.0208	0.0378	0.0700	0.0740	0.0738	0.0703
	100	0.0227	0.0434	0.1052	0.1460	0.1460	0.1467
	1000	0.0251	0.0477	0.1212	0.1870	0.1959	0.1944
	10000	0.0249	0.0488	0.1261	0.2016	0.2271	0.2278
	100000	0.0251	0.0568	0.1349	0.2069	0.2286	0.2364
	1000000	0.0524	0.0822	0.1728	0.2670	0.3068	0.3220
1000	10	0.0211	0.0353	0.0702	0.0710	0.0709	0.0705
	100	0.0270	0.0496	0.1137	0.1555	0.1522	0.1545
	1000	0.0744	0.1025	0.1888	0.2740	0.2785	0.2882
	10000	0.0872	0.1177	0.2057	0.2986	0.3195	0.3219
	100000	0.0749	0.1097	0.1995	0.2870	0.3202	0.3214
	1000000	0.0981	0.1299	0.2320	NA	NA	NA
1000000	10	0.0205	0.0345	0.0736	0.0707	0.0740	0.0744
	100	0.0270	0.0489	0.1113	0.1537	0.1529	0.1557
	1000	0.0740	0.1038	0.1916	0.2707	0.2829	0.2828
	10000	0.6122	0.6733	0.8894	1.1054	1.1564	1.1571
	100000	NA	NA	NA	NA	NA	NA
	1000000	NA	NA	NA	NA	NA	NA

Table A.38: Mean test times in milliseconds for mnb_c_jm_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.0269	0.0309	0.0320	0.0259	0.0280	0.0274
	100	0.0273	0.0265	0.0330	0.0430	0.0457	0.0411
	1000	0.0264	0.0285	0.0384	0.0362	0.0402	0.0394
	10000	0.0278	0.0264	0.0380	0.0405	0.0433	0.0427
	100000	0.0322	0.0332	0.0433	0.0377	0.0459	0.0493
	1000000	0.0264	0.0329	0.0373	0.0409	0.0479	0.0493
10	10	0.0253	0.0249	0.0287	0.0281	0.0282	0.0281
	100	0.0251	0.0302	0.0391	0.0395	0.0399	0.0398
	1000	0.0289	0.0292	0.0426	0.0407	0.0461	0.0444
	10000	0.0317	0.0389	0.0479	0.0446	0.0511	0.0528
	100000	0.0308	0.0381	0.0426	0.0477	0.0508	0.0524
	1000000	0.0274	0.0326	0.0476	0.0542	0.0633	0.0643
1000	10	0.0295	0.0252	0.0286	0.0277	0.0307	0.0304
	100	0.0372	0.0527	0.0646	0.0712	0.0772	0.0722
	1000	0.0958	0.2070	0.2736	0.2753	0.2810	0.2769
	10000	0.1072	0.1911	0.2768	0.2899	0.2921	0.2974
	100000	0.1586	0.2829	0.3603	0.3743	0.3796	0.3706
	1000000	0.2287	0.4620	0.7693	0.8320	0.8507	0.9116
1000000	10	0.0251	0.0293	0.0314	0.0326	0.0373	0.0278
	100	0.0304	0.0518	0.0698	0.0736	0.0772	0.0765
	1000	0.1004	0.2075	0.2731	0.2732	0.2786	0.2742
	10000	0.7595	1.7969	2.4888	2.4923	2.4271	2.4506
	100000	6.9852	21.368	28.749	28.485	28.168	28.752
	1000000	134.01	350.59	523.45	539.47	533.50	530.29

Table A.39: Mean test times in milliseconds for mnb_ht_c_jm_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.0262	0.0310	0.0265	0.0271	0.0275	0.0267
	100	0.0258	0.0256	0.0390	0.0380	0.0413	0.0397
	1000	0.0265	0.0249	0.0386	0.0389	0.0435	0.0459
	10000	0.0295	0.0256	0.0436	0.0385	0.0461	0.0460
	100000	0.0362	0.0252	0.0450	0.0392	0.0430	0.0433
	1000000	0.0360	0.0364	0.0361	0.0399	0.0464	0.0483
10	10	0.0279	0.0315	0.0368	0.0376	0.0367	0.0409
	100	0.0257	0.0376	0.0495	0.0620	0.0600	0.0608
	1000	0.0287	0.0336	0.0602	0.0701	0.0774	0.0791
	10000	0.0311	0.0429	0.0615	0.0851	0.0857	0.0881
	100000	0.0253	0.0439	0.0638	0.0712	0.0840	0.0846
	1000000	0.0332	0.0427	0.0538	0.0711	0.0837	0.0883
1000	10	0.0238	0.0277	0.0370	0.0373	0.0360	0.0365
	100	0.0456	0.0843	0.2114	0.3210	0.3108	0.3243
	1000	0.2150	0.7310	2.7171	4.5218	4.7183	4.7247
	10000	0.2109	0.7473	2.8806	5.0168	5.4344	5.4808
	100000	0.2050	0.7675	2.9801	4.9849	5.8185	5.8739
	1000000	0.1993	0.7965	3.1769	5.2456	5.8990	6.1975
1000000	10	0.0277	0.0282	0.0420	0.0389	0.0359	0.0362
	100	0.0446	0.0926	0.2184	0.3121	0.3164	0.3191
	1000	0.2094	0.7238	2.7596	4.5496	4.7158	4.6876
	10000	2.0148	8.4291	29.615	48.409	54.606	53.019
	100000	19.764	78.220	261.34	440.62	468.37	490.07
	1000000	198.98	NA	NA	NA	NA	NA

Table A.40: Mean test times in milliseconds for tdm_c_jm_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.0274	0.0332	0.0479	0.0488	0.0483	0.0498
	100	0.0395	0.0602	0.0962	0.1059	0.0975	0.1035
	1000	0.1741	0.3373	0.4816	0.4866	0.4907	0.4951
	10000	1.4231	3.2992	4.6781	4.6911	4.5519	4.6349
	100000	19.237	46.185	62.657	64.243	63.606	62.989
	1000000	267.21	NA	NA	NA	NA	NA
10	10	0.0251	0.0281	0.0407	0.0403	0.0413	0.0417
	100	0.0391	0.0635	0.0996	0.1125	0.0961	0.0947
	1000	0.1716	0.3294	0.4888	0.4899	0.4882	0.5132
	10000	1.4279	3.3132	4.7438	4.6983	4.6051	4.6182
	100000	18.831	46.034	61.461	63.622	63.287	63.418
	1000000	265.42	NA	NA	NA	NA	NA
1000	10	0.0250	0.0282	0.0437	0.0414	0.0529	0.0535
	100	0.0460	0.0979	0.1334	0.1332	0.1341	0.1342
	1000	0.2843	0.6062	0.8199	0.8345	0.8311	0.8248
	10000	1.5279	3.5055	5.0200	5.0526	4.9760	5.0519
	100000	19.982	47.736	63.311	64.237	65.530	65.136
	1000000	279.25	NA	NA	NA	NA	NA
1000000	10	0.0293	0.0279	0.0510	0.0410	0.0449	0.0404
	100	0.0465	0.0979	0.1464	0.1316	0.1321	0.1402
	1000	0.2749	0.6172	0.8331	0.8282	0.8326	0.8359
	10000	2.5532	6.4168	9.0145	9.1412	9.2989	9.1650
	100000	36.372	98.812	128.19	133.36	129.98	130.20
	1000000	582.13	NA	NA	NA	NA	NA

Table A.41: Mean test times in milliseconds for tdm_ht_c_jm_tXiX

Labelsets	Documents	Features					
		10	100	1000	10000	100000	1000000
1	10	0.0293	0.0328	0.0545	0.0454	0.0450	0.0454
	100	0.0421	0.1304	0.2424	0.4188	0.3597	0.3582
	1000	0.2162	0.7855	2.9544	4.8711	5.1273	5.1460
	10000	1.8547	9.3212	31.352	52.196	56.691	65.740
	100000	21.398	109.51	346.56	562.37	NA	NA
	1000000	250.39	NA	NA	NA	NA	NA
10	10	0.0284	0.0408	0.0538	0.0539	0.0526	0.0842
	100	0.0595	0.1107	0.2731	0.4045	0.4040	0.4030
	1000	0.2457	0.7934	2.9861	4.8913	5.1742	5.2528
	10000	1.8875	9.2534	31.093	51.808	56.994	57.804
	100000	21.260	110.42	355.04	585.36	NA	NA
	1000000	258.57	NA	NA	NA	NA	NA
1000	10	0.0284	0.0425	0.0638	0.0619	0.0796	0.0605
	100	0.0748	0.1927	0.5023	0.7102	0.7177	0.7253
	1000	0.7167	2.2339	6.4641	10.685	10.740	10.781
	10000	2.3641	10.905	34.880	57.516	62.991	63.478
	100000	22.435	108.343	372.44	588.89	NA	NA
	1000000	262.46	NA	NA	NA	NA	NA
1000000	10	0.0359	0.0477	0.0532	0.0533	0.0549	0.0532
	100	0.0749	0.2171	0.4952	0.7169	0.7345	0.7518
	1000	0.7072	2.2216	6.5075	10.124	10.834	10.797
	10000	6.5784	23.557	66.791	106.85	116.36	126.72
	100000	65.579	256.16	NA	NA	NA	NA
	1000000	NA	NA	NA	NA	NA	NA

Appendix B

Kaggle LSHTC4 Winning Solution

The appended document describes the winning solution to Kaggle LSHTC4[1] competition organized in 2014, that had 119 participating teams. The solution was based on the methods presented in the thesis, including the sparse inference, the modified and extended MNB models, the TDM model, and the random search development framework. Some additional techniques such as model-based feedback, label thresholding, and ensemble learning were developed for the competition, that fall outside the scope of the thesis. The developed ensemble learning is a natural continuation of the ideas presented in the thesis. It combines an ensemble of sparse generative model base-classifiers using a mixture model, where the weight of each component is dynamically predicted using a large number of meta-features, and the outputs of the base-classifiers are restricted to predicting a single most likely output for each input. The LSHTC ensemble solution was later extended for the Kaggle WISE2014[2] competition, where it came second out of 120 competing teams.

[1] http://www.kaggle.com/c/lshtc/
[2] http://www.kaggle.com/c/wise-2014/

Kaggle LSHTC4 Winning Solution

Antti Puurula[1], Jesse Read[2], and Albert Bifet[3]

[1] Department of Computer Science, The University of Waikato, Private Bag 3105, Hamilton 3240, New Zealand
[2] Department of Information and Computer Science, Aalto University, FI-00076 Aalto, Espoo, Finland
[3] Huawei Noah's Ark Lab, Hong Kong Science Park, Shatin, Hong Kong, China

1 Overview

Our winning submission to the 2014 Kaggle competition for Large Scale Hierarchical Text Classification (LSHTC) consists mostly of an ensemble of sparse generative models extending Multinomial Naive Bayes. The base-classifiers consist of hierarchically smoothed models combining document, label, and hierarchy level Multinomials, with feature pre-processing using variants of TF-IDF and BM25. Additional diversification is introduced by different types of folds and random search optimization for different measures. The ensemble algorithm optimizes macroFscore by predicting the documents for each label, instead of the usual prediction of labels per document. Scores for documents are predicted by weighted voting of base-classifier outputs with a variant of Feature-Weighted Linear Stacking. The number of documents per label is chosen using label priors and thresholding of vote scores.

This document describes the models and software used to build our solution. Reproducing the results for our solution can be done by running the scripts included in the Kaggle package[4]. A package omitting precomputed result files is also distributed[5]. All code is open source, released under GNU GPL 2.0, and GPL 3.0 for Weka and Meka dependencies.

2 Data Segmentation

Source files: MAKE_FILES, nfold_sample_corpus.py, fast_sample_corpus.py, shuffle_data.py, count_labelsets2.py

Training data segmentation is done by the script MAKE_FILES, included in the code package. This segments the original training dataset train.txt by random sampling into portions for base-classifier training and for ensemble training.

[4] https://kaggle2.blob.core.windows.net/competitions/kaggle/3634/media/LSHTC4_winner_solution.zip
[5] https://kaggle2.blob.core.windows.net/competitions/kaggle/3634/media/LSHTC4_winner_solution_omit_resultsfiles.zip

II

2,341,782 documents are segmented for the former portion and 23,654 documents for the latter. The base-classifier training dataset dry_train.txt is further sampled into 10 different folds, each with a 1000 document held-out portion dry_dev.txt for parameter optimization. Folds 0-2 have exclusive and different sampled sets for dry_dev.txt. Folds 3-5 sample dry_train.txt randomly into 3 exlusive training subsets, with a shared optimization portion. Folds 6-9 segment dry_train.txt in the original data order into 4 exclusive training subsets, with a shared optimization portion. For all folds, the training datasets are further shuffled to improve the online pruning of parameters in training.

3 Base-classifiers

Source files: SGM-45l/, SGM-45l_je/, Metaopt2.py, Make_templates.py, results/, RUN_DEVS, RUN_EVALS, meka.jar

The base-classifiers consist mostly of sparse generative model extensions of Multinomial Naive Bayes (MNB). These extend MNB by introducing constrained finite mixtures at the document and hierarchy level nodes, and performing inference from the Multinomial node-conditional models using hierarchical smoothing, and kernel densities in case of document-conditional nodes. A special case is models using BM25 for kernel densities and no hierarchical smoothing. The models are stored in a sparse precomputed format, and inference using inverted indices is used to reduce the inference complexity according to the sparsity of the model. The constrained mixture modeling and sparse inference makes the models as scalable for text modeling as Naive Bayes and KNN, but with higher modelling accuracy. A detailed description of basic models of this type are given in [1, 2]. Since the LSHTC models can contain up to 100 million parameters for word counts, the models are provided as configuration files in the package. Estimating the models from training data takes negligible time more compared to reading saved model files.

A development version of the SGMWeka toolkit[6] was customized to implement the models. The customized version is included as the Java source directory SGM-45l, and the program SGM_Tests.java used for training and testing the models can be compiled without external dependencies. The documentation for SGMWeka version 1.4.4[7] is accurate, but the development version contains additional functionalities. A modified version is in the directory SGM-45l_je. This includes the Meka toolkit[8] for doing multi-label decomposition used by one of the base-classifiers.

