Working Paper Series
ISSN 1170-487X

Predicting Library of Congress Classifications from Library of Congress Subject Headings

Eibe Frank and Gordon Paynter

Working Paper: 01/03
January 2003

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Predicting Library of Congress Classifications From Library of Congress Subject Headings

Eibe Frank
Department of Computer Science
University of Waikato
eibe@cs.waikato.ac.nz

Gordon W. Paynter
The INFOMINE Project, Science Library
University of California, Riverside
gordon.paynter@ucr.edu

Abstract

This paper addresses the problem of automatically assigning a Library of Congress Classification (LCC) to a work given its set of Library of Congress Subject Headings (LCSH). LCC are organized in a tree: the root node of this hierarchy comprises all possible topics, and leaf nodes correspond to the most specialized topic areas defined. We describe a procedure that, given a resource identified by its LCSH, automatically places that resource in the LCC hierarchy. The procedure uses machine learning techniques and training data from a large library catalog to learn a model which maps from sets of LCSH to classifications from the LCC tree. We present empirical results for our technique showing its accuracy on an independent collection of 50,000 LCSH/LCC pairs.

1 Introduction

The Library of Congress Classification (LCC) is a hierarchical set of topic descriptors used to categorize the intellectual content of a work, to situate the work relative to others in the tree of knowledge, and (more prosaically) to place books on shelves. Because LCC[1] are media-independent, they can be assigned to resources in digital and virtual libraries, providing compatibility with traditional resources and an access method familiar to librarians from the United States and around the world.

INFOMINE (http://infomine.ucr.edu/) is a virtual library of over 20,000 scholarly Internet resources maintained cooperatively by and for librarians. Each record was manually created by a librarian, and describes a resource with standard library cataloging techniques,

[1] Throughout this document we use the acronym LCC to refer both to the classification framework as a whole, and to specific classifications within the framework.

including controlled terms from the Library of Congress Subject Headings (LCSH). The INFOMINE Project requires a hierarchical classification for each resource, but in a collection this large (and growing) it is logistically impossible to assign such metadata manually.

This lack defines our problem: we wish to automatically assign a hierarchical classification to each INFOMINE record based on its extant metadata. Specifically, we will learn to assign a classification from the LCC Outline to a resource based on a set of LCSH that describe that resource. The solution we describe uses machine learning techniques and training data from an academic library catalog to learn a classification model that maps from sets of LCSH to nodes in the LCC tree.

The problem is complicated by the large number of potential classifications: most machine learning problems deal with hundreds of classes at most, but there are thousands of LCC. For this reason, prior work treats LCC classification as an information retrieval task: virtual documents are created representing each class, and new examples are classified by using similarity measures to find the most similar "documents" (Larson, 1991; Dolin, 1998; Thompson et *al.*, 1997). The hierarchical nature of the LCC is largely ignored.

Our solution addresses the problem by exploiting its hierarchical nature. A pairwise linear classifier is learned for every node in the LCC hierarchy that classifies an example as belonging to that node or belonging to one of its child nodes. Each new record is first classified by the root node classifier into one of the 21 top-level LCC nodes. The classifier at that node is then used to classify it into a child node. This process repeats until a leaf node is reached or an internal node classifier chooses itself ahead of any of its children.

We provide an implementation that assigns classifications from the LCC Outline. The LCC Outline is a hierarchy of over four thousand nodes from the top of the LCC tree that summarizes the full topic space. The Outline is only part of the LCC; we chose it because it offers sufficient detail, is electronically available, and has been used in earlier studies (Dolin, 1998). The implementation described is extensible to larger subsets of the LCC, and to other hierarchical classifications such as the Dewey Decimal Classification (DDC).

The hierarchical classifier is evaluated by training it on up to 800,000 records from the library catalog of the University of California at Riverside, and testing it on a further set of 50,000 records. The classification accuracy is 55% when the classifier is trained on the full training set, though this measure ignores "partial successes" like records that are assigned a classification that is too general or too specific, so we report additional statistics that measure partial success. We also extend the method to output multiple LCC classifications with corresponding confidence scores and evaluate the resulting ranked lists of predictions.

The paper is arranged as follows. Section 2 summarizes the INFOMINE requirement that motivated this work, and reviews related work on inferring LCC classifications from other data. Section 3 explains how machine learning techniques are used to build a hierarchical classifier for predicting LCC from LCSH. This method is evaluated in Section 4. The final sections discuss the method and evaluation, compares the results to the previous studies, and summarizes the work presented in this paper.

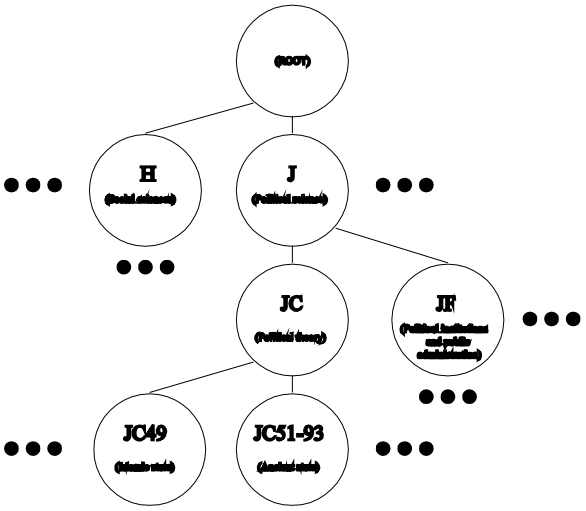

Figure 1: A (very small) part of the LCC tree

2 Background

The INFOMINE Project (http://infomine.ucr.edu/) maintains an continuously expanding database of over 20,000 scholarly Internet resources; each has LCSH metadata but lacks LCC metadata, a lack this research aims to remedy. This section provides background on the concepts and terminology used in the paper, describes the classification requirements of the INFOMINE Project, and reviews the previous work in this area.

2.1 Library of Congress Classifications and Subject Headings

Both the LCC and LCSH are sets of topic descriptors created and maintained by the Library of Congress (Library of Congress,1986-2001; Library of Congress Subject Cataloging Division, 2001). The LCC are arranged hierarchically and are typically assigned singly, while the LCSH have little hierarchical structure but are assigned in sets to each resource. There are approximately 100,000 LCC and 257,000 LCSH (Library of Congress Subject Cataloging Division, 2001) but the exact number is indeterminate as catalogers can *establish by pattern* descriptors pertaining to new authors, entities, and geographic regions.

The LCC are divided into 21 *Schedules*, or top-level classifications. A specific descriptor is identified by a key like *DA25.5*. This example identifies a work in Schedule D (i.e. *History: General and Old World*), in the subclass *DA* (i.e. *DA1-995 History of Great Britain*), in the interval *DA20-690* (*England*) and the narrower interval *DA20-27.5* (*General*). The LCC are organized in a strict hierarchy where the root node encompasses all possible topics, and leaf nodes correspond to the most specialized topic areas. LCC are

frequently used to assign "call numbers" to books for shelving purposes, consequently most resources are assigned only one LCC. They are typically extended with cutter numbers to expand the classification and create a unique call number. We are not interested in this level of detail, however, and will focus on the *LCC Outline*: 4214 classifications selected from the top levels of all 21 schedules by the Library of Congress for use by catalogers (Library of Congress, 1990). Figure 1 shows part of the LCC Outline. Classifications are shown as large circles, with child nodes connected to their parents with lines. Ellipses indicate the parts of the hierarchy that are not shown. For example, *JC51-93 Ancient state* is a leaf node, and is a child of *JC Political theory*.

LCSH descriptors consist of English words or phrases like *HISTORY* or *BOTANY*. The specification includes some hierarchical structure, but the hierarchy is incomplete so we treat the LCSH as flat. However, the descriptors can be refined through the use of subdivisions: for example *BOTANY - PUERTO RICO - NOMENCLATURE* describes a book about the naming of plants in Puerto Rico. The first descriptor is referred to as the subject heading, and the latter are *topical*, *form*, *chronological*, *geographical* or *free-floating* subdivisions.