[6] http://sourceforge.net/projects/sgmweka/
[7] http://sourceforge.net/p/sgmweka/wiki/SGMWeka%20Documentation%20v.1.4.4/
[8] http://meka.sourceforge.net/

III

The script Metaopt2.py optimizes a base-classifier on a development set according to a chosen performance measure, by iteratively estimating the classifier and classifying the development data portion. The script RUN_DEVS runs the development and compresses the log files. The configuration files for Metaopt2.py describes all the parameters provided to a SGM_Tests call, as well as the optimization measure to extract from the last line of the SGM_Tests log file. Metaopt2.py performs a Gaussian Random Search [3] for the chosen parameters, constrained and transformed according to the configuration file. The directories results_* contain the first and last parameter configuration file for each base-classifier type, after a 40x8 iteration random search. Some classifiers were constructed by copying the parameters for similar folds (3,4,5), and some used manually chosen parameter configurations. These classifiers have the final iteration parameter file wikip_large_X_params.txt_39_0, but not the initial file wikip_large_X_params.txt. The script Make_templates.py makes the parameter template files as specified in the global variable "configs".

The template files describe the model by suffixing the file name with modifications. For example, "mnb_mafs2_s8_lp_u_jm2_bm18ti_pct0_ps5_thr16.template" modifies a Multinomial Naive Bayes by optimizing the parameters for a modified version of macro-Fscore (_mafs2), uses data fold 8 (_s8), the Label Powerset method for multi-label classification (_lp), smoothing by a uniform background distribution (_u), a BM25 variant for feature weighting (_bm18ti), uses a safe pruning of pre-computed parameters (_pct0), constrains the scaling of label prior (_ps5) and uses 16 threads for parallel classification.

Some of the modifications have little influence on the results, such as _thr16 that instructs SGM_Tests to use 16 threads. More detailed explanations of the important modifications are given in the following sections. A total of 54 base-classifiers are used in the ensemble, selected down to 42 base-classifiers by model selection. Table 1 shows the base-classifiers sorted according to comb_dev.txt macro-averaged Fscore. It should be noted that the parameter ranges for some of the modifications were adjusted during the competition, and the parameter ranges in the individual template files can differ from those in Make_templates.py.

The word count vectors for LSHTC were preprocessed by the organizers to remove common words, stopwords and short words, as can be seen from looking at the distributions of words in the vectors. This causes problems for some models such as Multinomial models of text, that assume word vectors to distribute normally. Feature transforms and weighting can be used to correct this. Feature weighting is done by each base-classifier separately, using variants of TF-IDF and BM25. All models use 1-3 parameters to optimize the feature weighting on the dry_dev.txt portion of the fold. A variant of BM25 that proved most successful has the suffix "_bm18ti". As seen in TFIDF.java, this combines the term count normalization of BM25 with the parameterized length normalization and

IV

id	excluded	parameter configuration	maFscore
7		mafs3_s1_uc1_jm3_bm18ti_pci7_pct0_psX_fb_iw2	0.4155
9		mafs3_s1_uc1_jm3_bm18ti_pci7_pct0_psX_iw2	0.4082
11	X	mafs3_s2_uc1_jm2_bm18tid_pci7_pct0_ps8_iw1	0.3993
13		mafs3_s3_kd_u_jm3_kdp5_bm18ti_pct0_ps7_iw2	0.3982
17		mafs3_s4_kd_u_jm3_kdp5_bm18ti_pct0_ps7_iw2	0.3982
8		mafs3_s1_uc1_jm3_bm18ti_pci7_pct0_psX_iw1	0.3866
10	X	mafs3_s2_u_lp_jm2_bm18tib_pct0_ps7_iw0	0.3795
20		mafs3_s5_kd_u_jm3_kdp5_bm18ti_pct0_ps7_iw0	0.3771
12		mafs3_s3_kd_u_jm3_kdp5_bm18ti_pct0_ps7_iw0	0.3763
6		mafs3_s1_u_jm3_bm18ti_pct0_ps7_iw0	0.3689
16		mafs3_s4_kd_u_jm3_kdp5_bm18ti_pct0_ps7_iw0	0.3615
14		mafs3_s3_kd_uc1_jm2_kdp5_bm18tid_pct0_ps8_iw1	0.3466
5		mafs3_s0_kd_nobo_bm25c2_mi2_ps2_iw0	0.3380
19		mafs3_s4_u_jm2_bm18tib_pci6_pct0_ps7_cs0_iw2	0.3346
18	X	mafs3_s4_u_jm2_bm18tib_mc0_pci6_pct0_ps7_cs0_iw0	0.3114
21		mafs3_s5_u_jm2_bm18tib_mc0_pci6_pct0_ps7_cs0_iw0	0.3091
15		mafs3_s3_u_jm2_bm18tib_mc0_pci6_pct0_ps7_cs0_iw0	0.3082
33		mafs_s2_lp_u_jm5_pd2_bm16ti_mc0_pct0_ps0	0.2860
50		mjac_s2_kd_nobo_bm25c2_mc0_mlc0_ps2_lt5_mr0_tk1	0.2856
0		mafs2_s2_lp_u_jm2_bm18tib_mc0_pct0_ps5	0.2815
28	X	mafs_s1_kd_nobo_bm25c2_mc0_mlc0_ps2_lt5_mr0_tk2	0.2805
32		mafs_s2_lp_u_jm4_bm20ti_mc0_pct0_ps2	0.2723
27		mafs_s0_lp_u_jm2_bm18tic_fb3_mc0_pct0_ps6	0.2686
44	X	mjac_s0_lp_u_jm2_pd2_tXiX3_fb2_mc0_pci1_pct0_ps0	0.2678
52		ndcg5b_s4_kd_u_jm2_kdp5_bm18tib_mc0_pci0_pct0_mlc0_ps6_tk0	0.2659
51		ndcg5b_s3_kd_u_jm2_kdp5_bm18tib_mc0_pci0_pct0_mlc0_ps6_tk0	0.2650
53		ndcg5b_s5_kd_u_jm2_kdp5_bm18tib_mc0_pci0_pct0_mlc0_ps6_tk0	0.2643
30		mafs_s1_lp_u_jm6_tiX5_mc0_pct0_ps0	0.2618
29	X	mafs_s1_lp_u_jm4_pd2_tXiX2_fb2_mc0_pct0_ps0	0.2612
45	X	mjac_s0_lp_u_jm2_tiX3_mc0_pct0_ps0	0.2592
31	X	mafs_s2_lp_u_jm4_bm18ti_mc0_pct0_ps2	0.2567
23		mafs3_s7_kd_uc1_jm2_kdp5_bm18tid_mc0_pci1_pct0_ps8_iw1_ch80	0.2550
42		mifs_s2_lp_u_jm2_bm18tib_fb3_mc0_pct0_ps5	0.2530
46	X	mjac_s0_lp_u_jm4_bm15ti_mc0_pct0_ps0	0.2489
22		mafs3_s6_kd_uc1_jm2_kdp5_bm18tid_mc0_pci1_pct0_ps8_iw1_ch80	0.2444
35		mafs_s4_kd_u_jm3_kdp5_tXiX2_mc0_pci0_pct0_mlc0_ps5_lt5_mr0_tk2	0.2441
34		mafs_s3_kd_u_jm3_kdp5_tXiX2_mc0_pci0_pct0_mlc0_ps5_lt5_mr0_tk2	0.2422
36		mafs_s5_kd_u_jm3_kdp5_tXiX2_mc0_pci0_pct0_mlc0_ps5_lt5_mr0_tk2	0.2421
49		mjac_s1_u_jm3_tiX1_mc0_pci1_pct0_mlc0_ps1_lt1_mr0	0.2410
48	X	mjac_s1_u_jm2_tiX1_mc0_pct0_mlc0_ps2_lt2_mr0_tk0	0.2395
24		mafs3_s8_kd_uc1_jm2_kdp5_bm18tid_mc0_pci1_pct0_ps8_iw1_ch80	0.2335
26	X	mafs_s0_lp_u_jm2_bm18tib_mc0_pct0_ps5	0.2245
41		mifs_s1_lp_u_jm2_bm18tib_mc0_pct0_ps5	0.2232
25		mafs3_s9_kd_uc1_jm2_kdp5_bm18tid_mc0_pci1_pct0_ps8_iw1_ch80	0.2108
47		mjac_s0_u_jm3_bm18ti_pct0_ps5_je	0.2040
43		mjac_s0_lp_bm25c1_mc0_mlc0_ps3	0.1924
38		mafs_s7_kd_u_jm3_kdp1_bm18ti_mc0_pci1_pct0_ps5_lt5_mr1_tk2_ch80	0.1787
39		mafs_s8_kd_u_jm3_kdp1_bm18ti_mc0_pci1_pct0_ps5_lt5_mr1_tk2_ch80	0.1632
2		mafs2_s7_lp_u_jm2_bm18ti_pct0_ps5	0.1554
1	X	mafs2_s6_lp_u_jm2_bm18ti_pct0_ps5	0.1529
3		mafs2_s8_lp_u_jm2_bm18ti_pct0_ps5	0.1513
37		mafs_s6_kd_u_jm3_kdp1_bm18ti_mc0_pci1_pct0_ps5_lt5_mr1_tk2_ch80	0.1469
40		mafs_s9_kd_u_jm3_kdp1_bm18ti_mc0_pci1_pct0_ps5_lt5_mr1_tk2_ch80	0.1452
4		mafs2_s9_lp_u_jm2_bm18ti_pct0_ps5	0.1357

Table 1. Base-classifiers sorted in the order of comb_dev.txt macro-averaged Fscore, computed over the labels occurring in the set. Excluded models were removed by model selection from the ensemble.

V

idf weighting from TF-IDF that has been used earlier [3].

The Multinomials use hierarchical smoothing with a uniform background distribution [2]. The variant "_uc1" uses a uniform distribution interpolated with a collection model, improving the accuracy by a small amount. All models use Jelinek-Mercer "_jmX" for smoothing label and hierarchy level Multinomials, and Dirichlet Prior smoothing "_kdpX" for smoothing kernel density document models. The feature selection done by the organizers cause very unusual smoothing parameter configurations to be optimal. With Jelinek-Mercer values less than a heavy amount such as 0.98 become rapidly worse, with some models using a smoothing coefficient of 0.999.

Parameter pruning is chosen by the modifiers "_mcX", "_pciX", "_pctX", "_mlcX". "_mcX" prunes word features based on their frequency. "_pciX" selects on-line pruning of conditional parameters, "_pctX" performs mostly safe pruning of precomputed conditional parameters, "_mlcX" prunes labels based on their frequency.

One special classifier is the variant using the modifer "_je". This requires a development version of the Meka toolkit and the other files in the directory /SGM-45l_je. This model does classification with label powersets decomposed into meta-labels, and transforms the meta-labels back into labelsets after classification. The labelset decompositions are stored in a precomputed file loaded by the modified version of SGM_Tests.

Kernel densities are selected with the modifier "_kd", passing -kernel_densities to SGM_Tests. This constructs document-conditional models, and computes label-conditional probabilities using the document-conditionals as kernel densities [2]. The modifiers "_csX" load the LSHTC4 label hierarchy, and use random parent nodes to smooth the label-conditional Multinomials. The Label Powerset method for mapping a multi-label problem into a multi-class problem is done by the modifier "_lp", passing -label_powerset to SGM_Tests.

The modifier "_nobo" combined with "_kd" produces models for document instances with no back-off smoothing by label-conditional models. The modifiers "_bm25X" use BM25 instead of Multinomial distances. Combined with "_kd" and "_nobo", this produces a model that uses BM25 for kernel densities of each label.

The modifiers "_ndcg5", "_mjac", "_mifs" and "_mafsX" choose the optimization measure for MetaOpt2.py. These correspond to NDCG@5, Mean of Jaccard scores per instance, micro-averaged Fscore, macro-averaged Fscore and surrogate measures for maFscore. It was noticed early in the competition that computing and optimizing maFscore is problematic, since not all labels are present in the training set, and any subset chosen for optimization will contain only a tiny fraction of the 325k+ labels, with the rest being missing labels. Since most la-

VI

bels are missing labels, and any number of false positives for a missing label will equal an fscore of 0, optimizing maFscore becomes problematic. The "_mafsX" surrogates used two attempts to penalize for false positives of missing labels, but these was abandoned for a method that allows optimizing macroFscore better without producing too many instances per label.

The modifiers "_iwX" select a method developed in this competition. This causes the base-classifier to predict instances per label, instead of labels per instance. A sorted list of the best scores for each label is stored, and for each classified instance the lists for labels are updated. A full distribution of labels is computed for each instance, and the label→instance scores are computed from the rank of the label for each instance. After classification of the dataset, the sparse label→instances scores are transposed and outputted and evaluated in the instance→labels format. The arguments -instantiate_weight X and -instantiate_threshold X passed to SGM_Tests control the number of top scoring instances stored for each label. The ensemble combination uses transposed prediction of the same kind to do the classification.

4 Ensemble Model

Source files: RUN_METACOMB, MetaComb2.java, TransposeFile.py, SelectClassifiers.py, SelectDevLabels.py, comb_dev_results/, eval_results/, weka.jar

The ensemble model is built on our earlier LSHTC3 ensemble [3], but performs classification by predicting instances per label. The classifier vote weight prediction is a case of Feature-Weighted Linear Stacking [4], but the regression models are trained separately for each base-classifier, using reference weights that approximate optimal weights per label in a development set.

The base-classifier result files are tranposed from a document→labels per line format to a label→documents per line format. After prediction the ensemble result file is transposed back to the document→labels per line .csv format used by the competition. The script RUN_METACOMB performs all the required steps, using the result files stored in /comb_dev_results for training the ensemble and /eval_results to do the classifier combination.