There is an explicit correlation between the two schemes: approximately 36% of LCSH headings have an associated LCC (Library of Congress Subject Cataloging Division, 2001). In most library catalogs where they are employed, a single LCC is assigned in conjunction with several LCSH. The LC rules of classification state that the LCC is based on the first LCSH, but in practice this rule is often ignored (Larson, 1992). Consequently there is no comprehensive crosswalk from LCSH to LCC.

2.2 The classification goals of the INFOMINE Project

Table 1 shows a typical INFOMINE record: as for all INFOMINE records, a librarian has assigned it a broad INFOMINE category and a set of LCSH, but no hierarchical classification. As the INFOMINE collection has grown, it has become difficult to browse with non-hierarchical structures: for example, browsing a list of thousands of keywords alphabetically is awkward and unrewarding, and further growth will exacerbate the problem. A more scalable approach is to let the user navigate a topic-based hierarchy from broad top-level subject areas down to specialized topics and the documents pertaining to them. To implement this solution, the collection requires hierarchical subject-area metadata. Specifically, each INFOMINE record will be classified into the LCC Outline.

The LCC Outline has many advantages to recommend it. Although many categorization schemes have been proposed for Web resources, most are "lacking the rigorous hierarchical structure and careful conceptual structure found in established schemes" like the LCC and DDC (Chan, 2001). The LCC hierarchy, on the other hand, is maintained by the Library of Congress, and refined as new topic areas are discovered; consequently it has high-quality descriptors and great depth of coverage, and carries institutional authority. Additionally, it is used in most US research libraries, as well as the Library of Congress, so is well known to most librarians, and example classifications are easily found.

Field	Value
Record	26396
Title	AmphibiaWeb
URL	http://elib.cs.berkeley.edu/aw/index.html
Category	Biological, Agricultural and Medical Sciences
LCSH	Amphibians; Amphibians - Ecology; Amphibians - Identification; Amphibians - Taxonomy; Herpetology
Keywords	Amphibia; Databases; Distribution map; Frogs; Images; Nature conservation; Reference resources; Salamanders; Searchable; Wildlife conservation
Authors	Wake, David; Museum of Vertebrate Zoology; University of California, Berkeley;
Description	“AmphibiaWeb, a site inspired by global amphibian declines, is an online system that allows free access to information on amphibian biology and conservation. AmphibiaWeb offers ready access to taxonomic information for every recognized species of amphibian in the world. Species descriptions, life history information, conservation status, literature references, photos and range maps are available for many species and are being added to regularly by specialists and volunteers from around the world. In addition, AmphibiaWeb provides easy and fast access to museum specimen data from large herpetological collections. We hope AmphibiaWeb will encourage a shared vision for the study of amphibian declines and the conservation of remaining amphibians.”
Created	2001-10-11 by SFM
Last modified	2001-10-11 by SFM

Table 1: An INFOMINE record from the “Biological, Agricultural and Medical Sciences” category

2.3 Related work: Automatically assigning LCC

Previous examples of automatic LCC assignment differ from the present work in that they are based on information retrieval techniques, that they attempt to assign LCC to new documents using other types of metadata, and in the extent to which the hierarchical nature of the LCC is exploited. Like the present work, all rely on large training datasets drawn from existing catalogs and can be adapted to other hierarchies.

The earliest comparable research is Larson's study of the automatic selection of LCC for Schedule Z (*Bibliography and Library Science*) (Larson, 1992). Larson's approach is to create 8,435 clusters of documents with similar LCC, then to create virtual documents representing each LCC cluster using *Title* and *LCSH* metadata from 30,000 library catalog records. New documents are classified by extracting their metadata, using an Information Retrieval similarity measure (Salton, 1971) to find the closest virtual document, and assigning that virtual document's LCC to the new document.

Larson considered many parameters in his experiments (60 sets of results are presented). The most interesting parameter is the metadata chosen to create the virtual documents. Of the five combinations tested—*title and all subject headings*, *title and first subject*, *all subject headings*, *first subject only* and *title only*—the third (*all subject headings*) is almost identical to the data considered here. Another parameter is whether complete subject headings should be treated as attributes, or broken down into their constituent words. Of the 60 combinations of parameters tested, the best overall performance (*first subject only*) was correct for 46.6% of 283 new books from Schedule Z. The accuracy when *all subject headings* are considered and treated as terms was approximately 38%.[2] However, the large number of parameter combinations and the small amount of test data suggest these results are optimistic. It is unclear how they generalize to the more realistic setting where all top-level Schedules are considered.

Scorpion is a program developed by the OCLC Project to assign DDC to Web resources and other full-text documents that has been adapted to use the LCC (Thompson et *al.*, 1997; Godby & Stuler, 2001). Like Larson's system, it works by creating virtual documents representing each of the possible classifications and using information retrieval measures to compare new examples to the virtual documents.

Scorpion differs from Larson in two important respects. First, the virtual documents representing possible classifications are not created by clustering similarly classified documents together, instead they are generated from the LCC hierarchy. Starting with the full LCC, those classifications whose textual descriptions contain country names or generic names, or cross-references to other classifications are removed: in experiments with the *Q*, *R*, *S* and *T* schedules 91% of the classifications are eliminated, leaving 6,314 classifications (Godby & Stuler, 2001). Second, virtual documents for each LCC are derived by selecting co-occuring terms from OCLC's World Cat (a database of bibliographic records) and from the Library of Congress Subject Authority (a database of canonical names and terms). Although the virtual documents are based on the hierarchical LCC, they are (as in Larson) treated as flat by the similarity measure when classes are assigned to new documents. No evaluation of Scorpion on independent data is reported.

Pharos is a system for selecting information sources, such as newsgroups, by querying their automatically assigned LCC (Dolin et *al.*, 1998; Dolin, 1998). Underlying the system is a mechanism for automatically classifying documents into a hierarchy in a similar manner to Larson and Scorpion.

Pharos classifies documents into the LCC Outline. Virtual documents are created for each classification using words from the title of the classification and from Title and Subject Heading fields drawn from a library catalog. Unlike Larson and Scorpion, which use the SMART information retrieval system (Salton, 1971), Pharos uses Latent Semantic Indexing (Dumais, 1991) to calculate similarity measures (in principle, any information retrieval system, including SMART, can be used).

Pharos was trained on 1.5 million records from the UCSB library, and evaluated on a

[2] Estimated from Larson (1992), Figure 3.

subset of these records[3] and on a set of newsgroup documents (this experiment was inconclusive) (Dolin, 1998). The former resembles our own task, and classified 7214 records from all parts of the LCC hierarchy to an accuracy of around 14%.

2.4 Related work: Hierarchical classification

In the machine learning literature, Greiner et *al.*'s work (1997) is most closely related to what is presented in this paper. It investigates the effect of using a hierarchy of classes in the context of a document categorization problem with 48 possible classifications. When using pairwise classification in combination with a linear classifier[4] (as we do in this paper), they find that exploiting the class hierarchy is beneficial both in terms of computational complexity and predictive accuracy.

Also closely related is Dumais and Chen's work (2000). They use the support vector machine algorithm to learn linear classifiers (as we do in this paper). However, instead of using pairwise classification to solve the multi-class classification problems that occur in the hierarchy, they employ the one-against-all technique, which is known to be inferior both in terms of computational complexity and in terms of generalization accuracy (Fürnkranz, 2002). They evaluate their method on a hierarchy with 13 top-level categories and 150 second-level categories, and find a significant improvement compared to using a “flat” classifier that is trained on the 150 second-level classes alone. A very similar approach is used by Sun et *al.* (2001, 2002), and evaluated on hierarchies with up to eight categories. D'Alessio et *al.* (1998) use a heuristic linear classifier in conjunction with a hierarchical feature selection scheme in a similar fashion and find an improvement in performance compared to a flat classifier based on experiments with a dataset containing 27 categories.