Metacomb2.java perfoms the ensemble classification. The ensemble uses linear regression models to predict the weight of each base-classifier, using metafeatures computed from label information and classifier outputs to predict the optimal classifier weight for each label. The most useful metafeatures in the LSHTC3 submission used labelset correlation features between the base-classifiers for each document instance [3]. This ensemble uses instance-set correlation features for each label analogously.

metafeature	description
labelProb	indicator feature for low-frequency labels (<10)
labelProb2	indicator feature for high-frequency labels (>50)
uniqInstancesets	# different instance sets in the classifier outputs
maxVotes	# votes given to most voted instance set
minInstFreq_i	the frequency of least frequent instance in the output of classifier i
maxInstFreq_i	the frequency of most frequent instance in the output of classifier i
minInstCount_i	the count of the lowest count instance in the output of classifier i
instCount_i	# instances in the output of classifier i
emptySet_i	indicator if the classifier output i has no instances for the label
setCount_i	# of classifiers with the same output as classifier i
modePrec_i	precision of classifier i using the mode of outputs as reference
modeRec_i	recall of classifier i using the mode of outputs as reference
modeJaccard_i	Jaccard similarity of classifier i and the mode of outputs
maxPrec_i_j	intersection of classifier i and j outputs, divided by maximum length

Table 2. Metafeatures used for voting classifier weights. Metafeatures are computed for each label, given the instance set outputs from each base-classifier. Regression models for each baseclassifier weight uses the features that match the classifier id i, and not the metafeatures for other classifiers. Metafeature maxPrec_i_j is computed for all the other base-classifiers j, resulting in 42-1 additional metafeatures. Metafeatures are normalized and log-transformed based on development set performance.

Table 2 shows the metafeatures used by MetaComb2.java. For efficiency and memory use, MetaComb2 adds the correlation metafeatures to each base-classifier before predicting the vote weights, and doesn't keep all possible metafeatures in memory at any time. This keeps the memory complexity of the ensemble combination linear in the number of base-classifiers. Functions constuctData(), pruneGlobalFeatures() and addLocalFeatures() in MetaComb2.java show how the features are constructed as Weka [5] Instances.

The regression models use Weka LinearRegression for implementing the variant of Feature-Weighted Linear Stacking. For each label in comb_dev.txt, optimal reference weights are approximated by distributing a weight of 1 uniformly to the base-classifiers that score highest on the performance measure. Initially fscore was used as the measure, as averaging the fscores across the labels gives maFscore. This however doesn't use rank information in the instance sets. A small improvement in maFscore was gained by using a similar measure that takes rank information into account. approximateOracleWeights() and updateEvaluationResults() in MetaComb2.java show how the reference vote weights are constructed.

Following vote weight prediction, the label→instances scores are summed for each instance from the weighted votes in the function voteFold(). A combination of label prior information and thresholding similar to one used in the base-classifiers is used to choose the number of instances per label. The label prior information selects a number of instances for the label proportional to the

VIII

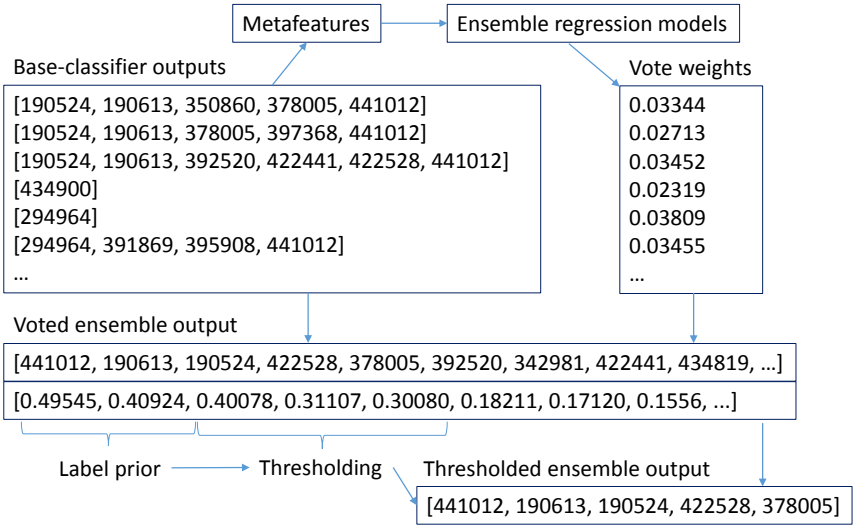


Fig. 1. Ensemble voting and selection of instances from the base-classifier outputs.

label frequency in training data, multiplied by the parameter 0.95 passed to set_instantiate(). The thresholding then includes to the set all instances with score more than 0.5 of the mean of the initial instance set scores. Figure 1 illustrates the ensemble combination and selection of instances.

Development of the ensemble by n-fold cross-validation can be done by changing the global variable "developmentRun" in MetaComb2.java to 1. Selection of base-classifiers can be done by giving the classifiers to remove as integer arguments to MetaComb2. The list of removed classifiers used in the final evaluation run in RUN_METACOMB was developed by running the classifier selection script SelectClassifiers.py with the n-fold crossvalidation. SelectClassifiers.py performs hill-climbing searches, maximizing the output of MetaComb2 by removing and adding classifiers to the ensemble.

5 How to Generate the Solution

The programs and scripts described above can be run to produce the winning submission file. Some of the programs can take considerable computing resources to produce. Both optimizing the base-classifier parameters and classifying the 452k document test set can take several days or more, depending on the model. We used a handful of quadcore i7-2600 CPU processors with 16GB RAM over

the competition period to develop and optimize the models. At least 16GB RAM is required to store the word counts reaching 100M parameters. Ensemble combination takes less than 8GB memory, and can be computed from the provided .results files. The base-classifier result files are included in the distribution, as computing these takes considerable time.

For optimizing base-classifiers, compile SGM_Tests.java with javac, configure Make_templates.py or copy an existing template, and run RUN_DEVS. For classifying the comb_dev.txt and test.txt results with a base-classifier, configure and run RUN_EVALS. For combining the base-classifier results with the ensemble, run RUN_METACOMB. The global variables in each script can be modified to change configurations.

6 What Gave us the First Place?

The competition posed a number of complications different from usual Kaggle competitions. Most of our tools were developed over the last LSHTC challenges, and this gave us a big advantage. The biggest complication in the competition was scalability of both the base-classifiers and ensemble. Our solution uses sparse storage and inverted indices for inference, a modeling idea that enabled us to use an ensemble of tens of base-classifiers. With the SGMWeka toolkit we could combine parameterized feature weighting [3], hierarchical smoothing [2], kernel densities [2], model-based feedback [6], etc. Other participants used KNN with inverted indices, but our solution provides a diversity of structured probabilistic base-classifiers with much better modeling accuracies.

Another complication was the preprocessed pruned feature vectors. This made usual Multinomial or Language Model solutions usable only with very untypical and heavy use of linear interpolation smoothing. The commonly used TF-IDF feature transforms also corrected the problem only somewhat. Our solutions for smoothing and feature weighting with a customized BM25 variant took extensive experimentation to discover, but improved the accuracy considerably. It is likely that the other teams had less sophisticated text similarity measures available, and the ones having good measures scored better in the contest.

A second difficult complication in the contest was the choice of maFscore for evaluation measure, in contrast to earlier LSHTC competitions. What surprised the contestants was that optimization of maFscore with high numbers of labels is problematic, since most labels will be missing. With maFscore a label occurring once is just as important as one occurring 1000 times, and a label never predicted and one predicted by a 1000 false positives have the same effect on the score. Combined with most labels missing, normal optimization of classifiers proved difficult. It took us some time to figure out the right way to solve this problem, but the solution made it possible for us to compete for the win. Before

x

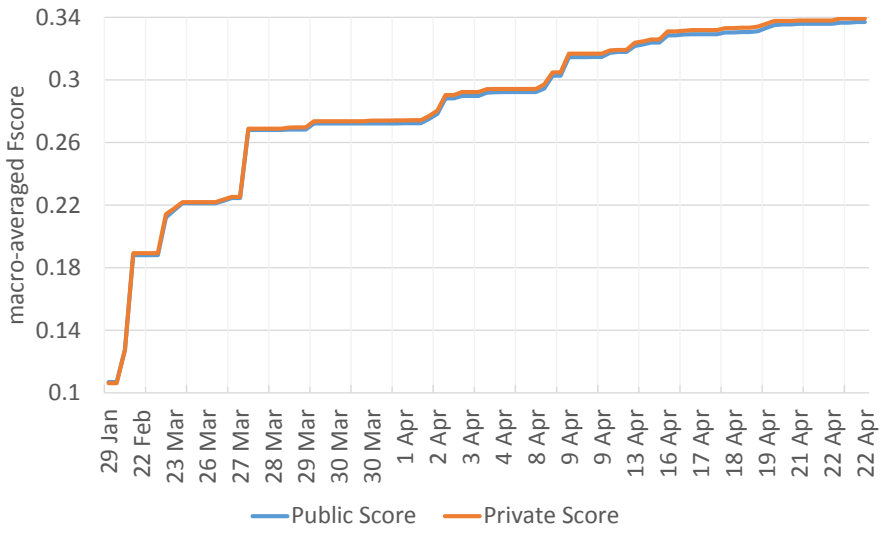


Fig. 2. Progress on the test.txt macro-averaged Fscore during the competition. Growing the ensemble brought steady improvement, implementing the transposed prediction caused jumps in the maFscore.

developing the transposed prediction used in both the base-classifiers and the ensemble, our leaderboard score was around 22%. A couple of simple corrections for maximizing maFscore correctly brought the ensemble combination close to 27%, and using the transposed prediction with a larger and more diverse ensemble gave us the final score close to 34%. Other participants noticed this problem of optimizing maFscore, but likely most of them did not find a good solution.

The use of metafeature regression in the ensemble instead of majority voting provided a moderate improvement of about 0.5%, and this much was needed for the win. It is likely that the metafeatures optimized on the 23k comb_dev.txt documents looked different from the metafeatures computed for the 452k test.txt documents, even though the metafeatures were chosen or normalized to be stable to change in the number of documents. The optimal amount of regularization for the Weka LinearRegression was untypically high at 1000. More complicated Weka regression models for the vote weight prediction failed to improve the test set score, likely due to overfitting the somewhat unreliable features. Another reason could be the small size of the comb_dev.txt for ensemble combination. The ensemble fits the parameters for 55 metafeatures to predict the vote weight of each of the 42 base-classifiers, using only 23k points of data shared by the 42 regression models. The improvement from Feature Weighted Linear Stacking could have been considerably larger, if a larger training set had been segmented for the ensemble.

7 Acknowledgements

We'd like to thank Kaggle and the LSHTC organizers for their work in making the competition a success, and the machine learning group at the University of Waikato for the computers we used for our solution.

References

[1] Puurula, A.: Scalable text classification with sparse generative modeling. In: Proceedings of the 12th Pacific Rim International Conference on Trends in Artificial Intelligence. PRICAI'12, Berlin, Heidelberg, Springer-Verlag (2012) 458–469

[2] Puurula, A., Myaeng, S.H.: Integrated instance- and class-based generative modeling for text classification. In: Proceedings of the 18th Australasian Document Computing Symposium. ADCS '13, New York, NY, USA, ACM (2013) 66–73

[3] Puurula, A.: Combining modifications to multinomial naive bayes for text classification. In Hou, Y., Nie, J.Y., Sun, L., Wang, B., Zhang, P., eds.: Information Retrieval Technology. Volume 7675 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 114–125

[4] Sill, J., Takcs, G., Mackey, L., Lin, D.: Feature-weighted linear stacking. CoRR **abs/0911.0460** (2009)

[5] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explor. Newsl. **11**(1) (November 2009) 10–18

[6] Puurula, A.: Cumulative progress in language models for information retrieval. In: Proceedings of the Australasian Language Technology Association Workshop 2013 (ALTA 2013), Brisbane, Australia (December 2013) 96–100

References

Alekh Agarwal, Oliveier Chapelle, Miroslav Dudík, and John Langford. A Reliable Effective Terascale Linear Learning System. *Journal of Machine Learning Research*, 15:1111–1133, 2014. 154

Charu C. Aggarwal and ChengXiang Zhai, editors. *Mining Text Data*. Springer, 2012. ISBN 978-1-4419-8462-3. 4, 13, 26

Eugene Agichtein. Scaling Information Extraction to Large Document Collections. In *IEEE Data Engineering Bulletin*, 2005. 43, 46

Helena Ahonen, Oskari Heinonen, Mika Klemettinen, and A. Inkeri Verkamo. Applying Data Mining Techniques in Text Analysis. Technical report, 1997a. 10, 12, 13, 18, 34, 35

Helena Ahonen, Oskari Heinonen, Mika Klemettinen, and A.Inkeri Verkamo. Mining in the phrasal frontier. In Jan Komorowski and Jan Zytkow, editors, *Principles of Data Mining and Knowledge Discovery*, volume 1263 of *Lecture Notes in Computer Science*, pages 343–350. Springer Berlin Heidelberg, 1997b. 10, 12, 43

Rudolf Albrecht, Karl Schwarzschild Str, D-Garching, and Dieter Merkl. Knowledge Discovery in Literature Data Bases. In *Library and Information Services in Astronomy III (LISA III)*, volume 153 of *ASP Conference Series*, 1998. 10

James Allan, Ron Papka, and Victor Lavrenko. On-line New Event Detection and Tracking. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 37–45, New York, NY, USA, 1998. ACM. 28

Gene M. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM. 43

Sarabjot S. Anand, David A. Bell, and John G. Hughes. The Role of Domain Knowledge in Data Mining. In *Proceedings of the Fourth International*

Conference on Information and Knowledge Management, CIKM '95, pages 37–43, New York, NY, USA, 1995. ACM. 20

Björn Andres, Thorsten Beier, and Jörg H. Kappes. OpenGM: A C++ Library for Discrete Graphical Models. *CoRR*, abs/1206.0111, 2012. 63, 66, 154

Nikolay Archak, Anindya Ghose, and Panagiotis G. Ipeirotis. Show me the money!: deriving the pricing power of product features by mining consumer reviews. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 56–65, New York, NY, USA, 2007. ACM. 17, 28, 29

Nikolay Archak, Anindya Ghose, and Panagiotis G. Ipeirotis. Deriving the Pricing Power of Product Features by Mining Consumer Reviews. *Management Science*, 57(8):1485–1509, 2011. 17, 28

Nima Asadi and Jimmy Lin. Document vector representations for feature extraction in multi-stage document ranking. *Information Retrieval*, 16(6): 747–768, 2013. 40

Arthur Asuncion, Max Welling, Padhraic Smyth, and Yee Whye Teh. On smoothing and inference for topic models. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 27–34, Arlington, Virginia, United States, 2009. AUAI Press. 58, 68, 69

Xavier L. Aubert. An overview of decoding techniques for large vocabulary continuous speech recognition. *Computer Speech & Language*, 16(1):89 – 114, 2002. 109, 156

T. Bailey and C. Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. Technical report, UCSD, 1994. 55

Arindam Banerjee, Inderjit S. Dhillon, Joydeep Ghosh, and Suvrit Sra. Clustering on the Unit Hypersphere using von Mises-Fisher Distributions. *Journal of Machine Learning Research*, 6:1345–1382, 2005. 29, 51

Michele Banko and Eric Brill. Scaling to Very Very Large Corpora for Natural Language Disambiguation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, ACL '01, pages 26–33, Stroudsburg, PA, USA, 2001. Association for Computational Linguistics. 40

Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open Information Extraction from the Web. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2670–2676, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. 46

Bruce E. Bargmeyer and Daniel W. Gillman. Metadata Standards and Metadata Registries: An Overview. In *International Conference on Establishment Surveys II*, 2000. 19

Rahul C. Basole, C. David Seuss, and William B. Rouse. It innovation adoption by enterprises: Knowledge discovery through text analytics. *Decision Support Systems*, 54(2):1044–1054, 2013. 17

Leonard E. Baum and Ted Petrie. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966. 61

Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics*, 41(1): 164–171, 1970. 34, 71, 85

Robert Baumgartner, Georg Gottlob, and Marcus Herzog. Scalable Web Data Extraction for Online Market Intelligence. *Proc. VLDB Endow.*, 2(2):1512–1523, August 2009. 17, 43

Timothy Bell, Ian H. Witten, and John G. Cleary. Modeling for text compression. *ACM Comput. Surv.*, 21(4):557–591, December 1989. 61

R. Bellman. On the Theory of Dynamic Programming. In *Proceedings of the National Academy of Sciences*, volume 38, pages 716–719, 1952. 2, 68, 70

Yoshua Bengio. Learning Deep Architectures for AI. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009. 154

Paul N. Bennett. Assessing the calibration of naive Bayes posterior estimates. Technical report, 2000. 52

Paul N. Bennett. Using asymmetric distributions to improve text classifier probability estimates. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, SIGIR '03, pages 111–118, New York, NY, USA, 2003. ACM. 51

Shane Bergsma, Emily Pitler, and Dekang Lin. Creating Robust Supervised Classifiers via Web-scale N-gram Data. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 865–874, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. 40

Dmitriy Bespalov, Yanjun Qi, Bing Bai, and Ali Shokoufandeh. Sentiment Classification with Supervised Sequence Embedding. In *ECML/PKDD (1)*, pages 159–174, 2012. 127, 128

C. Bielza, G. Li, and P. Larraòaga. Multi-dimensional Classification with Bayesian Networks. *Int. J. Approx. Reasoning*, 52(6), September 2011. 27

Jeff Bilmes. A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Technical report, International Computer Science Institute, 1998. 54, 55, 61, 71, 85

Maximilian Bisani and Hermann Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434 – 451, 2008. 88

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ix, 30, 33, 50

William Black. Encyclopedia of Language & Linguistics. chapter Text Mining. Elsevier Ltd, 2006. 4, 11, 15

D. Blei and J. Lafferty. Correlated topic models. In *Advances in Neural Information Processing Systems*, 2006. 58

David M. Blei. Probabilistic topic models. *Commun. ACM*, 55(4):77–84, 2012. 58, 68, 69

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003. ISSN 1532-4435. 11, 13, 24, 33, 35, 39, 48, 57, 96

John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL*, 2007. 127

Harold Borko and Myrna Bernick. Automatic Document Classification. *J. ACM*, 10(2):151–162, April 1963. 24

Harold Borko and Myrna Bernick. Automatic Document Classification Part II . Additional Experiments. *J. ACM*, 11(2):138–151, April 1964. 24, 33

Lutz Bornmann and Rüdiger Mutz. Growth rates of modern science: A bibliometric analysis. *CoRR*, 2014. 17

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM. 33

Léon Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, August 2010. Springer. 33, 42

Léon Bottou. Stochastic Gradient Descent Tricks. In Grégoire Montavon, GenevièveB. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 421–436. Springer Berlin Heidelberg, 2012. 33, 34

Guillaume Bouchard and Bill Triggs. The tradeoff between generative and discriminative classifiers. In *IASC International Symposium on Computational Statistics (COMPSTAT)*, pages 721–728, Prague, August 2004. 98

Matthew R. Boutell, Jiebo Luo, Xipeng Shen, and Christopher M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757 – 1771, 2004. 117, 144

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, Jeffrey Dean, and Google Inc. Large language models in machine translation. In *In EMNLP*, pages 858–867, 2007. 38, 40, 41, 59

L. Breiman. Statistical modeling: The two cultures. *Statistical Science*, 16(3): 199–215, 2001a. 13

Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001b. 34

Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984. 34

Alan Brier and Bruno Hopp. Computer assisted text analysis in the social sciences. *Quality & Quantity*, 45(1):103–128, 2011. 17

Christian Buck, Kenneth Heafield, and Bas van Ooyen. N-gram Counts and Language Models from the Common Crawl. In *Proceedings of the Language Resources and Evaluation Conference*, Reykjavík, Iceland, May 2014. 20, 40

Gilbert Burck. *The computer age and its potential for management*. Harper & Row, 1965. 21

Michael J. Cafarella and Oren Etzioni. A search engine for natural language applications. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 442–452, New York, NY, USA, 2005. ACM Press. 46

Michael J. Cafarella, Doug Downey, Stephen Soderland, and Oren Etzioni. KnowItNow: fast, scalable information extraction from the web. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 563–570, Stroudsburg, PA, USA. Association for Computational Linguistics. 46

Aoife Cahill, Michael Burke, Ruth O'Donovan, Josef van Genabith, and Andy Way. Long-distance Dependency Resolution in Automatically Acquired Wide-coverage PCFG-based LFG Approximations. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics. 20, 21

Berkant Barla Cambazoglu and Ricardo A. Baeza-Yates. Scalability and efficiency challenges in commercial web search engines. In *SIGIR*, page 1124, 2013. 40

Qing Cao, Wenjing Duan, and Qiwei Gan. Exploring Determinants of Voting for the "Helpfulness" of Online User Reviews: A Text Mining Approach. *Decis. Support Syst.*, 50(2):511–521, January 2011. 28

Ana Cardoso-Cachopo. *Improving Methods for Single-label Text Categorization*. PhD thesis, Instituto Superior Técnico - Universidade Ténica de Lisboa, October 2007. 127, 128

Bob Carpenter. Lazy sparse stochastic gradient descent for regularized multinomial logistic regression. International finance discussion papers, Alias-i, 2008. 33

Richard Caruana. Multitask Learning: A Knowledge-Based Source of Inductive Bias. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 41–48. Morgan Kaufmann, 1993. 27

Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. Training and Testing Low-degree Polynomial Data Mappings via Linear SVM. *J. Mach. Learn. Res.*, 11:1471–1490, August 2010. 33

K. Chard, M. Russell, Y. A. Lussier, E. A. Mendonca, and J. C. Silverstein. Scalability and Cost of a Cloud-based Approach to Medical NLP. In *Proceedings of the 2011 24th International Symposium on Computer-Based Medical*

Systems, CBMS '11, pages 1–6, Washington, DC, USA, 2011. IEEE Computer Society. 43

Chandra Chekuri, Michael H. Goldwasser, Prabhakar Raghavan, and Eli Upfal. Web Search Using Automatic Classification. In *In Sixth World Wide Web Conference*, 1997. 42

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. One billion word benchmark for measuring progress in statistical language modeling. 2013. 38

Ming-Syan Chen, Jiawei Han, and Philip S. Yu. Data Mining: An Overview from a Database Perspective. *IEEE Trans. on Knowl. and Data Eng.*, 8(6): 866–883, December 1996. 12

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, ACL '96, pages 310–318, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics. 59, 77, 80

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4): 359–393, 1999. 59, 60, 77, 78, 79, 80, 102

E Choi and P Hall. Data sharpening as a prelude to density estimation. *Biometrika*, 86(4):941–947, 1999. 113

N. Chomsky. Three models for the description of language. *Information Theory, IRE Transactions on*, 2(3):113–124, September 1956. 45, 59

Pirooz Chubak and Davood Rafiei. Efficient Indexing and Querying over Syntactically Annotated Trees. *PVLDB*, 5(11):1316–1327, 2012. 24

Kenneth W. Church and William A. Gale. Poisson Mixtures. In *Natural Language Engineering*, volume 1, pages 163–190, 1995. 51

Kenneth Ward Church. A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In *Proceedings of the Second Conference on Applied Natural Language Processing*, ANLC '88, pages 136–143, Stroudsburg, PA, USA, 1988. Association for Computational Linguistics. 45

Patrick Marques Ciarelli, Elias Oliveira, and Claudine Badue. Multi-Label Text Categorization Using a Probabilistic Neural Network. *International Journal of Computer Information Systems and Industrial Management Applications*, 1:133–144, 2009. 51

John G. Cleary and Ian H. Witten. Data Compression using Adaptive Coding and Partial String Matching. *IEEE Transactions on Communications*, 32 (4):396–402, April 1984. 79

Aaron M. Cohen and William R. Hersh. A survey of current work in biomedical text mining. *Briefings in Bioinformatics*, 6(1):57–71, March 2005. 11, 17

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural Language Processing (Almost) from Scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November 2011. 39, 45, 154

Gordon Cormack. TREC 2006 Spam Track Overview. In *Proceedings of TREC 2006*, 2006. 127

Gordon V. Cormack and Thomas R. Lynam. Validity and Power of T-test for Comparing MAP and GMAP. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 753–754, New York, NY, USA, 2007. ACM. 125

Michele Coscia and Viridiana Rios. Knowing Where and How Criminal Organizations Operate Using Web Content. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 1412–1421, New York, NY, USA, 2012. ACM. 17

T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, January 1967. 28, 33

Mark Craven, Dan Dipasquo, Dayne Freitag, Andrew K. Mccallum, Tom M. Mitchell, Kamal Nigam, and Seán Slattery. Learning to Construct Knowledge Bases from the World Wide Web. *Artificial Intelligence*, 118(1/2): 69–113, 2000. 51, 52, 88

W.B. Croft and D.J. Harper. Using probabilistic models of document retrieval without relevance information. *Journal of Documentation*, 35:285–295, 1979. 87

J. N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. In *The Annals of Mathematical Statistics*, volume 43, pages 1470–1480, 1972. 89

Dipanjan Das and Andre' F. T. Martins. A Survey on Automatic Text Summarization. Technical report, November 2007. 15

Michael Daum, Kilian A. Foth, and Wolfgang Menzel. Constraint Based Integration of Deep and Shallow Parsing Techniques. In *EACL*, pages 99–106. The Association for Computer Linguistics, 2003. 45

Kushal Dave, Steve Lawrence, and David M. Pennock. Mining the Peanut Gallery: Opinion Extraction and Semantic Classification of Product Reviews. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 519–528, New York, NY, USA, 2003. ACM. 17, 24

Jeffrey Dean and Luiz André Barroso. The Tail at Scale. *Commun. ACM*, 56 (2):74–80, February 2013. 42

Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. 44, 50, 54, 154

Reinhold Decker and Michael Trusov. Estimating aggregate consumer preferences from online product reviews. *International Journal of Research in Marketing*, 27(4):293 – 307, 2010. 17

Scott Deerwester. Improving Information Retrieval with Latent Semantic Indexing. In Christine L. Borgman and Edward Y. H. Pai, editors, *Proceedings of the 51st ASIS Annual Meeting (ASIS '88)*, volume 25, Atlanta, Georgia, October 1988. American Society for Information Science. 24, 155

Sarah Jane Delany, Pádraig Cunningham, and Barry Smyth. ECUE: A Spam Filter that Uses Machine Learning to Track Concept Drift. In *Proceedings of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29 – September 1, 2006, Riva del Garda, Italy*, pages 627–631, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press. 127, 128

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977. 55

Kris Demuynck, Antti Puurula, Dirk Van Compernolle, and Patrick Wambacq. The esat 2008 system for n-best dutch speech recognition benchmark. In *ASRU*, pages 339–344, 2009. 22

Janez Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.*, 7:1–30, December 2006. 125

Murat Deviren, Khalid Daoudi, and Kamel Smaïli. Language modeling using dynamic Bayesian networks. In *4th International Conference on Language Resources and Evaluation - LREC 2004*, Lisbonne, Portugal, 2004. 64, 65

Murat Deviren, Khalid Daoudi, and Kamel Smali. Rethinking Language Models Within the Framework of Dynamic Bayesian Networks. In Balázs Kégl and Guy Lapalme, editors, *Advances in Artificial Intelligence*, volume 3501 of *Lecture Notes in Computer Science*, pages 432–437. Springer Berlin Heidelberg, 2005. 64, 65

Thomas G. Dietterich. Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Comput.*, 10(7):1895–1923, October 1998. 125

Laura Dietz. Directed Factor Graph Notation for Generative Models. Technical report, Max Planck Institute for Informatics, Saarbrücken, Germany, 2010. 63, 66

Pedro Domingos and Michael Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Mach. Learn.*, 29(2-3):103–130, November 1997. 49, 50, 51

Jochen Dörre, Peter Gerstl, and Roland Seiffert. Text mining: finding nuggets in mountains of textual data. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '99, pages 398–401, New York, NY, USA, 1999. ACM. 10, 13, 35, 37

John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.*, 12: 2121–2159, July 2011. 33, 154

Daniel M. Dunlavy, Timothy M. Shead, and Eric T. Stanton. ParaText: scalable text modeling and analysis. In Salim Hariri and Kate Keahey, editors, *HPDC*, pages 344–347. ACM, 2010. 43

Greg Durrett and Dan Klein. An Empirical Investigation of Discounting in Cross-domain Language Models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 24–29, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. 77

Charles Elkan. Boosting And Naive Bayesian Learning. Technical report, 1997. 11, 48

Tamer Elsayed, Jimmy Lin, and Douglas W. Oard. Pairwise Document Similarity in Large Collections with MapReduce. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, HLT-Short '08, pages 265–268, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics. 42

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A Library for Large Linear Classification. *J. Mach. Learn. Res.*, 9:1871–1874, June 2008. ISSN 1532-4435. 29, 32, 33

Weiguo Fan, Linda Wallace, Stephanie Rich, and Zhongju Zhang. Tapping the Power of Text Mining. *Commun. ACM*, 49(9):76–82, September 2006. 15