Ng et *al.* (1997) apply a hierarchical classifier to a text categorization problem with 93 categories but do not take advantage of the hierarchy at training time. Once a linear classifier based on the perceptron algorithm has been built for each category, the resulting set of classifiers is organized into a hierarchy, which is traversed recursively at classification time. Ruiz and Srinivasan (2002) extend this method by incorporating feature and training set selection and non-linear multi-layer perceptrons instead of linear perceptrons, testing it on a problem with 103 categories.

Weigend et *al.* (1999) propose a hierarchical classifier that learns a multi-layer perceptron to distinguish between a set of “meta”-categories and a multi-layer perceptron for each individual category within a meta-category. Only data within a particular meta-category is used to train the corresponding second level classifiers, making use of the hierarchy at training time. The approach is evaluated on a dataset with 92 topics and shown to improve on a flat multi-layer perceptron.

Koller and Sahami (1997) hierarchically also demonstrate that using hierarchical structure

[3] It is not clear that the testing set is distinct from the training set.

[4] Note that they explicitly omit details of the learning algorithm that is used to generate the linear classifiers.

can improve classification performance in the context of two document categorization problems with four and six classes respectively. They use a Bayesian classifier at each node of the classification hierarchy and employ a feature selection method to find a set of discriminating features (i.e. words) for each node. Mladenic and Grobelnik (1999) use this approach to automatically place web documents into several large hierarchies extracted from the real-world Yahoo hierarchy of web sites. Our method does not require a feature selection step because we use the support vector machine learning algorithm (Cristianini & Shaw-Taylor, 2000), which implicitly determines the features that are important for classification (Taira & Haruno, 1999).

McCallum et *al.* (1998) show that a hierarchical class structure can be used to improve the probability estimates of a Bayesian classifier by shrinkage-based smoothing of the estimates obtained along a path through the hierarchy. The EM algorithm is used to find good parameter values for smoothing. Experimental results show that the smoothing procedure improves classification performance in three document categorization problems with 71, 20, and 264 possible classifications respectively (compared to a flat Bayesian classifier that does not exploit the hierarchical structure).

3 Predicting LCC from LCSH

The goal of the present work is to classify resources into the LCC Outline based on the metadata available in INFOMINE records. The problem exhibits two interesting features: it is very sparse, and has a hierarchical class. Our method takes advantage of both properties.

To avoid the costs of manually classifying example records, we will draw examples from library catalogs and use them to train a classification model based on the support vector machine learning algorithm (Cristianini & Shaw-Taylor, 2000). Although both Title and LCSH metadata are available in INFOMINE and library catalog records, we ignore Title metadata because it is not drawn from a controlled vocabulary and in prior work it provides little additional benefit (Larson, 1992).

Our application essentially poses a text categorization problem, where the “text” of the document consists of the LCSH, and the category to predict is the corresponding LCC range. Text categorization problems are generally sparse problems, and our data is particularly sparse because there are few LCSH per document. Support vector machines are the state-of-the-art machine learning method for sparse classification, and they are known to be more accurate than other machine learning and information retrieval techniques on text categorization problems (Joachims, 1998; Dumais, 1998).

3.1 Preparing the data

The classifier is trained on example records that have both an LCC and a set of LCSH assigned. Most research library catalogs in the United States incorporate this metadata and can be exported in Machine-Readable Cataloging (MARC) format (Library of Congress,

1999).

In our experiments, LCC data is extracted from MARC fields 050 or 090 (subfield $a), and LCSH data from fields 650 and 651 (when the second indicator tag has value 0). Only records that have both LCSH and LCC metadata are considered, and both the LCC and LCSH are normalized. Individual LCC are converted to the most specific matching LCC interval from the LCC Outline. This is usually a leaf node, but can be an internal node. LCSH are converted to lowercase, all text appearing in parentheses is removed, and all subdivisions (of all types) are removed. Table 2 shows some examples of normalized LCC and LCSH with the raw versions that appear in the MARC records.

The LCC Outline is available in book form from the Library of Congress (1990), but we require an electronic reproduction. For consistency, we used the same version as the Pharos project; see Dolin (1998) for a description of how it was assembled. This version differs in some ways from the printed editions, from recent electronic versions in the *Classification Web* subscription service[5], and from the on-line version currently (June 2003) provided by the Library of Congress[6], (apparently an update of the 6th printed edition). As the full LCC is continually being improved, all of these versions provide only a snapshot of the top of the LCC tree at a specific time. The work described in this paper is applicable to any of these versions of the Outline, or any other hierarchical, electronically-available classification scheme, so these differences are not material to the results presented below. An electronic copy of the LCC Outline used in this work can be obtained from the Pharos Web site[7], the INFOMINE Web site[8], or by contacting the authors directly.

Propositional machine learning methods require a very restricted form of input consisting of a single table of data. The rows—properly called *instances* or *examples*—correspond to the individual records. In our application each row corresponds to a single MARC record. The columns—called *attributes*—encode the information in each record. In our application, the first attribute corresponds to the normalized LCC. This attribute is called the *class attribute*. The other attributes encode the normalized LCSH set. The classifier will learn to predict the value of the class attribute from the values of the other attributes by performing a statistical analysis of the dependencies in the data.

Each row, or instance, must have the same set of attributes. This is achieved by creating a binary attribute for every LCSH in the training data. The value of an attribute representing a subject heading is "1" if the corresponding MARC record contains this subject heading, and "0" otherwise. Table 3 shows a simple example training dataset with three records and Table 4 shows the converted dataset. Note that in practice the vector of binary attribute values for each instance is extremely sparse because most records are assigned fewer than ten of the thousands of possible LCSH.

[5] http://lcweb.loc.gov/cds/classweb.html
[6] http://lcweb.loc.gov/catdir/cpso/lcco/lcco.html
[7] http://pharos.alexandria.ucsb.edu/data/
[8] http://infoine.ucr.edu/projects/lcc_classification/

Initial form	Normalized LCC
HE559.A5	HE380.8-560
NE642.T9	NE1-978
BJ37	BJ1-219
Initial form	**Normalized LCSH**
Harbors	harbors
Harbors - Great Britain	harbors
Ethics - Methodology	ethics

Table 2: Examples of normalized LCC and LCSH and the raw version from the SCOTTY catalog.

LCC	LCSH
HE380.8-560	harbors; pilot guides
NE1-978	harbors
BJ1-219	judgment; duty; ethics

Table 3: Example training dataset with three records.

LCC	harbors	pilot guides	judgment	duty	ethics
HE380.8-560	1	1	0	0	0
TP1142	1	0	0	0	0
BJ37	0	0	1	1	1

Table 4: Dataset from Table 3 converted into attribute-value format

3.2 Building a hierarchical classifier

The objective of the classifier is to predict the most specific LCC range corresponding to any set of one or more LCSH. There are a large number of possible nodes—4214 in the LCC Outline—and this makes the learning problem computationally complex. Fortunately the tree structure of the LCC Outline allows a hierarchical approach to learning, where the learning task is decomposed into a series of feasible sub-problems.

The key insight is that each node of the LCC tree poses a separate classification problem by itself. At each node we have to decide which branch to follow—or whether to stop at that node—for the set of LCSH that we want to classify. This means that we can apply the learning algorithm at each node of the tree to generate a classifier that will tell us the most likely branch for an instance with an unknown LCC. If there are N possible branches at a node, the corresponding learning problem is an $(N + 1)$-class problem. To train a classifier at a particular node we can extract all the instances in the training data that pertain to that node by looking at their LCC.

By filtering an instance down the tree according to the predictions of the classifiers that it encounters along the way, it will finally arrive at a node that corresponds to the most specific LCC interval that can be assigned to it. This approach to hierarchical classification is also known as the Pachinko Machine (McCallum et *al.*, 1998). A crucial feature of the

problem considered in this paper is that an instance may not be filtered down all the way to a leaf because the classifier at an internal node may decide that it is most appropriate for the instance to stay at that node instead of being propagated further down.

As well as making the learning problem more tractable, the hierarchical approach has the advantage that it naturally captures the structure of the domain. The predictions at the root node of the hierarchy are the most accurate but least specific ones, predictions become less reliable (but more specific) further down the tree. Accurate high-level metadata can be generated by looking at the predictions at the top level of the hierarchy.

3.3 Learning the classification model

Linear support vector machines are used to generate the individual classifiers at each node of the hierarchy because they are known to perform very well in text categorization problems (Joachims, 1998; Dumais, 1998), and are computationally very efficient with sparse data. The data is only used in terms of dot products between pairs of instances and these dot products can be implemented as sparse dot products (Platt, 1999), which effectively means that only "1" values in the data incur a computational cost. Since our data is extremely sparse—even compared to other text categorization problems—linear support vector machines enable us to process large collections of training instances. They also make the classification process for new instances very efficient.

A linear support vector machine is a linear discriminant for separating instances belonging to two classes. (We explain below how it can be applied to multi-class problems.) It differs from standard statistical techniques for finding linear discriminants in that it generates a very special discriminant function: the one that is maximally distant from the two classes involved. This function is called the *maximum margin hyperplane.* Figure 2 shows a maximum margin hyperplane for an artificial learning problem with two classes (because there are only two dimensions the hyperplane is a line). The fact that this particular discriminant is used is a key factor for the excellent generalization performance that support vector machines exhibit in practice. The solution typically depends on only a few training instances—the ones closest to the decision boundary—and these are called "support vectors". Discarding any of the other instances does not change the position of the boundary.

Finding the maximum margin hyperplane is a quadratic optimization problem and the underlying mathematical theory is complex. Fortunately there is a relatively simple algorithm, *Sequential Minimal Optimization*, for solving this optimization task (Platt, 1999). This algorithm is computationally efficient in terms of both running time and memory requirements. We used the implementation from the WEKA machine learning workbench (Witten & Frank, 2000) which is efficient and incorporates other optimizations.

The concept of a maximum margin hyperplane is only applicable if the data is perfectly separable by a linear discriminant—otherwise this hyperplane is not defined. Fortunately the support vector machine algorithm can be extended to a *soft margin classifier* by

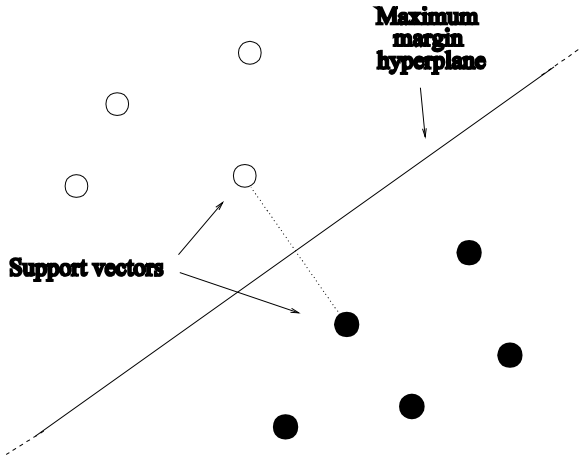

Figure 2: A small two-class dataset with nine instances and the corresponding maximum margin hyperplane

allowing some instances to lie on the "wrong" side of the hyperplane (Cristianini & Shaw-Taylor, 2000). This is achieved by introducing an upper bound on the influence of the support vectors. The larger this upper bound the more closely the classifier can fit the training data. Too large a value may lead to overfitting. In preliminary experiments we tried two settings for this parameter. First we set the upper bound to 1000, allowing the classifier to fit the training data very closely. Then we tried an upper bound of 1. We found that both settings resulted in very similar accuracy. However, using a value of 1 made learning much faster. Consequently we adopted this value for the experimental results reported in Section 4.

The basic support vector machine performs binary (i.e. two-class) classifications; it is not directly applicable to multi-class problems. However, the LCC hierarchy poses multi-class problems at every node. It has been found experimentally that pairwise classification is an efficient and effective method of solving multi-class problems with binary classifiers (Fürnkranz, 2002). In this method a binary classifier is learned to distinguish between each pair of classes using only instances from those two classes. If there are m classes that means $m \times (m - 1)/2$ classifiers will be built. Only the instances pertaining to the two classes involved are used to build a particular classifier, so pairwise classification scales well with the number of classes.

3.4 Making predictions

Once the hierarchical classifier has been built by placing support vector machines at every internal node of the tree, it can be used to classify a new instance for which the LCSH are known but not the LCC. The first step is to convert the data into the appropriate format so that it can be processed by the support vector machine classifiers. To this end the LCSH are matched against the dictionary of headings that has been built up from the training data. If a particular heading from the training data is present in the new record, the corresponding attribute's value is set to "1", otherwise it is set to "0".

The resulting sparse feature vector is used to filter the instance down the classification hierarchy. This process starts by applying the support vector machine classifiers at the tree's root node, where there are 21 possible classifications. A child node is chosen using the procedure for pairwise classification described above: the instance is processed by each of the 210 $(21 \times (21 - 1)/2)$ binary classifiers and the class that receives the most votes in total is predicted. Based on this prediction the instance is propagated down to one of the 21 possible top-level LCC classifications. The classification process is repeated recursively within this category, and increasingly specific classifications are made. The process stops when a leaf node is reached, or when the classifier at an internal node decides that the current node is more likely then any of its children.

3.5 Predicting multiple LCC

A disadvantage of this procedure is that it only returns one path through the LCC hierarchy for a given set of LCSH. However, in practice it may be desirable to obtain a ranked list of possible classifications because many documents cover more than one topic, and in some applications it is preferable to allow for inaccuracies in the first classification.

A ranked list can be established by associating a confidence score with each possible path through the hierarchy in the following way. First we compute a score for every possible prediction for every node of the hierarchy. Assume that there are m possible classes at a node. Let n_i be the number of votes associated with class i, and n_{best} the number of votes for the "winner". Then we assign a confidence of n_i/n_{best} to class i. This means the "best" candidate at a particular node receives a score of one. Then, given these individual confidence scores for every possible classification in the tree, we can calculate a confidence score for every path by computing the product of the individual confidence scores along it.

The top-ranked path will always have a score of one. If there is no tie for first this will be the same LCC as is output by the single-classification method above. In case of ties the LCC classifications are output in the order in which they are encountered in a depth-first search through the tree structure. Ties for first do not occur very frequently because in most cases n_{best} is greater than the number of votes for the next-best class. Using the classifier generated from the largest amount of training data that we considered in our experiments resulted in a tie for 3% of the records in the test data.