Hui Fang, Tao Tao, and ChengXiang Zhai. A Formal Study of Information Retrieval Heuristics. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 49–56, New York, NY, USA, 2004. ACM. 32

Michele Fattori, Giorgio Pedrazzi, and Roberta Turra. Text mining applied to patent mapping: a practical business case. *World Patent Information*, 25 (4):335 – 342, 2003. 28

R. R. Favreau and R. G. Franks. Statistical optimization. In *Proceedings Second International Analog Computer Conference*, 1958. 123

Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. Advances in Knowledge Discovery and Data Mining. chapter From Data Mining to Knowledge Discovery: An Overview, pages 1–34. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996. 10, 12, 35, 43

Ingo Feinerer, Kurt Hornik, and David Meyer. Text mining infrastructure in r. *Journal of Statistical Software*, 25(5):1–54, 3 2008. ISSN 1548-7660. URL `http://www.jstatsoft.org/v25/i05`. 12

R. Feldman and I. Dagan. KDT - Knowledge Discovery in Texts. In *Proc. of the First Int. Conf. on Knowledge Discovery (KDD)*, pages 112–117, 1995. 10, 34

Ronen Feldman and James Sanger. *Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, New York, NY, USA, 2006. 2, 4, 10, 13, 15, 20, 22, 26, 34, 35

Ronen Feldman, Willi Klösgen, Yaniv Ben-Yehuda, Gil Kedar, and Vladimir Reznikov. Pattern Based Browsing in Document Collections. In *Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery*, PKDD '97, pages 112–122, London, UK, UK, 1997. Springer-Verlag. 10, 34

Ronen Feldman, Moshe Fresko, Yakkov Kinar, Yehuda Lindell, Orly Liphstat, Martin Rajman, Yonatan Schler, and Oren Zamir. Text Mining at the Term Level. In *PKDD*, pages 65–73, 1998. 10

George Forman. An Extensive Empirical Study of Feature Selection Metrics for Text Classification. *J. Mach. Learn. Res.*, 3:1289–1305, March 2003. 42

Eibe Frank and Remco R. Bouckaert. Naive bayes for text classification with unbalanced classes. In *Proceedings of the 10th European conference on Principle and Practice of Knowledge Discovery in Databases*, PKDD'06, pages 503–510, Berlin, Heidelberg, 2006. Springer-Verlag. 48, 51, 53, 86

Eibe Frank, Leonard Trigg, Geoffrey Holmes, and Ian H. Witten. Naive Bayes for Regression. In *Machine Learning*, pages 5–26, 1998. 50, 52, 155

Ildiko E. Frank and Jerome H. Friedman. A Statistical View of Some Chemometrics Regression Tools. *Technometrics*, 35(2):109–135, 1993. 30

B.J. Frey and Nebojsa Jojic. A comparison of algorithms for inference and learning in probabilistic graphical models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(9):1392–1416, 2005. 62, 64, 66

Brendan J. Frey. Extending factor graphs so as to unify directed and undirected graphical models. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, UAI'03, pages 257–264, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc. 66, 67

Jerome H. Friedman. Stochastic Gradient Boosting. *Comput. Stat. Data Anal.*, 38(4):367–378, February 2002. 34

Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian Network Classifiers. *Machine Learning*, 29(2-3):131–163, November 1997. 69

M. Frydenberg. The chain graph Markov property. *Scandinavian Journal of Statistics*, 17:333–353, 1990. 66

K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21(1):32–40, January 1975. 113

Mark Gales and Steve Young. The Application of Hidden Markov Models in Speech Recognition. *Found. Trends Signal Process.*, 1(3):195–304, January 2007. ISSN 1932-8346. 97

A. Gammerman, V. Vovk, and V. Vapnik. Learning by Transduction. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 148–155, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. 26

Michael Gamon. Sentiment Classification on Customer Feedback Data: Noisy Data, Large Feature Vectors, and the Role of Linguistic Analysis. In *Proceedings of the 20th International Conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics. 24

Anindya Ghose and Panagiotis G. Ipeirotis. Estimating the Helpfulness and Economic Impact of Product Reviews: Mining Text and Reviewer Characteristics. *IEEE Trans. on Knowl. and Data Eng.*, 23(10):1498–1512, October 2011. 17, 28

Shantanu Godbole and Shourya Roy. Text to Intelligence: Building and Deploying a Text Mining Solution in the Services Industry for Customer Satisfaction Analysis. In *IEEE SCC (2)*, pages 441–448. IEEE Computer Society, 2008. 17, 37

Sharon Goldwater, Thomas L. Griffiths, and Mark Johnson. Interpolating between Types and Tokens by Estimating Power-Law Generators. *Advances in Neural Information Processing Systems*, 18, 2006. 79, 80

J. Goodman. A Bit of Progress in Language Modeling. Technical report, Microsoft Research, 56 Fuchun Peng, 2000. 59, 79, 85

Siddharth Gopal and Yiming Yang. Recursive Regularization for Large-scale Classification with Hierarchical and Graphical Dependencies. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 257–265, New York, NY, USA, 2013. ACM. 44, 45

William Sealy Gosset. The Probable Error of a Mean. *Biometrika*, 6(1):1–25, March 1908. Originally published under the pseudonym "Student". 125

Thomas L. Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America*, 101 (Suppl 1):5228–5235, April 2004. 58

Justin Grimmer and Brandon M. Stewart. Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts. *Political Analysis*, 21(3):267–297, July 2013. 17

Ralph Grishman and Beth Sundheim. Message Understanding Conference-6: A Brief History. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 1*, COLING '96, pages 466–471, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics. 14

Marko Grobelnik, Dunja Mladenic, and Natasa Milic-Frayling. Text mining as integration of several related research areas: Report on kdd'2000 workshop on text mining. *SIGKDD Explorations*, 2(2):99–102, 2000. 12

Oskar Gross, Antoine Doucet, and Hannu Toivonen. Named Entity Filtering based on Concept Association Graphs. In Alexander F. Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing - 14th International Conference, CICLing 2013, Samos, Greece, March 24-30, 2013*, page 11, 2013. 38

Axel Groß-Klußmann and Nikolaus Hautsch. When machines read the news: Using automated text analytics to quantify high frequency news-implied market reactions. *Journal of Empirical Finance*, 18(2):321 – 340, 2011. 17

Daniel Gruhl, Laurent Chavet, David Gibson, Jörg Meyer, Pradhan Pattanayak, Andrew Tomkins, and Jason Y. Zien. How to build a WebFountain: An architecture for very large-scale text analytics. *IBM Systems Journal*, 43(1):64–77, 2004. 17, 46

Maya R. Gupta and Yihua Chen. Theory and Use of the EM Algorithm. *Found. Trends Signal Process.*, 4(3):223–296, 2011. 55

Eui-Hong Han and George Karypis. Centroid-Based Document Classification: Analysis and Experimental Results. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '00, pages 424–431, London, UK, UK, 2000. Springer-Verlag. ISBN 3-540-41066-X. 29

Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. Scalable Modified Kneser-Ney Language Model Estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, Sofia, Bulgaria, August 2013. 79, 80

Marti A. Hearst. Untangling Text Data Mining. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*, ACL '99, pages 3–10, Stroudsburg, PA, USA, 1999. Association for Computational Linguistics. 4, 11, 14, 17

William Hersh, Chris Buckley, T. J. Leone, and David Hickam. OHSUMED: An Interactive Retrieval Evaluation and New Large Test Collection for Research. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, pages 192–201, New York, NY, USA, 1994. Springer-Verlag New York, Inc. 126, 128

Andrew Hickl, Kirk Roberts, Bryan Rink, Jeremy Bensley, Tobias Jungen, Ying Shi, and John Williams. Question Answering with LCCs Chaucer-2 at TREC 2007. In *In Proceedings of the Sixteenth Text REtrieval Conference*, 2007. 46

Djoerd Hiemstra. A Linguistically Motivated Probabilistic Model of Information Retrieval. In *Research and Advanced Technology for Digital Libraries*, volume 1513 of *Lecture Notes in Computer Science*, pages 569–584. Springer Berlin Heidelberg, 1998. 5, 100

Djoerd Hiemstra. *Using Language Models for Information Retrieval*. PhD thesis, Enschede, January 2001. 4, 81, 86

Djoerd Hiemstra and Wessel Kraaij. Twenty-One at TREC-7: Ad-hoc and Cross-Language Track. In *In Proc. of Seventh Text REtrieval Conference (TREC-7)*, pages 227–238, 1998. 29, 47

Djoerd Hiemstra, Stephen Robertson, and Hugo Zaragoza. Parsimonious language models for information retrieval. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, pages 178–185, New York, NY, USA, 2004. ACM. 80, 85, 86

Derrick Higgins, Chris Brew, Michael Heilman, Ramon Ziai, Lei Chen, Aoife Cahill, Michael Flor, Nitin Madnani, Joel R. Tetreault, Daniel Blanchard, Diane Napolitano, Chong Min Lee, and John Blackmore. Is getting the right answer just about choosing the right words? the role of syntactically-informed features in short answer scoring. *CoRR*, abs/1403.0801, 2014. 28, 29

Mark D. Hill. What is Scalability? *SIGARCH Comput. Archit. News*, 18(4): 18–21, December 1990. 42

Geoffrey E. Hinton. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14(8):1771–1800, 2002. 98

Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '99, pages 50–57, New York, NY, USA, 1999. ACM. 11, 24, 35, 42, 48, 57

Andreas Hotho, Andreas Nrnberger, and Gerhard Paa. A Brief Survey of Text Mining. *LDV Forum - GLDV Journal for Computational Linguistics and Language Technology*, 20(1):19–62, May 2005. 4, 11

Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A Dual Coordinate Descent Method for Large-scale Linear SVM. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 408–415, New York, NY, USA, 2008. ACM. 121

Jian Huang, Jianfeng Gao, Jiangbo Miao, Xiaolong Li, Kuansan Wang, Fritz Behr, and C. Lee Giles. Exploring Web Scale Language Models for Search Query Processing. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 451–460, New York, NY, USA, 2010. ACM. 40

Songfang Huang and Steve Renals. Power law discounting for n-gram language models. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 5178–5181, 2010. 4, 59, 78, 80, 84, 134

David Hull. Using Statistical Testing in the Evaluation of Retrieval Experiments. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '93, pages 329–338, New York, NY, USA, 1993. ACM. 125

Samuel Huston and W. Bruce Croft. A Comparison of Retrieval Models Using Term Dependencies. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 111–120, New York, NY, USA, 2014. ACM. 5

Frankie James. Modified Kneser-Ney Smoothing of n-gram Models. Technical report, 2000. 79

Kalervo Järvelin and Jaana Kekäläinen. Cumulated Gain-based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, October 2002. 28, 119

Frederick Jelinek and Robert L. Mercer. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, 1980. 59, 60, 76, 80

Lars J. Jensen, Jasmin Saric, and Peer Bork. Literature mining for the biologist: from information retrieval to biological discovery. *Nature Reviews Genetics*, 7(2):119–129, February 2006. 15, 17

Thorsten Joachims. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 143–151, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. 28

Thorsten Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *Proceedings of the 10th European Conference on Machine Learning*, ECML '98, pages 137–142, London, UK, UK, 1998. Springer-Verlag. 11, 13, 29, 33, 44, 48, 50, 127

Thorsten Joachims. Transductive Inference for Text Classification Using Support Vector Machines. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 200–209, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2. 26

George H. John and Pat Langley. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, UAI'95, pages 338–345, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. 113

W. E. Johnson. Probability : deductive and inductive problems. *Mind*, 41: 421–423, 1932. 76

Mahesh Joshi, Dipanjan Das, Kevin Gimpel, and Noah A. Smith. Movie Reviews and Revenues: An Experiment in Text Regression. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 293–296, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. 28, 29

Alfons Juan and Hermann Ney. Reversing and Smoothing the Multinomial Naive Bayes Text Classifer. In *In Proceedings of the 2nd Int. Workshop on Pattern Recognition in Information Systems (PRIS 2002*, pages 200–212, 2002. 53, 54, 88, 95

Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition) (Prentice Hall Series in Artificial Intelligence)*. Prentice Hall, 2 edition, 2008. 20

T. Kalt. A New Probabilistic Model of Text Classification and Retrieval. Technical report, Amherst, MA, USA, 1996. 29, 32, 47

Yuji Kaneda, Naonori Ueda, and Kazumi Saito. Extended parametric mixture model for robust multi-labeled text categorization. In *KES'04*, pages 616–623, 2004. 57

Hillol Kargupta, Ilker Hamzaoglu, and Brian Stafford. Scalable, Distributed Data Mining - An Agent Architecture. In David Heckerman, Heikki Mannila, and Daryl Pregibon, editors, *KDD*, pages 211–214. AAAI Press, 1997. 42

Mikaela Keller and Samy Bengio. Theme Topic Mixture Model: A Graphical Model for Document Representation. In *In: PASCAL Workshop on Learning Methods for Text Understanding and Mining*, pages 04–05, 2004. 56, 57, 58

Daniel Keysers, Roberto Paredes, Enrique Vidal, and Hermann Ney. Comparison of Log-Linear Models and Weighted Dissimilarity Measures. In *Iberian Conference on Pattern Recognition and Image Analysis*, pages 370–377, Puerto de Andratx, Spain, June 2003. Springer Verlag. 33

Ashraf M. Kibriya, Eibe Frank, Bernhard Pfahringer, and Geoffrey Holmes. Multinomial naive bayes for text categorization revisited. In *Proceedings of the 17th Australian joint conference on Advances in Artificial Intelligence*, AI'04, pages 488–499, Berlin, Heidelberg, 2004. Springer-Verlag. 86

Chloe Kiddon and Pedro Domingos. Leveraging Ontologies for Lifted Probabilistic Inference and Learning. In *Statistical Relational Artificial Intelligence*, volume WS-10-06 of *AAAI Workshops*. AAAI, 2010. 46

Sang-Bum Kim, Kyoung-Soo Han, Hae-Chang Rim, and Sung Hyon Myaeng. Some Effective Techniques for Naive Bayes Text Classification. *IEEE Trans. on Knowl. and Data Eng.*, 18(11):1457–1466, November 2006. 51

Ross Kindermann and J. L. Snell. *Markov Random Fields and Their Applications*. AMS, 1980. 62

Dietrich Klakow. Log-linear interpolation of language models. In *ICSLP*, 1998. 98