3.6 Implementation

Our implementation of the LCC classifier is Free Software distributed under the terms of the GNU General Public License and is available from the INFOMINE Project.[9]

LCSHtoLCC is implemented in Java, and exploits two existing projects: *MARC4J* and *WEKA*. The MARC4J record parser[10] is used to extract LCC and LCSH data from the

[9] http://infomine.ucr.edu/download/

[10] http://marc4j.tigris.org/

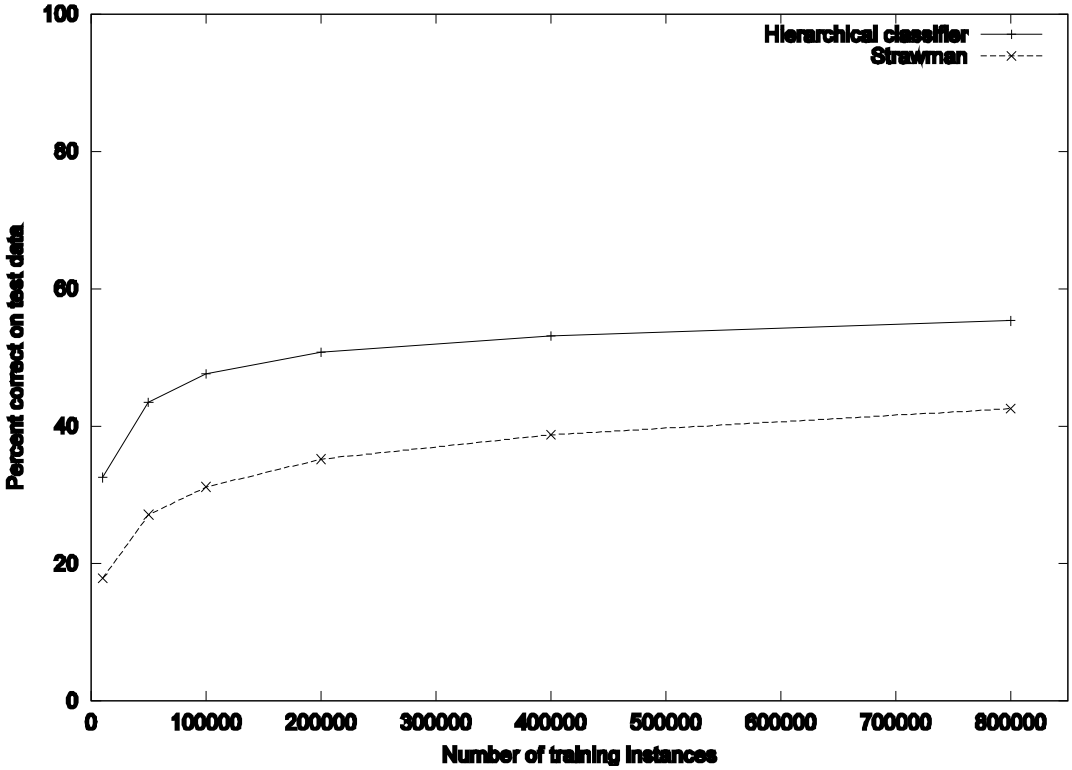

Figure 3: Accuracy on test data for given amount of training data

MARC records. Any library catalog with LCSH and LCC metadata that can be exported as MARC records can be used to train and test LCSHtoLCC. The text processing and learning components are drawn from the WEKA machine learning workbench (Witten & Frank, 2000).[11]

4 Empirical results

This section provides empirical results to evaluate the method described in the previous section. Our dataset is drawn from the library catalog of the University of California at Riverside. It contains 868,836 examples, distributed among the 4214 classifications of the LCC Outline. After shuffling the data randomly, we set the last 50,000 instances aside for testing purposes, and trained the classifier on six separate training datasets, comprised of the first 10,000; 50,000; 100,000; 200,000; 400,000; and 800,000 instances.

4.1 Absolute accuracy

Figure 3 shows the accuracy (in terms of percent correct) of the classifier on the test data as the amount of training data increases. As a baseline it also shows the performance of a simple, flat lookup-table classifier—called the "strawman". The strawman classifier counts

[11] http://www.cs.waikato.ac.nz/ml/weka

	Number of training instances					
	10,000	50,000	100,000	200,000	400,000	800,000
Hierarchical classifier	32.53	43.47	47.63	50.76	53.11	55.32
Strawman	17.82	27.08	31.13	35.19	38.74	42.54
LCSH seen (%)	30.88	44.56	50.41	56.23	61.48	66.61

Table 5: Accuracy on test data and percentage of test data seen in training data for given amount of training data

the number of times the sequence of normalized LCSH from a test instance co-occurs with each LCC Outline classification in the training data, and predicts the classification with the highest co-occurrence. In the case of a tie it chooses randomly, and for LCSH records that do not occur in the training data it predicts the LCC classification that is most frequent overall in the training data.

Table 5 summarizes the results in tabular form. It also shows, as an upper bound for the strawman classifier, the percentage of the sequences of normalized LCSH in the test data that occur at least once in the training data. Interestingly this percentage is much larger than the accuracy of the strawman classifier. This implies that there are a large number of records with contradictory LCC classifications (partly because we discard subdivisions in the LCSH normalization process). This ambiguity means that no learning algorithm can predict the test data perfectly.

Classification accuracy increases as the amount of training data increases but the returns diminish. The largest increase in accuracy occurs after processing the first 50,000 training instances, demonstrating that at this point the main branches of the hierarchy are already sufficiently populated for the learning algorithm to obtain accurate estimates. However, by moving from 400,000 to 800,000 instances the accuracy still improves by more than two percent. This indicates that adding even more training data would result in further accuracy gains (although it appears that exponential growth in the training data is required to achieve significant gains). Approximately 1% of the test instances do not exhibit any LCSH occuring in the 800,000 training instances. For these instances the machine learning algorithm cannot make reliable predictions, though we have included them when we calculated the performance statistics. When the hierarchical classifier is applied in real applications, these records should be referred to a librarian for manual classification.

4.2 Partial successes

Straightforward accuracy is not the only performance measure that makes sense in this application because it does not reward partially correct predictions (i.e. whether parts of the predicted path through the LCC hierarchy are correct). Table 6 contains some additional performance statistics that take the hierarchical nature of the problem into account.

Percent too specific measures the proportion of assigned LCC intervals that are too specific (i.e. where the correct path is a prefix of the predicted path). *Percent too general* measures the proportion of assigned LCC intervals that are too general (i.e. where the

	Number of training instances					
	10,000	50,000	100,000	200,000	400,000	800,000
Percent too specific	3.53	3.83	3.84	3.76	3.78	3.77
Percent too general	3.77	3.33	3.21	3.16	3.19	2.93
Average overlap	46.71	57.15	60.95	63.75	65.83	67.61
Percent correct at level 1	62.39	71.66	74.76	77.33	78.94	80.27
Percent correct at level 2	48.45	59.15	63.07	65.73	67.78	69.54
Percent correct at level 3	35.05	45.79	49.92	53	55.39	57.55
Percent correct at level 4	24.89	34.54	38.68	41.4	43.71	46.24
Percent correct at level 5	21.34	31.96	35.83	38	40.38	42.22
Percent correct at level 6	14	21.13	24.48	26.15	28.23	30.08
Percent correct at level 7	8.44	11.16	12.5	13.66	14.39	16.12

Table 6: Additional performance statistics for test data

predicted path is a prefix of the correct path). These two criteria tell us which errors are due to cases where the classifier decides to stop too late or too early respectively (as opposed to cases where the predicted and the correct path diverge). As can be seen from Table 6 the values of these statistics change very little as the amount of training increases, although the percentage of predictions that are too general appears to drop slightly. We conclude that these errors are either due to an inherent limitation of our method or the result of true ambiguity among the possible classifications. Either way, only a minority of the classification errors (less than 7% of the test instances for the largest model) fall into one of these two categories.

The *average overlap* measures the overlap between the predicted path and the correct path (Table 6). The overlap is computed in the following way. Let m be the maximum of the length of the predicted path and the length of the correct path. Let k be the number of matching nodes in the correct path and the predicted path (i.e. the length of the common prefix). Then the overlap is given by k/m. Consequently the overlap is one if a prediction is correct. It is greater than zero as long as parts of the predicted path are correct. The average overlap is simply the average of this quantity with respect to all test instances. As can be seen from Table 6, the value of this statistic increases as the amount of training data increases, again with diminishing returns. By definition it is always larger than the percentage of correct predictions from Table 5. The difference is approximately 14% at 50,000 training instances and 12% at 800,000 training instances, suggesting that a large number of the test instances that are not correctly predicted are at least partially correct.

These partially-correct instances are quantified in the *accuracy at level* rows of Table 6. For each level of the LCC hierarchy this is the proportion of instances that are correctly classified at that level among all the instances that reach it—either by exhibiting a correct classification at that level or by getting a prediction assigned at that level (this implies that the classification is counted as an error if either one of the two is missing). For example, level 1 shows the accuracy in assigning records at the top level of the LCC (i.e. as, A, B, C, etc.) and that with 800,000 training instances, 80.27% of the test instances are classified correctly at the top level. With the same number of training instances only 16.12% of the test instances are correctly classified at level 7. 484 test instances reach this level, of which 56 have no predicted classification and 329 have no correct classification (at this level). In

	total	A	B	C	D	E	F	G	H	J	K	L	M	N	P	Q	R	S	T	U	V	Z
A	208	21	7		11	2	3		7	2		3		3	17	11						3
B	3512		81		3		1		2						2	3						
C	370		3	49	15	6	5	2	9					1	2	2						
D	4484		1		84			1	3	1					1	1						
E	1620				3	76	5	1	3	2					1	1						
F	2496				1	4	78	1	3	1						3						
G	1781		1		5	3	3	65	5						1	6			1			
H	7951				5	1	3		80	1						1			1			
J	1713				13	3	6		8	60	1					2						
K	694		1		2	2	1		16	8	55	1				5			1			
L	1619		1			1			3			86			1	2						
M	2486												95			3						
N	2333				1				1					89		2						
P	6430		1		4	1	1		1						84	2						
Q	6365															91	1	1	2			
R	1380		2						4							11	76					
S	950								7							11		77	1			
T	1985							1	5					1		10			76			
U	312				19	9	3		2	5						3				54		
V	49				20	2	2	6	10							6					53	
Z	1262		1		8	3	3	2	12	1		1		2	13	3			1			41

Table 7: Confusion matrix for top-level (LCC Schedule) classifications. Columns *A* to *Z* show percentages of the total; rows may not add to 100% due to rounding.

21 cases both are present but do not match. Hence (56 + 329 + 21)/484 = 83.88% of the instances that reach this level are classified incorrectly.

The percentage of correct predictions decreases as the number of intermediate classification steps increases because errors made at higher levels of the classification hierarchy cannot be rectified at lower levels. This is compounded by the fact that less and less data is available the deeper in the hierarchy a classifier is formed by the machine learning algorithm.

The *accuracy at level 1* row of Table 6 shows that even when 800,000 instances are used for training, almost 20% of the test records are incorrectly classified at the root of the hierarchy. This is perhaps not too surprising given that there are 21 possible classifications at the root. Table 7 sheds some light on this issue. It contains a confusion matrix for the predictions at the schedule level: the leftmost column contains the correct value of each instance, the second column contains the number of test instances in that schedule, and the remaining columns show the proportion of these test instances that were classified to each of the possible schedules. Correct classifications appear along the leading diagonal. For example, of the 208 test instances in Schedule A, 21% were classified into Schedule A, 7% into Schedule B, 11% into Schedule D, and so on. The matrix shows that many of the errors are due to records from the less populous schedules being classified into the more populous branches, particularly into Schedule Q, which contains the most common class, *QA1-43*.

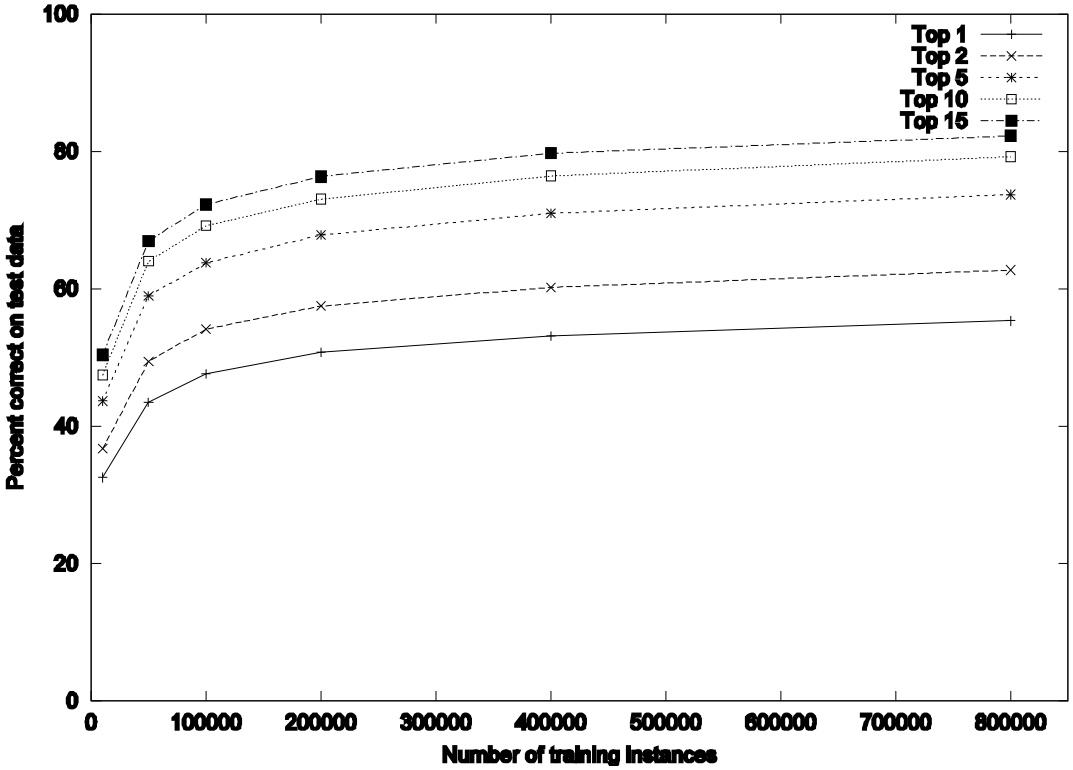

Figure 4: Accuracy on test data using top *N* predictions

4.3 Multiple predictions

Section 3.5 describes how a list of the top *N* predictions can be generated. Figure 4 shows the percentage of times the correct LCC appears among the top *N* predictions. This measure is equivalent to the average recall, though (like Larson) we think this label is misleading because there can only be one correct classification per document regardless of the size of *N*. Results are shown for the models described above with *N* set to 1, 2, 5, 10, and 15.

Accuracy increases as *N* increases, but with diminishing returns, indicating that the ranking method described in Section 3 produces useful confidence scores. Using the 800,000 instances model the correct classification appears among the top 15 predictions 82.2% of the time, but the gain compared to using the top 10 predictions (79.2% percent correct) is small.

4.4 Running time

Figure 5 shows the running time of the learning algorithm as the number of training instances increases. With 800,000 instances, training took approximately 27,000 minutes, or 19 days. These estimates were generated using the IBM Java Runtime Environment for Linux, Version 1.3.1, on an AMD Athlon XP 1600+ processor with 512 KB of RAM. Plotting this data on a log/log scale results in a line with slope 1.7. This means the learning