Roman Klinger and Katrin Tomanek. Classical Probabilistic Models and Conditional Random Fields. Technical Report TR07-2-013, December 2007. 50, 62, 64

Antonina Kloptchenko, Tomas Eklund, Jonas Karlsson, Barbro Back, Hannu Vanharanta, and Ari Visa. Combining Data and Text Mining Techniques for Analysing Financial Reports: Research Articles. *Int. J. Intell. Syst. Account. Financ. Manage.*, 12(1):29–41, January 2004. 17

R. Kneser and H. Ney. Improved backing-off for M-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184, May 1995. 4, 79, 84

Daphne Koller and Mehran Sahami. Hierarchically Classifying Documents Using Very Few Words. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 170–178, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. 45

Petri Kontkanen, Petri Myllymaki, and Henry Tirri. Constructing Bayesian finite mixture models by the EM algorithm. Technical report, ESPRIT Working Group on Neural and Computational Learning (NeuroCOLT, 1996. 56

Anna Korhonen, Diarmuid Saghdha, Ilona Silins, Lin Sun, Johan Hgberg, and Ulla Stenius. Text Mining for Literature Review and Knowledge Discovery in Cancer Risk Assessment and Research. *PLoS ONE*, 7(4):e33427, 04 2012. 17

Kimmo Koskenniemi. Finite-state Parsing and Disambiguation. In *Proceedings of the 13th Conference on Computational Linguistics - Volume 2*, COLING '90, pages 229–232, Stroudsburg, PA, USA, 1990. Association for Computational Linguistics. 45

Eyal Krikon and Oren Kurland. A Study of the Integration of Passage-, Document-, and Cluster-based Information for Re-ranking Search Results. *Inf. Retr.*, 14(6):593–616, December 2011. 102

F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor Graphs and the Sum-Product Algorithm. *IEEE transactions on Information Theory*, 47(2): 498–519, 2001. 66, 71

H. Kucera and W. N. Francis. *Computational analysis of present-day American English*. Brown University Press, Providence, RI, 1967. 14

Taku Kudo and Yuji Matsumoto. Fast Methods for Kernel-based Text Analysis. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 24–31, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. 25

Julian Kupiec. Robust part-of-speech tagging using a hidden markov model. *Computer Speech & Language*, 6(3):225 – 242, 1992. 34

Mikko Kurimo, Sami Virpioja, Ville Turunen, and Krista Lagus. Morpho Challenge 2005-2010: Evaluations and Results. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, pages 87–95, Uppsala, Sweden, July 2010. Association for Computational Linguistics. 23

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. 11, 13, 34, 39

Mirella Lapata and Frank Keller. Web-based Models for Natural Language Processing. *ACM Trans. Speech Lang. Process.*, 2(1), February 2005. 40

Pierre-Simon Laplace. *Essai philosophique sur les probabilits.* Paris: Courcier, 1814. 76

Leah S. Larkey. Automatic Essay Grading Using Text Categorization Techniques. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 90–95, New York, NY, USA, 1998. ACM. 24, 28, 97

Peder Olesen Larsen and Markus von Ins. The rate of growth in scientific publication and the decline in coverage provided by Science Citation Index. *Scientometrics*, 84(3):575–603, 2010. 17

Steffen L. Lauritzen and David J. Spiegelhalter. Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. *Journal of the Royal Statistical Society, Series B*, 50(2):157–224, 1988. 62

Nevena Lazic, C. M. Bishop, and J. Winn. Structural expectation propagation (sep): Bayesian structure learning for networks with latent variables. In *Proceedings Sixteenth International Conference on Artificial Intelligence and Statistics (AIStats)*. AISTATS, 2013. 66, 69

Heeyoung Lee, Mihai Surdeanu, Bill Maccartney, and Dan Jurafsky. On the Importance of Text Analysis for Stock Price Prediction. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland, may 2014. European Language Resources Association (ELRA). 28

Lillian Lee. IDF revisited: a simple new derivation within the Robertson-Spärck Jones probabilistic model. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 751–752, New York, NY, USA, 2007. ACM. 87

Michael E. Lesk. Word-word associations in document retrieval systems. *American Documentation*, 20(1):27–38, 1969. 24, 33

David D. Lewis. Feature Selection and Feature Extraction for Text Categorization. In *Proceedings of the Workshop on Speech and Natural Language*, HLT '91, pages 212–217, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics. 42, 127

David D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 4–15, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE. 2, 11, 48, 50, 51

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1: A New Benchmark Collection for Text Categorization Research. *J. Mach. Learn. Res.*, 5:361–397, December 2004. 28, 128

Hang Li and Kenji Yamanishi. Document classification using a finite mixture model. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, EACL '97, pages 39–47, Stroudsburg, PA, USA, 1997. Association for Computational Linguistics. 33, 48, 57

Hang Li and Kenji Yamanishi. Topic analysis using a finite mixture model. In *In Proceedings of Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 35–44, 2000. 57

Jia Li and Hongyuan Zha. Two-way Poisson mixture models for simultaneous document classification and word clustering. *Comput. Stat. Data Anal.*, 50: 163–180, January 2006. 51

Mu Li, Li Zhou, Zichao Yang, Aaron Li, Fei Xia, David G. Andersen, and Alexander Smola. Parameter Server for Distributed Machine Learning. In *Proc. NIPS workshop on parallel and large-scale machine learning (Big Learning)*, December 2013. 154

Wei Li and Andrew McCallum. Pachinko allocation: DAG-structured mixture models of topic correlations. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 577–584, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. 58

Percy Liang and Dan Klein. Online EM for unsupervised models. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, pages 611–619, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. 42, 55, 73

Elizabeth D. Liddy. Text mining. *American Society for Information Science.Bulletin of the American Society for Information Science*, 27(1):13–14, Oct 2000. 10, 12

G. Lidstone. Note on the general case of the Bayes–Laplace formula for inductive or a posteriori probabilities. *Transactions of the Faculty of Actuaries*, 8:182–192, 1920. 76

Chih-Jen Lin, Ruby C. Weng, and S. Sathiya Keerthi. Trust Region Newton Method for Logistic Regression. *J. Mach. Learn. Res.*, 9:627–650, June 2008. 121

H.-A. Loeliger. An introduction to factor graphs. *Signal Processing Magazine, IEEE*, 21(1):28–41, 2004. 62, 66, 71

Julie B. Lovins. Development of a Stemming Algorithm. *Mechanical Translation and Computational Linguistics*, 11, June 1968. 23

H. P. Luhn. A Business Intelligence System. *IBM J. Res. Dev.*, 2(4):314–319, October 1958. 17

Marco Lui. Feature stacking for sentence classification in evidence-based medicine. *Australasian Language Technology Association Workshop 2012*, page 134, 2012. 24

Marco Lui and Li Wang. Recovering Casing and Punctuation using Conditional Random Fields. In *Australasian Language Technology Workshop*, 2013. 40, 45

Sean Luke. *Essentials of Metaheuristics*. Lulu, version 1.2 edition, 2009. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/. 123

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. 127, 128

Przemyslaw Maciolek and Grzegorz Dobrowolski. Cluo: Web-scale text mining system for open source intelligence purposes. *Computer Science*, 14(1), 2013. 17, 34, 35

David J. C. MacKay and Linda C. Bauman Peto. A hierarchical Dirichlet language model. *Natural Language Engineering*, 1:289–308, 9 1995. ISSN 1469-8110. 75, 80

David Madigan, Alexander Genkin, David D. Lewis, Er Genkin David D. Lewis, Shlomo Argamon, Dmitriy Fradkin, Li Ye, and David D. Lewis Consulting. Author Identification on the Large Scale. In *In Proc. of the Meeting of the Classification Society of North America*, 2005. 24

Rasmus E. Madsen, David Kauchak, and Charles Elkan. Modeling word burstiness using the Dirichlet distribution. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 545–552, New York, NY, USA, 2005. ACM. 53

Christopher D. Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, Cambridge, MA, USA, 1999. ix, 14, 81, 86

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521865719, 9780521865715. 13, 14, 32, 42, 69, 100, 102, 119, 122, 145, 152

A. Markov. Extension of the Limit Theorems of Probability Theory to a Sum of Variables Connected in a Chain. In R. Howard, editor, *Dynamic Probabilistic Systems (Volume I: Markov Models)*, chapter Appendix B, pages 552–577. John Wiley & Sons, Inc., New York City, 1971. 59

M. E. Maron. Automatic Indexing: An Experimental Inquiry. *J. ACM*, 8: 404–417, July 1961. 13, 28, 29, 32, 48, 51

Aleix M. Martínez and Jordi Virtriá. Learning mixture models using a genetic version of the em algorithm. *Pattern Recognition Letters*, 21(9):759–769, 2000. 73

J. Matyas. Random Optimization. *Automation and Remote Control*, 26:244–251, 1965. 123

A. McCallum. Multi-label text classification with a mixture model trained by EM. In *Proceedings of the AAAI' 99 Workshop on Text Learning*, 1999. 49, 53, 57

A. McCallum and K. Nigam. Text Classification by Bootstrapping with Keywords, EM and Shrinkage. In *ACL-99 Workshop for Unsupervised Learning in Natural Language Processing*, pages 52–58, 1999. 102

Andrew McCallum and Wei Li. Early Results for Named Entity Recognition with Conditional Random Fields, Feature Induction and Web-enhanced Lexicons. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 188–191, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. 28

Andrew McCallum and Kamal Nigam. A comparison of event models for Naive Bayes text classification. In *AAAI-98 workshop on learning for text*

categorization, pages 41–48. AAAI Press, 1998. 48, 51, 52, 56, 95, 96, 97, 127

Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient Clustering of High-dimensional Data Sets with Application to Reference Matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 169–178, New York, NY, USA, 2000. ACM. 45

Andrew McCallum, Karl Schultz, and Sameer Singh. Factorie: Probabilistic programming via imperatively defined factor graphs. In *In Advances in Neural Information Processing Systems 22*, pages 1249–1257, 2009. 66, 154

H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. Ad Click Prediction: A View from the Trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1222–1230, New York, NY, USA, 2013. ACM. 154

B. Medlock. An Adaptive, Semi-Structured Language Model Approach to Spam Filtering on a New Corpus. In *CEAS 2006 - Third Conference on Email and Anti-Spam*, July 2006. 28, 59

Eneldo Loza Mencía and Johannes Fürnkranz. Efficient Multilabel Classification Algorithms for Large-Scale Problems in the Legal Domain. In Enrico Francesconi, Simonetta Montemagni, Wim Peters, and Daniela Tiscornia, editors, *Semantic Processing of Legal Texts – Where the Language of Law Meets the Law of Language*, volume 6036 of *Lecture Notes in Artificial Intelligence*, pages 192–215. Springer-Verlag, 1 edition, May 2010. 127, 128

Dieter Merkl. Text Data Mining. In *In A Handbook of Natural Language Processing: Techniques and Applications for the Processing of Language as Text*, pages 269–276. Marcel Dekker, 1998. 10

Donald Metzler and Bruce W. Croft. Linear Feature-based Models for Information Retrieval. *Inf. Retr.*, 10(3):257–274, June 2007. 28, 29, 30

Donald Metzler and W. Bruce Croft. A Markov Random Field Model for Term Dependencies. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 472–479, New York, NY, USA, 2005. ACM. 98

Donald Metzler, Jasmine Novak, Hang Cui, and Srihari Reddy. Building enriched document representations using aggregated anchor text. In *SIGIR'09*, pages 219–226, 2009. 120

T. Mikolov, S. Kombrink, L. Burget, J.H. Cernocky, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531, May 2011. 154

David R. H. Miller, Tim Leek, and Richard M. Schwartz. A hidden Markov model information retrieval system. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '99, pages 214–221, New York, NY, USA, 1999. ACM. 4, 59, 61, 81, 86, 89

T. Minka, J.M. Winn, J.P. Guiver, and D.A. Knowles. Infer.NET 2.5, 2012. Microsoft Research Cambridge. http://research.microsoft.com/infernet. 154

Thomas Minka and John Lafferty. Expectation-Propagation for the Generative Aspect Model. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, 2002. 57, 58

Tom Minka and John Winn. Gates: A graphical notation for mixture models. Technical report, Microsoft Research Ltd., 2008. 63, 66, 67

Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. 13, 26

A. Moffat. Implementing the PPM data compression scheme. *Communications, IEEE Transactions on*, 38(11):1917–1921, November 1990. 79

Mehryar Mohri. *Weighted Finite-State Transducer Algorithms: An Overview*. Physica-Verlag, 2004. 42

Saedeeh Momtazi. *Advanced Language Modeling for Sentence Retrieval and Classification in Question Answering Systems*. PhD thesis, Computer Science Department, Saarland University, 2010. 77, 78, 88, 89

Saeedeh Momtazi and Dietrich Klakow. Hierarchical Pitman-Yor language model for information retrieval. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 793–794, New York, NY, USA, 2010. ACM. 77, 78

Saeedeh Momtazi, Matthew Lease, and Dietrich Klakow. Effective term weighting for sentence retrieval. In *Proceedings of the 14th European conference on Research and advanced technology for digital libraries*, ECDL'10, pages 482–485, Berlin, Heidelberg, 2010. Springer-Verlag. 86, 88, 136

Stefano Monti and Gregory F. Cooper. A Bayesian network classifier that combines a finite mixture model and a naive bayes model. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, UAI'99, pages 447–456, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. 48, 52, 56

Raymond J. Mooney and R. Bunescu. Mining Knowledge from Text Using Information Extraction. *SIGKDD Explorations (special issue on Text Mining and Natural Language Processing)*, 7(1):3–10, 2005. 11, 14

Kevin Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, Computer Science Division, July 2002. 64

Un Yong Nahm. *Text Mining with Information Extraction*. PhD thesis, 2004. 11, 14

Courtney Napoles, Matthew Gormley, and Benjamin Van Durme. Annotated Gigaword. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*, AKBC-WEKEX '12, pages 95–100, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. 20, 38, 40

Oded Netzer, Ronen Feldman, Jacob Goldenberg, and Moshe Fresko. Mine your own business: Market-structure surveillance through text mining. *Marketing Science*, 31(3):521–543, 2012. 17

Graham Neubig. *Unsupervised Learning of Lexical Information for Language Processing Systems*. PhD thesis, PhD thesis, Kyoto University, 2012. 77, 80

Gnter Neumann and Jakub Piskorski. A shallow text processing core engine. *Computational Intelligence*, 18:451–476, 2002. 45

H. Ney, U. Essen, and R. Kneser. On Structuring Probabilistic Dependencies in Stochastic Language Modelling. *Computer Speech and Language*, 8:1–38, 1994. 59, 77, 78, 89

Kamal Nigam, Andrew Mccallum, and Tom Mitchell. Semi-supervised Text Classification Using EM, year = 2006. In *Semi-Supervised Learning*, pages 33–56. MIT Press. 40, 55, 56, 95, 110

Jana Novovicova and Antonin Malik. Application of Multinomial Mixture Model to Text Classification. In *Pattern Recognition and Image Analysis*, volume 2652 of *Lecture Notes in Computer Science*, pages 646–653. Springer Berlin / Heidelberg, 2003. 48, 56, 95, 110

Daniel Oberhoff, Dominik Endres, MartinA. Giese, and Marina Kolesnik. Gates for Handling Occlusion in Bayesian Models of Images: An Initial Study. In Joscha Bach and Stefan Edelkamp, editors, *KI 2011: Advances in Artificial Intelligence*, volume 7006 of *Lecture Notes in Computer Science*, pages 228–232. Springer Berlin Heidelberg, 2011. 66

J. Olive, C. Christianson, and J. McCary. *Handbook of Natural Language Processing and Machine Translation: DARPA Global Autonomous Language Exploitation*. SpringerLink : Bücher. Springer, 2011. 120

Pentti Paatero and Unto Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994. 155

Bo Pang and Lillian Lee. Opinion Mining and Sentiment Analysis. *Found. Trends Inf. Retr.*, 2(1-2):1–135, January 2008. 17, 36

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs Up?: Sentiment Classification Using Machine Learning Techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 79–86, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. 28

Neil Parikh and Mark Drezde. Graphical models for primarily unsupervised sequence learning. Technical Report MS-CIS-07-18, University of Pennsylvania, Department of Computer and Information Science, 2007. 62, 64

Emanuel Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962. 33, 113

Adam Pauls and Dan Klein. Faster and smaller N-gram language models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 258–267, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. 59

Dmitry Pavlov, Ramnath Balasubramanyan, Byron Dom, Shyam Kapur, and Jignashu Parikh. Document Preprocessing for Naive Bayes Classification and Clustering with Mixture of Multinomials. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data*

Mining, KDD '04, pages 829–834, New York, NY, USA, 2004. ACM. 29, 52, 86

Judea Pearl. Markov and Bayes networks: A comparison of two graphical representations of probabilistic knowledge (CSD. University of California at Los Angeles. Computer Science Department). Technical report, 1986. 62, 71

Karl Pearson. Contributions to the Mathematical Theory of Evolution. *Philosophical Transactions of the Royal Society of London. (A.)*, 185:71–110, 1894. 55

Ekin Pehlivan, Funda Sarican, and Pierre Berthon. Mining messages: Exploring consumer response to consumer- vs. firm-generated ads. *Journal of Consumer Behaviour*, 10(6):313–321, 2011. 17

Fuchun Peng and Dale Schuurmans. Combining Naive Bayes and n-Gram Language Models for Text Classification. In *In 25th European Conference on Information Retrieval Research (ECIR*, pages 335–350. Springer-Verlag, 2003. 59

Aritz Pérez, Pedro Larraòaga, and Iòaki Inza. Bayesian classifiers based on kernel density estimation: Flexible classifiers. *Int. J. Approx. Reasoning*, 50 (2):341–362, February 2009. 113

Franz Pernkopf and Djamel Bouchaffra. Genetic-based EM algorithm for learning gaussian mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1344–1348, 2005. 73

Slav Petrov and Dan Klein. Learning and Inference for Hierarchically Split PCFGs. In *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07, pages 1663–1666. AAAI Press, 2007. 46

Steven T. Piantadosi, Harry Tily, and Edward Gibson. The communicative function of ambiguity in language. *Cognition*, 122(3):280 – 291, 2012. 22

John R. Pierce. Language and Machines: Computers in Translation and Linguistics. Technical Report 1416, National Academy of Sciences/National Research Council, 1966. 14

John M. Pierre. Mining Knowledge from Text Collections Using Automatically Generated Metadata. In *Proceedings of the 4th International Conference on Practical Aspects of Knowledge Management*, PAKM '02, London, UK, UK, 2002. Springer-Verlag. 20

David Pinto, José-Miguel Benedí, and Paolo Rosso. Clustering Narrow-Domain Short Texts by Using the Kullback-Leibler Distance. In *Proceedings of the 8th International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing '07, pages 611–622, Berlin, Heidelberg, 2007. Springer-Verlag. 52

Emily Pitler, Shane Bergsma, Dekang Lin, and Kenneth Church. Using Web-scale N-grams to Improve Base NP Parsing Performance. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 886–894, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. 40

Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '98, pages 275–281, New York, NY, USA, 1998. ACM. 47

Hoifung Poon and Pedro Domingos. Sum-Product Networks: A New Deep Architecture. In *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, pages 337–346, 2011. 154

Martin F. Porter. An algorithm for suffix stripping. *Program: Electronic Library & Information Systems*, 40(3):211–218, 1980. 23, 128

M. J. D. Powell. Direct search algorithms for optimization calculations. *Acta Numerica*, 7:287–336, 1998. 123

Daniel Preotiuc-Pietro and Trevor Cohn. A temporal model of text periodicities using Gaussian Processes. In *EMNLP*, pages 977–988. ACL, 2013. 154

Antti Puurula. Mixture Models for Multi-label Text Classification. In *10th New Zealand Computer Science Research Student Conference*, 2011a. 5

Antti Puurula. Large Scale Text Classification with Multi-label Naive Bayes. *Journal of Measurement Science and Instrumentation*, 2:35–45, 2011b. 5

Antti Puurula. Scalable Text Classification with Sparse Generative Modeling. In *PRICAI2012*, pages 458–469, 2012a. 5, 25, 41

Antti Puurula. Combining Modifications to Multinomial Naive Bayes for Text Classification. In Yuexian Hou, Jian-Yun Nie, Le Sun, Bo Wang, and Peng Zhang, editors, *Information Retrieval Technology*, volume 7675 of *Lecture Notes in Computer Science*, pages 114–125. Springer Berlin Heidelberg, 2012b. 6, 51, 85, 86, 87

Antti Puurula. Cumulative Progress in Language Models for Information Retrieval. In *Australasian Language Technology Workshop*, 2013. 6, 86

Antti Puurula and Albert Bifet. Ensembles of sparse multinomial classifiers for scalable text classification. In *ECML/PKDD - PASCAL Workshop on Large-Scale Hierarchical Classification*, 2012. 6, 34, 38

Antti Puurula and Dirk Compernolle. Dual stream speech recognition using articulatory syllable models. *Int. J. Speech Technol.*, 13(4):219–230, December 2010. 73

Antti Puurula and Sung-Hyon Myaeng. Integrated Instance- and Class-based Generative Modeling for Text Classification. In *Australasian Document Computing Symposium*, 2013. 6, 110, 111

Antti Puurula, Jesse Read, and Albert Bifet. Kaggle LSHTC4 Winning Solution. *CoRR*, abs/1405.0546, 2014. 6, 127, 156

J. R. Quinlan. Induction of Decision Trees. *Mach. Learn.*, 1(1):81–106, March 1986. 34

L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989. 61, 70, 71, 82, 85, 100

Martin Rajman and Romaric Besanon. Text Mining - Knowledge extraction from unstructured textual data. In Alfredo Rizzi, Maurizio Vichi, and Hans-Hermann Bock, editors, *Advances in Data Science and Classification*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 473–480. Springer Berlin Heidelberg, 1998. 10

Martin Rajman, Romaric BESANON, and R. Besancon. Text Mining: Natural Language techniques and Text Mining applications. In *In Proceedings of the 7 th IFIP Working Conference on Database Semantics (DS-7). Chapam*, pages 7–10. Hall, 1997. 10, 37

Parikshit Ram and Alexander G. Gray. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pages 931–939, New York, NY, USA, 2012. ACM. 109, 156

Daniel Ramage, David Hall, Ramesh Nallapati, and Christopher D. Manning. Labeled LDA: a supervised topic model for credit attribution in multi-labeled corpora. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, EMNLP '09,

pages 248–256, Morristown, NJ, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-59-6. 57

Carl Edward Rasmussen. Gaussian Processes in Machine Learning. In Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rtsch, editors, *Advanced Lectures on Machine Learning*, volume 3176, pages 63–71. Springer, 2003. 154

Jason D. Rennie, Lawrence Shih, Jaime Teevan, and David R. Karger. Tackling the poor assumptions of naive bayes text classifiers. In *ICML'03*, pages 616–623, 2003. 2, 48, 50, 51, 86, 87, 136

Jason D. M. Rennie. Improving Multi-class Text Classification with Naive Bayes. Master's thesis, Massachusetts Institute of Technology, 2001. 53, 75, 76

Thomas Richardson and Peter Spirtes. Ancestral graph Markov models. *Annals of Statistics*, 30(4):962–1030, 2002. 66

Loïs Rigouste, Olivier Cappé, and François Yvon. Inference and Evaluation of the Multinomial Mixture Model for Text Clustering. *Inf. Process. Manage.*, 43(5):1260–1280, September 2007. 29, 52

S. Robertson and D. A. Hull. The TREC-9 filtering track final report. pages 25–40, 2001. 126, 127

S. E. Robertson and K. Sparck Jones. Relevance weighting of search terms. *J. Am. Soc. Inf. Sci.*, 27(3):129–146, 1976. 51, 86

S.E. Robertson, S. Walker, S. Jones, M.M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *NIST Special Publication 500-226: Overview of the Third Text REtrieval Conference (TREC-3)*, pages 109–126, 1996. 126

Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for IDF. *Journal of Documentation*, 60:2004, 2004. 86, 87

Stephen Robertson and Hugo Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.*, 3:333–389, April 2009. 122

J. J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The Smart retrieval system - experiments in automatic document processing*, pages 313–323. Englewood Cliffs, NJ: Prentice-Hall, 1971. 29, 32, 122

Frank Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6):386–408, 1958. 34

R. Rosenfeld. Two decades of statistical language modeling: where do we go from here? *Proceedings of the IEEE*, 88(8):1270–1278, 2000. 40, 79

Andrey Rzhetsky, Michael Seringhaus, and Mark Gerstein. Seeking a New Biology through Text Mining. *Cell*, 134(1):9–13, July 2008. 17

Kenji Sagae, Yusuke Miyao, and Jun'ichi Tsujii. HPSG Parsing with Shallow Dependency Constraints. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 624–631, Prague, Czech Republic, June 2007. Association for Computational Linguistics. 45

G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Commun. ACM*, 18(11):613–620, November 1975. 23, 29

Gerard Salton. Associative Document Retrieval Techniques Using Bibliographic Information. *J. ACM*, 10(4):440–457, October 1963. 23, 44

Gerard Salton. Syntactic Approaches To Automatic Book Indexing, 1988. 24

Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, August 1988. 24, 86

Mark Sanderson and Justin Zobel. Information Retrieval System Evaluation: Effort, Sensitivity, and Reliability. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 162–169, New York, NY, USA, 2005. ACM. 125

Sunita Sarawagi. Information Extraction. *Found. Trends databases*, 1(3):261–377, March 2008. 14, 15

Issei Sato and Hiroshi Nakagawa. Knowledge discovery of multiple-topic document using parametric mixture model with dirichlet prior. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 590–598, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-609-7. 57

Karl-Michael Schneider. Techniques for Improving the Performance of Naive Bayes for Text Classification. In *In Proceedings of CICLing 2005*, pages 682–693, 2005. 48, 51, 52, 53, 88

Susan Schreibman, Ray Siemens, and John Unsworth. *A Companion to Digital Humanities*. Wiley Publishing, 2008. 15, 17

Hinrich Schütze. Integrating History-length Interpolation and Classes in Language Modeling. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 1516–1525, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-932432-87-9. 59, 77

Fabrizio Sebastiani. Machine Learning in Automated Text Categorization. *ACM Comput. Surv.*, 34(1):1–47, March 2002. 5, 13, 14, 24

Vaughan R. Shanks, Hugh E. Williams, and Adam Cannane. Indexing for fast categorisation. In *Proceedings of the 26th Australasian computer science conference - Volume 16*, ACSC '03, pages 119–127, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc. 5, 25, 100

Claude E. Shannon. *A Mathematical Theory of Communication*. CSLI Publications, 1948. 59

Ian Shemilt, Antonia Simon, Gareth J. Hollands, Theresa M. Marteau, David Ogilvie, Alison O'Mara-Eves, Michael P. Kelly, and James Thomas. Pinpointing needles in giant haystacks: use of text mining to reduce impractical screening workload in extremely large scoping reviews. *Research Synthesis Methods*, 2013. 17

V. Siivola, T. Hirsimaki, and S. Virpioja. On Growing and Pruning KneserNey Smoothed N-Gram Models. *Trans. Audio, Speech and Lang. Proc.*, 15(5): 1617–1624, July 2007. ISSN 1558-7916. 59

Vesa Siivola and Bryan L. Pellom. Growing an n-gram language model. In *INTERSPEECH*, pages 1309–1312. ISCA, 2005. 79, 80

Joseph Sill, Gábor Takács, Lester Mackey, and David Lin. Feature-Weighted Linear Stacking. *CoRR*, abs/0911.0460, 2009. 34

Sameer Singh, Amarnag Subramanya, Fernando Pereira, and Andrew McCallum. Large-scale Cross-document Coreference Using Distributed Inference and Hierarchical Models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 793–803, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. 46

Amit Singhal, Chris Buckley, and Mandar Mitra. Pivoted document length normalization. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '96, pages 21–29, New York, NY, USA, 1996. ACM. 87

Amit Singhal, John Choi, Donald Hindle, David D. Lewis, and Fernando C. N. Pereira. AT&T at TREC-7. In *TREC*, pages 186–198, 1998. 87

Gleb Skobeltsyn, Flavio Junqueira, Vassilis Plachouras, and Ricardo Baeza-Yates. ResIn: A Combination of Results Caching and Index Pruning for High-performance Web Search Engines. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 131–138, New York, NY, USA, 2008. ACM. 42

Mark D. Smucker and James Allan. Lightening the load of document smoothing for better language modeling retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 699–700, New York, NY, USA, 2006. ACM. 86, 88, 136

Mark D. Smucker and James Allan. An Investigation of Dirichlet Prior Smoothings Performance Advantage. Technical report, Department of Computer Science, University of Massachusetts, Amherst, 2007. 53, 76, 77, 80

Mark D. Smucker, James Allan, and Ben Carterette. A Comparison of Statistical Significance Tests for Information Retrieval Evaluation. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 623–632, New York, NY, USA, 2007. ACM. 125

Mark D. Smucker, James Allan, and Ben Carterette. Agreement Among Statistical Significance Tests for Information Retrieval Evaluation at Varying Sample Sizes. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 630–631, New York, NY, USA, 2009. ACM. 125

Fei Song and W. Bruce Croft. A general language model for information retrieval. In *Proceedings of the eighth international conference on Information and knowledge management*, CIKM '99, pages 316–321, New York, NY, USA, 1999. ACM. 59

D. F. Specht. Probabilistic neural networks for classification, mapping, or associative memory. In *Procedings of IEEE International Conference On Neural Networks*, pages 525–532, 1988. 113

Tadej Stajner, Delia Rusu, Lorand Dali, Blaz Fortuna, Dunja Mladenic, and Marko Grobelnik. A Service Oriented Framework for Natural Language Text Enrichment. *Informatica (Slovenia)*, 34(3):307–313, 2010. 45

Anna Stavrianou, Periklis Andritsos, and Nicolas Nicoloyannis. Overview and Semantic Issues of Text Mining. *SIGMOD Rec.*, 36(3):23–34, September 2007. 4, 11, 14

Andreas Stolcke. SRILM – an extensible language modeling toolkit. In *Proceedings of ICSLP*, volume 2, pages 901–904, Denver, USA, 2002. 89

Trevor Strohman, Howard Turtle, and W. Bruce Croft. Optimization strategies for complex queries. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 219–225, New York, NY, USA, 2005. ACM. 108

Jiang Su, Jelber S. Shirab, and Stan Matwin. Large Scale Text Classification using Semi-supervised Multinomial Naive Bayes. In *ICML*, 2011. 40

Charles Sutton and Andrew McCallum. *Introduction to Conditional Random Fields for Relational Learning*, chapter 4. MIT Press, 2007. 50, 62, 64

Jun Suzuki, Akinori Fujino, and Hideki Isozaki. Semi-supervised structured output learning based on a hybrid generative and discriminative approach. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 791–800, Prague, Czech Republic, June 2007. Association for Computational Linguistics. 98

Don R. Swanson. Historical Note: Information Retrieval and the Future of an Illusion. *Journal of the American Society for Information Science*, 39(2): 92–98, 1988. 15

Don R. Swanson. Complementary Structures in Disjoint Science Literatures. In *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '91, pages 280–289, New York, NY, USA, 1991. ACM. 17

Valentin Tablan, Ian Roberts, Hamish Cunningham, and Kalina Bontcheva. GATECloud.net: a platform for large-scale, open-source text processing on the cloud. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1983), 2013. 43

Yik-Cheung Tam and Tanja Schultz. Correlated Bigram LSA for Unsupervised Language Model Adaptation. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *NIPS*, pages 1633–1640. Curran Associates, Inc., 2008. 88

Ah-Hwee Tan. Text Mining: The state of the art and the challenges. In *In Proceedings of the PAKDD 1999 Workshop on Knowledge Disocovery from Advanced Databases*, pages 65–70, 1999. 10, 18, 22

Niket Tandon and Gerard de Melo. Information Extraction from Web-Scale N-Gram Data. In Chengxiang Zhai, David Yarowsky, Evelyne Viegas, Kuansan Wang, and Stephan Vogel, editors, *Web N-gram Workshop. Workshop of the 33rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, volume 5803, pages 8–15. ACM, 2010. 45

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin Markov networks. In *NIPS*. MIT Press, 2003. 34

Yee Whye Teh. A Hierarchical Bayesian Language Model Based on Pitman-Yor Processes. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, ACL-44, pages 985–992, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. 79, 80

D. Tkach. Text mining technology: Turning information into knowledge. *IBM White paper*, 1997. 10

Kristina Toutanova, Francine Chen, Kris Popat, and Thomas Hofmann. Text classification in a hierarchical mixture model for small training sets. In *Proceedings of the tenth international conference on Information and knowledge management*, CIKM '01, pages 105–113, New York, NY, USA, 2001. ACM. 48

Andrew Trotman, Antti Puurula, and Blake Burgess. Improvements to BM25 and Language Models Examined. In J. Shane Culpepper, Laurence A. F. Park, and Guido Zuccon, editors, *Proceedings of the 2014 Australasian Document Computing Symposium, ADCS 2014, Melbourne, VIC, Australia, November 27-28, 2014*, page 58. ACM, 2014. 6

Grigorios Tsoumakas, Apostolos Papadopoulos, Weining Qian, Stavros Vologiannidis, Alexander D'yakonov, Antti Puurula, Jesse Read, Jan Švec, and Stanislav Semenov. WISE 2014 challenge: Multi-label classification of print media articles to topics. 6, 156

Grigorios Tsoumakas, Ioannis Katakis, and Ioannis P. Vlahavas. Mining Multi-label Data. In *Data Mining and Knowledge Discovery Handbook*, pages 667–685. 2010. 27, 42, 45, 118

Grigorios Tsoumakas, Manos Laliotis, Nikos Markantonatos, and Ioannis P. Vlahavas. Large-Scale Semantic Indexing of Biomedical Publications. In *BioASQ@CLEF*, 2013. 24, 43

Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. Stochastic Gradient Descent Training for L1-regularized Log-linear Models with Cumulative Penalty. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 477–485, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. 33

Naonori Ueda and Kazumi Saito. Parametric mixture models for multi-labeled text. In *In Advances in Neural Information Processing Systems 15*, pages 721–728. MIT Press, 2002a. 57

Naonori Ueda and Kazumi Saito. Single-shot detection of multiple categories of text using parametric mixture models. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 626–631, New York, NY, USA, 2002b. ACM. 57

Sofie Van Landeghem, Jari Björne, Chih-Hsuan Wei, Kai Hakala, Sampo Pyysalo, Sophia Ananiadou, Hung-Yu Kao, Zhiyong Lu, Tapio Salakoski, Yves Van de Peer, and Filip Ginter. Large-scale event extraction from literature with multi-level gene normalization. *PLoS One*, 8, 2013. 17

David Vilar, Hermann Ney, Alfons Juan, and Enrique Vidal. Effect of Feature Smoothing Methods in Text Classification Tasks. In *PRIS*, pages 108–117, 2004. 53, 79, 88

Jorge J. Villalón and Rafael A. Calvo. A Decoupled Architecture for Scalability in Text Mining Applications. *J. UCS*, 19(3):406–427, 2013. 34, 35, 43

A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, pages 260–269, 1967. 68, 71

Ellen M. Voorhees and Donna Harman. Overview of the Eighth Text REtrieval Conference (TREC-8). In *TREC*, 1999. 14, 126

Radim Řehůřek. *Scalability of Semantic Analysis in Natural Language Processing*. PhD thesis, Masaryk University, May 2011. 41, 42, 43

Artur Šilić and Bojana Dalbelo Bašić. Visualization of Text Streams: A Survey. In *Proceedings of the 14th International Conference on Knowledge-based and*

Intelligent Information and Engineering Systems: Part II, KES'10, pages 31–43, Berlin, Heidelberg, 2010. Springer-Verlag. 35

Hongning Wang, Minlie Huang, and Xiaoyan Zhu. A Generative Probabilistic Model for Multi-label Classification. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 628–637, Washington, DC, USA, 2008. IEEE Computer Society. 57

Hongning Wang, Yue Lu, and Chengxiang Zhai. Latent Aspect Rating Analysis on Review Text Data: A Rating Regression Approach. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 783–792, New York, NY, USA, 2010a. ACM. 28

Kuansan Wang, Xiaolong Li, and Jianfeng Gao. Multi-style language model for web scale information retrieval. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 467–474, New York, NY, USA, 2010b. ACM. 49

Lidan Wang, Jimmy Lin, and Donald Metzler. A Cascade Ranking Model for Efficient Ranked Retrieval. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 105–114, New York, NY, USA, 2011. ACM. 45

Xuerui Wang, Andrew McCallum, and Xing Wei. Topical N-Grams: Phrase and Topic Discovery, with an Application to Information Retrieval. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, ICDM '07, pages 697–702, Washington, DC, USA, 2007. IEEE Computer Society. 64

Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Tie-Yan Liu, and Wei Chen. A Theoretical Analysis of NDCG Type Ranking Measures. *CoRR*, 2013. 119

Taro Watanabe, Hajime Tsukada, and Hideki Isozaki. A succinct N-gram language model. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, ACLShort '09, pages 341–344, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. 59

Sholom M. Weiss, Nitin Indurkhya, and Tong Zhang. *Fundamentals of Predictive Text Mining*. Springer London Ltd, England, 2012. 4, 11, 15

R. C. White. A survey of random methods for parameter optimization. *Simulation*, 17:197–205, 1971. 123

B. Widrow. An adaptive 'Adaline' neuron using chemical 'memistors'. Technical Report 1553-2, Stanford University, 1960. 34

Pascal Wiggers and L.J.M. Rothkrantz. Topic-based language modeling with dynamic bayesian networks. In *Proceedings of the Ninth International Conference on Spoken Language Processing (Interspeech 2006 - ICSLP)*, pages 1866–1869, sep 2006. 64, 65

John Winn. Causality with Gates. In *Proceedings Artificial Intelligence and Statistics*. The Society for Artificial Intelligence and Statistics, April 2012. 67

Ian Witten. *Text Mining*. Chapman Hall & CRC Press, 2004. 4, 10, 11, 15

Ian H. Witten. Applications of Lossless Compression in Adaptive Text Mining, 2000a. 10

Ian H. Witten. Adaptive Text Mining: Inferring Structure from Sequences. *J. Discrete Algorithms*, 2:137–159, 2000b. 10

Ian H. Witten, Timothy C. Bell, and Alistair Moffat. *Managing Gigabytes: Compressing and Indexing Documents and Images*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. 25, 26

Ian H. Witten, Zane Bray, Malika Mahoui, and W. J. Teahan. Text Mining: A New Frontier for Lossless Compression. In *Data Compression Conference*, pages 198–207. IEEE Computer Society, 1999. 10

Rui Xia, Chengqing Zong, and Shoushan Li. Ensemble of Feature Sets and Classification Algorithms for Sentiment Classification. *Inf. Sci.*, 181(6): 1138–1152, March 2011. 24

Jinxi Xu and Ralph Weischedel. Cross-lingual information retrieval using hidden Markov models. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13*, EMNLP '00, pages 95–103, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics. 4, 81

Yi Yang and Jacob Eisenstein. A Log-Linear Model for Unsupervised Text Normalization. *Proceedings of the Empirical Methods on Natural Language Processing (EMNLP)*, pages 61–72, 2013. 22

Yiming Yang. Expert network: effective and efficient learning from human decisions in text categorization and retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '94, pages 13–22, New York, NY, USA, 1994. Springer-Verlag New York, Inc. 5, 25, 100

Yiming Yang and Jan O. Pedersen. A Comparative Study on Feature Selection in Text Categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. 42

Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. Recent Advances of Large-Scale Linear Classification. *Proceedings of the IEEE*, 100(9):2584–2603, 2012. 33

Hugo Zaragoza, Henning Rode, Peter Mika, Jordi Atserias, Massimiliano Ciaramita, and Giuseppe Attardi. Ranking Very Many Typed Entities on Wikipedia. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 1015–1018, New York, NY, USA, 2007. ACM. 120

ChengXiang Zhai. Statistical Language Models for Information Retrieval A Critical Review. *Found. Trends Inf. Retr.*, 2(3):137–213, March 2008. 86

Chengxiang Zhai and John Lafferty. The Dual Role of Smoothing in the Language Modeling Approach. In *Proceedings of the Workshop on Language Models for Information Retrieval (LMIR) 2001*, pages 31–36, 2001a. 47, 77, 86

Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to Ad Hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '01, pages 334–342, New York, NY, USA, 2001b. ACM. 5, 77, 80, 100

Chengxiang Zhai and John Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *Proceedings of the tenth international conference on Information and knowledge management*, CIKM '01, pages 403–410, New York, NY, USA, 2001c. ACM. 88

Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22 (2):179–214, April 2004. 77, 80

Hui Zhang and David Chiang. Kneser-Ney smoothing on expected counts. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, ACL '14. Association for Computational Linguistics, 2014. 4, 78, 79, 80, 85, 88, 89, 90, 91

Nevin Zhang and David Poole. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, pages 171–178, 1994. 42, 71

Yi Zhang and Wei Xu. Fast exact maximum likelihood estimation for mixture of language model. *Inf. Process. Manage.*, 44(3):1076–1085, 2008. 85

Yi Zhang, Jamie Callan, and Thomas Minka. Novelty and redundancy detection in adaptive filtering. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, pages 81–88, New York, NY, USA, 2002. ACM. 102

H. Zhao and Q. Liu. The CIPS-SIGHAN CLP 2010 Chinese word segmentation bakeoff. In *Proceedings of CIPS-SIGHAN Joint Conference on Chinese Language Processing (CLP10)*, pages 199–209, 2010. 23

Shi Zhong and Joydeep Ghosh. Generative model-based document clustering: a comparative study. *Knowl. Inf. Syst.*, 8(3):374–384, September 2005. 29, 47, 55

Xuezhong Zhou, Yonghong Peng, and Baoyan Liu. Text mining for traditional Chinese medical knowledge discovery: a survey. *Journal of biomedical informatics*, 43(4):650–660, August 2010. 17

Conghui Zhu, Jie Tang, Hang Li, Hwee T. Ng, and Tiejun Zhao. A Unified Tagging Approach to Text Normalization. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 688–695, Prague, Czech Republic, June 2007. Association for Computational Linguistics. 22

Imed Zitouni and Qiru Zhou. Hierarchical linear discounting class N-gram language models: A multilevel class hierarchy approach. In *ICASSP*, pages 4917–4920. IEEE, 2008. 102

Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2), July 2006. 25, 26, 46, 101, 156