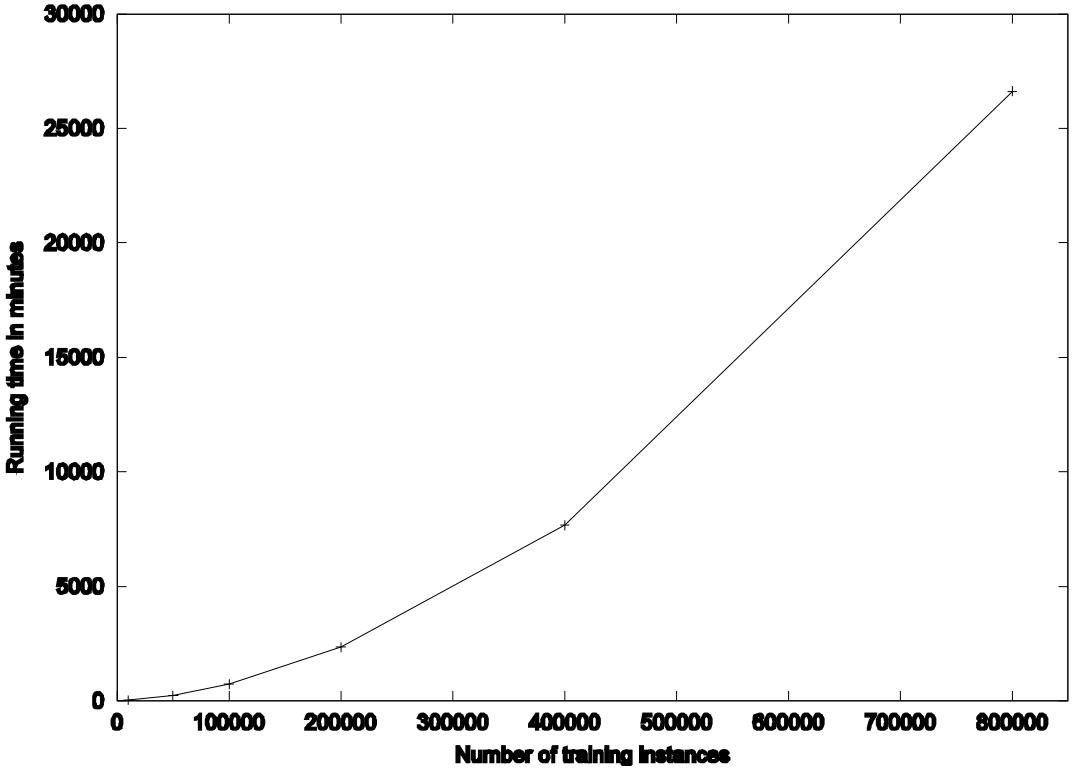

Figure 5: Running time for given amount of training data

algorithm scales at the order of $n^{1.7}$, where n is the number of training instances.

The time that is required for classifying a test record is also important when applying our method in practice. As the size of the model increases, the classification time also increases. To produce a ranked list of classifications (as described in Section 3) our method has to invoke an support vector machine classifier at every internal node of the LCC hierarchy. Using the largest model we observed a throughput of approximately 21 test instances per second, making the method sufficiently responsive even for interactive use by a librarian.

5 Discussion

The hierarchical classifier has been used to assign LCC to every INFOMINE record based on their LCSH metadata. Figure 6 shows the hierarchy at its root node; users can click on topic names to browse more specific subjects.[12] The figures to the right are the number of records beneath each child node; for example, INFOMINE contains 5,648 records that have been assigned an LCC in Schedule Q, more than any other Schedule.

To assess the quality of the classifications we have measured the accuracy of the hierarchical classifier on catalog data using information retrieval and machine learning metrics. The related projects described in Section 2.3 have different goals and have elected

[12] http://infomine.ucr.edu/dbase/cache/LCC/

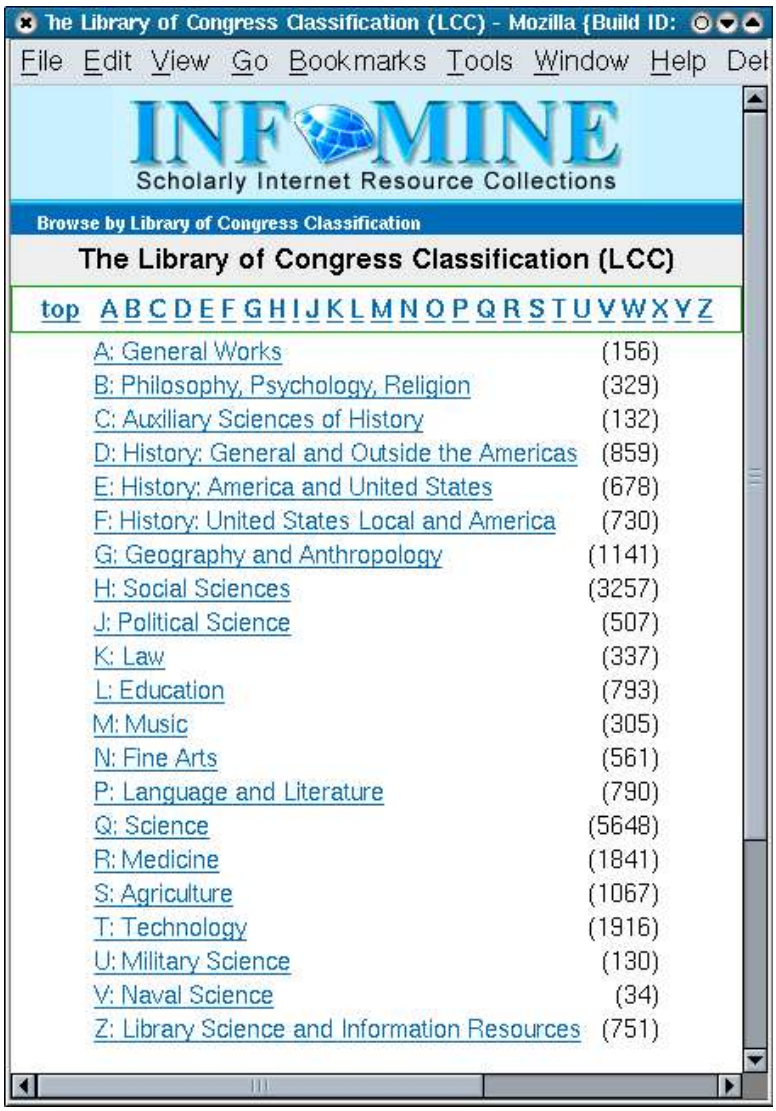

Figure 6: The INFOMINE LCC Browsing Interface.

to report different evaluation metrics; however, some comparisons are still possible.

5.1 Evaluation methodology

The primary evaluation metric in this paper is *classification accuracy*, or the proportion of times the classifier's best prediction for an example exactly matches its actual LCC. This is appropriate because the hierarchical classifier is optimized to assign a single LCC to each new record, because it is the standard for machine learning research, and because our applications call for a single classification.

When more than one classification is made, we have reported the number of times the correct class appears in the top *N* predictions. This is equivalent to recall. Larson (1991) argues that "with a single relevant class in a fairly large collection of classes, the

conventional retrieval effectiveness measures (such as recall and precision) are not very illuminating”, particularly given the hierarchical nature of this problem. Instead, he reports the mean and median *rank* of the correct classification when all potential classifications are sorted in order of predicted likelihood. It is not clear whether this is an improvement.

A problem with all these accuracy measures in a hierarchical setting is that they do not account for classifications that are partially correct. Partially correct classifications are more general or more specific than the true classification; for example, an instance may be assigned *DA1-995: History of Great Britain* instead of its sub-classification *DA20-690 England.* To account for the hierarchical class variable, we have also reported the proportion of too-specific and too-general classifications and the overlap measure.

5.2 Related work

Of the related work in Section 2.3, Pharos is the most comparable as it uses the LCC Outline and considers data from all 21 Schedules of the LCC (Dolin, 1998). Pharos accuracy for a single prediction is approximately 14%,[13] much lower than the 55.3% reported for the hierarchical classifier, and lower even than our strawman classifier (Section 4.1). This performance is satisfactory in Pharos because of the nature of the application, and because multiple LCC are assigned to each document. However, even when the best 10 phrases are chosen, Pharos' recall (less than 50%) trails the hierarchical classifier (79.2%).

Larson (1991) conducted a range of experiments, but his results are generally not comparable because he considers only Schedule Z, hierarchy is ignored, more classes are considered overall, and no LCSH normalization is attempted. Further, the test dataset was small: though sufficient catalog data was available (30,000 MARC records) for comprehensive training and testing, Larson evaluates his system on a small collection of 283 new books, 50 of which matched none of the 8,435 classifications derived from the training data.

When using similar training data to the hierarchical classifier (*all subject headings* treated as terms) the accuracy is approximately 38% for a single prediction, or 63% for 10 phrases[14]; for the *first subject only* training data the figures are 46.6% and 74.4% respectively. The hierarchical classifier appears to perform better, especially as these figures assume the best possible values for all other parameters of Larson's experiments, and in the 50 cases where no class was available, the nearest classification is considered correct. However, a direct comparison of the two approaches is impossible.

The hierarchical classifier ignores the order in which the LCSH are stored in the MARC records and ignores Title metadata. Larson's work suggests LCSH order is a significant predictor of LCC and could improve performance, but that the Title can be ignored.

[13] Estimated from Dolin (1998), Figure 7.5a.

[14] Estimated from Larson (1992), Figure 3.

5.3 Data ambiguity

The most fundamental problem with the classification problem is that LCSH are not always consistently applied by human experts. This phenomenon is well-documented (Chan, 1989), and we have observed it in practice in experiments with the combined University of California library catalog. Such inconsistencies in the training and testing datasets suggest the problem cannot be perfectly solved.

A further evaluation issue is raised by instances in the test data whose LCSH do not appear at all in the training data. The hierarchical classifier follows the machine learning practice of defaulting to the most likely child of each node, resulting in a hierarchical classification of *QA1-43 Science / Mathematics / General* in our datasets. This bias is evident in column *Q* of Table 7. When they appear in a browsing interface, these cases are not classified at all; instead they are labeled "unknown" so as not to mislead the librarians and other people who use the classifications. Approximately 1000 INFOMINE records, or 4% of the collection, fall into this category, compared to 1% for the test set drawn from the library catalog. Though we have not explored this area further, the relative benefits and costs of correct, incorrect and partial classifications is an interesting avenue for research.

A more subtle problem is caused by LCSH that are used only once in the training data, and then as non-primary descriptors that are only tangentially related to the LCC they purport to describe. Test instances that contain these LCSH will be assigned the incorrect class from the training data. This problem may be solved by increasing the size and internal consistency of the training dataset.

Another difficulty is caused by very general LCSH headings like "History" and "United States". These occur frequently in library catalogs but usually have subdivisions. However, our normalization process removes these subdivisions and the potentially important context they provide. It may be possible to address this problem using a normalization process that only removes rare subdivisions from LCSH headings.

6 Conclusions

We have presented a machine-learning-based system for assigning LCC to documents based on LCSH sets that exploits the hierarchical nature of the problem and outperforms similar work on LCC assignment. Although this is a specialized problem, the techniques are generally applicable to hierarchical classification problems, particularly those using very large hierarchies. The solution presented has been applied to the INFOMINE virtual library, where it is used to assign LCC to tens of thousands of records and supports a hierarchical browsing interface.

Acknowledgments

The authors would like to thank Nancy Douglas, Head of Cataloging at the Library of the

University of California at Riverside, for her assistance, and the U.S. Institute of Museum and Library Services, the Library of the University of California, Riverside, and the INFOMINE Project for their support.

Bibliography

Chan, L. M. (1989). Inter-indexer consistency in subject cataloging. *Information Technology and Libraries, 8(4),* 349-58.

Chan, L. M. (2001). Exploiting LCSH, LCC, and DDC to retrieve networked resources: Issues and challenges. In *Proceedings of the Bicentennial Conference on Bibliographic Control for the New Millennium* (pp. 159-78). Library of Congress.

Cristianini, N. & Shaw-Taylor, J. (2000). *Support vector machines and other kernel-based learning methods.* Cambridge University Press.

D'Alessio, S., Murray, M., Schiaffino, R. & Kershenbaum, A. (1998). Category levels in hierarchical text categorization. In *Proceedings of the 3rd Conference on Empirical Methods in Natural Language Processing.* SIGDAT (ACL Special Interest Group).

Dolin, R., Agrawal, D., Abbadi, A. E. & Pearlman, J. (1998). Using automated classification for summarizing and selecting heterogeneous information sources. *D-Lib Magazine.*

Dolin, R. A. (1998). *Pharos: A Scalable Distributed Architecture for Locating Heterogeneous Information Sources.* PhD thesis, University of California, Santa Barbara.

Dumais, S. (1991). Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments, & Computers, 23(2),* 229-236.

Dumais, S., Platt, J., Heckerman, D. & Sahami, M. (1998). Inductive learning algorithms and representations for text categorization. In *Proceedings of the International Conference on Information and Knowledge Management* (pp. 148-155). ACM Press.

Dumais, S. T. & Chen, H. (2000). Hierarchical classification of web content. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 256-263). ACM Press.

Fürnkranz, J. (2002). Round robin classification. *Journal of Machine Learning Research, 2,* 721-747.

Godby, C. J. & Stuler, J. (2001). The library of congress classification as a knowledge base for automatic subject categorization. In *Subject Retrieval in a Network Environment: Papers Presented at an IFLA Satellite Meeting Sponsored by the IFLA Section on Classification and Indexing and IFLA Section of Information Technology, Dublin, Ohio, USA* (pp. 14-16). OCLC.

Greiner, R., Grove, A. & Schuurmans, D. (1997). On learning hierarchical classifications. http://ai.uwaterloo.ca/~dale/papers/hier.ps.gz.

Joachims, T. (1998). Text categorization with support vector machines: learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning* (pp. 137-142). Springer Verlag.

Koller, D. & Sahami, M. (1997). Hierarchically classifying documents using very few words. In *Proceedings of the 14th International Conference on Machine Learning* (pp. 170-178). Morgan Kaufmann.

Larson, R. R. (1992). Experiments in automatic library of congress classification. *JASIS, 43(2),* 130-148.

Library of Congress (1986-2001). *SuperLCCS: Library of Congress Classification Schedules combined with additions and changes.* Gale Research Inc.

Library of Congress (1990). *LC Classification Outline* (6 Ed.). Library of Congress.

Library of Congress (1999). *MARC 21 Format for Bibliographic Data.* Library of Congress.

Library of Congress Subject Cataloging Division (2001). *Library of Congress Subject Headings* (24 Ed.). Library of Congress.

McCallum, A. K., Rosenfeld, R., Mitchell, T. M. & Ng, A. Y. (1998). Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the 15th International Conference on Machine Learning* (pp. 359-367). Morgan Kaufmann.

Mladenic, D. & Grobelnik, M. (1999). Assigning keywords to documents using machine learning. In *Proceedings of the 10th International Conference on Information and Intelligent Systems IIS-99.* Faculty of Organization and Informatics, University of Zagreb.

Ng, H. T., Goh, W. B. & Low, K. L. (1997). Feature selection, perceptron learning, and a usability case study for text categorization. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 67-73). ACM Press.

Platt, J. (1999). *Advances in Kernel Methods—Support Vector Learning*, chapter Fast training of support vector machines using sequential minimal optimization, (pp. 185-208). MIT Press.

Ruiz, M. E. & Srinivasan, P. (2002). Hierarchical text categorization using neural networks. *Information Retrieval, 5(1)*, 87-118.

Salton, G. (Ed.). (1971). *The SMART retrieval system*. Prentice-Hall.

Sun, A. & Lim, E.-P. (2001). Hierarchical text classification and evaluation. In *Proceedings of the 2001 IEEE International Conference on Data Mining* (pp. 521-528). IEEE Computer Society.

Sun, A., Lim, E.-P. & Ng, W.-K. (2002). Personalized classification for keyword-based category profiles. In *Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries* (pp. 61-74). Springer Verlag.

Taira, H. & Haruno, M. (1999). Feature selection in SVM text categorization. In *Proceedings of the 16th Conference of the American Association for Artificial Intelligence* (pp. 480-486). AAAI Press.

Thompson, R., Shafer, K. & Vizine-Goetz, D. (1997). Evaluating Dewey concepts as a knowledge base for automatic subject assignment. In *Proceedings The Second ACM International Conference on Digital Libraries* (pp. 37-46). ACM Press.

Weigend, A., Wiener, E. & Pedersen, J. (1999). Exploiting hierarchy in text categorization. *Information Retrieval, 1(3)*, 193-216.

Witten, I. H. & Frank, E. (2000). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann.