Working Paper Series
ISSN 1177-777X

Text Categorization and Similarity Analysis: Similarity measure, Architecture and Design

Michael Fowke[1], Annika Hinze[1], Ralf Heese[2]

Working Paper: 12/2013
December 2013

[1]Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand
[2] Pingar International Ltd.
152 Quay St, Auckland, New Zealand

Categorization and Similarity Analysis: Similarity measure, Architecture and Design.

Michael Fowke[1], Annika Hinze[1], Ralf Heese[2]

[1]University of Waikato, Hamilton, New Zealand
[2]Pingar International Ltd, Auckland, New Zealand

Abstract

This research looks at the most appropriate similarity measure to use for a document classification problem. The goal is to find a method that is accurate in finding both semantically and version related documents. A necessary requirement is that the method is efficient in its speed and disk usage. Simhash is found to be the measure best suited to the application and it can be combined with other software to increase the accuracy. Pingar have provided an API that will extract the entities from a document and create a taxonomy displaying the relationships and this extra information can be used to accurately classify input documents. Two algorithms are designed incorporating the Pingar API and then finally an efficient comparison algorithm is introduced to cut down the comparisons required.

1. Introduction

Document classification and provenance has become an important area of computer science as the amount of digital information is growing significantly. Software is now required to show similarities between documents (i.e. document classification) and to point out duplicates and possibly the history of each document (i.e. provenance). This honours project is done with Pingar who are a company based in Auckland who aim to help organise the growing amount of unstructured digital data. Pingar provided the Pingar API and taxonomy generator which are software tools that assist with document classification. Pingar also provided a document corpus to use for testing of the software to be created.

The intended outcome of the system is the ability to find the strength of semantic relationships and version relationships between an input document and any of the documents in the corpus. If a company has a collection of documents, they will be able to use the software to analyse a new document and find semantically related documents and if it is a version of an existing document. The literature review [1] covered a number of different existing implementations that attempted to classify documents and make it easier for a user to organise their digital information. This report follows on from the literature review and is a more extensive look at the different similarity measures. The similarity measures are analysed on their ability to find related documents both semantically and in version. The measures are also analysed on their speed and disk space required. The aim is to fully understand each and to choose the most appropriate to implement in the design of the software.

The reminder of the document is structured as follows. We start with some background on the project. The next section looks at each of the different similarity measures used to classify documents. The approaches are analysed on a set of criteria to determine the most appropriate. This method was then chosen and the rest of the document covers the research using this method. By the end of the report the design of the software can begin based on the conclusions found in the research.

2. Background

This covers the background and resources that are required to understand the research problem.

2.1 Example documents

Throughout the report, five example documents will be used to illustrate how each of the similarity measures performs. As input text examples, the short documents [2], [3], [4] shown in the appendix are used. Figure 1 shows the relationship between each of the documents.

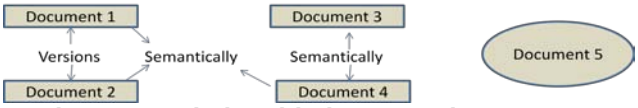

Figure 1: Relationship between documents

2.2 Semantic technology

Pingar have provided two of their semantic technology software: Pingar API for extracting entities from a document collection and a Taxonomy generator for finding relationships between extracted entities. These two technologies will be used to generate more accurate classifications than using the document text alone.

An example to illustrate the technology is one using document 3. The Pingar API extracts *Jason Dufner* and *Keegan Bradley* as entities from the document. These entities are related as they are both people and the API returns this information which is visualised in Figure 2. The entities are the objects and the taxonomy gives the semantic relationships between them.

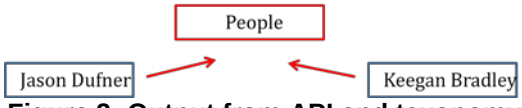

Figure 2: Output from API and taxonomy

2.3 Software structure

As an outcome of the literature study reported in [1], the overall structure of the document classification software was designed as shown in Figure 3. The blue rectangles in Figure 3 are software and include the Pingar API and taxonomy generator as well as the software that will be created. The green ovals show the output from the Pingar software. The input to the system is shown at the top and is a document to be analysed. The output from the system is shown at the bottom and is the document with calculated relationships. The grey rectangle is the similarity measure which is the main topic of this report.

As shown in Figure 3, the software will receive a static input document with the aim of identifying which documents in the document corpus are semantically related and which ones are versions of the same text. As only the content of the static documents is known (no history data), the only classification strategy available is content analysis. The outcome of the software will be a set of relationships between the input document and documents within the collection. Each relationship will have a value from 0.0-1.0 to show how strong the relationship is between documents.

Input Data: Pingar provided a document corpus the software will be tested on. Static input documents mean that the software is unaware of any interactions a person has made with the documents before classification. A single input document will be compared against the documents in the corpus.

Pingar Software: will be used to initially analyse the documents. More information about the Pingar API and taxonomy generator is given in section 2.2. The software will produce a taxonomy and a set of named entities with location references. The taxonomy shows the relationships between the extracted entities (e.g. a hierarchical structure). The API also gives information on which documents an entity occurs in. Named entities are phrases that contain the names of persons, organizations and locations [5].

The classification software: will use these entities from the Pingar API and taxonomy generator as well as the original document text to output the similarities between the input document and documents in the corpus. These outputs are both required and are then fed into the software.

Distance measure: This is the component of the system that takes the input and determines the distance between two documents i.e. their similarity. The measure will determine the strength of the relationship between each document.

Output data: The output will be a list of relationship values between the input document and each of the documents in the corpus. The values will be between 0.0 and 1.0 to show the strength of the relationship with 1.0 being a perfect match and 0.0 being no relation. A version relationship is a closer relationship than being semantically related so values from Θ - 1.0 are reserved for documents being related by version. The symbol Θ is a placeholder as it is not yet known what value should be the lower range of the version relationship. A score of Θ means a weak version relationship. Scores below Θ are given to documents with no version relationship. Only document relationships will be shown that are over a certain threshold i.e. 0.5.

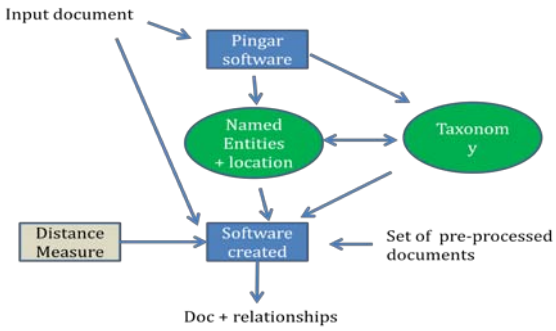

Figure 3: Overall structure of the software

2.4 Example of system

When document 3 is fed into the system, the extracted entities include *golf*, *major*, *Jason Dufner* and *New York*. The taxonomy will also identify that *New York* is a location and Jason Dufner is a person. This information is then input to the software to be created as well as the original document text. The system has access to a corpus of pre-processed documents that will be compared to the input document. These are processed so the computation time is minimised. The software created will take the input document and information from Pingar software and use the distance measure to find any documents in the corpus that are related. If Document 4 is in the corpus then one of the relationships output will be that document 4 has a high semantic relationship with document 3. This is due to document 4 sharing common entities such as *golf* and *Jason Dufner*. Details on the similarity measure are provided in the next section.

3. Similarity Approaches

The similarity measure is the remaining part to finalise before the software can be built. This is the measure that will determine to which extend two static input documents are related. To make an informed decision for the most appropriate similarity measure a number of criteria are introduced to help identify the best approach (section 3.1). Each of the measures analysed is then discussed in its design and illustrated using the example scenario from section 2.1. A table at the end of this chapter will summarise the performance of each measurement. Each approach is awarded a ++ or + to show very well performing or fairly well performing or a -- or - to show very poorly performing or slightly bad performance.

3.1 Assessment criteria

The following criteria are used to evaluate the similarity measures.

1)Accurately find versions of the same document

The similarity approach must accurately identify that versions of the same document are related. A version is a document with mainly the same content with only a few extra words inserted or removed. Documents 1 and 2 are versions as the first two paragraphs in document 2 have only a couple of extra words and document 2 has an

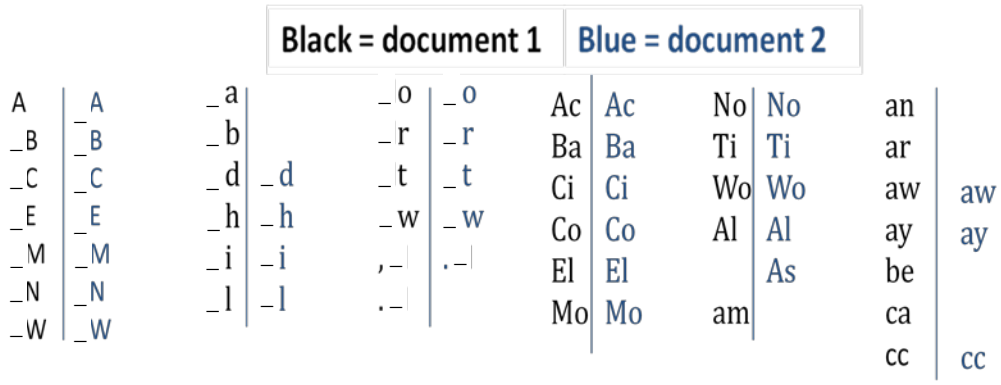

Figure 4: Shingles from documents 1 and 2

extra paragraph at the end. A similarity approach is awarded ++ if it is capable of finding document versions using the initial document text alone and a single + if it can find versions but only by incorporating the Pingar API.

2) Accurately find semantically related documents

The similarity approach must accurately identify that two documents that share a high number of common themes and topics are semantically related. Documents 3 and 4 are semantically related as they are both talking about *Jason Dufner* winning his first *PGA Championship* title. A similarity approach is awarded a ++ if it can find semantically similar documents from the initial document text alone, and a single + if it can only when the Pingar API is incorporated.

3) To produce a smaller representation of the input document (disk space)

The space required to store digital information is increasing and the method chosen should represent the input document in a more compact form without losing important information. The software should use as little disk space as possible so this criterion is necessary. A similarity measure is awarded a mark between ++ and -- based on how compact a document becomes with ++ being maximum compression.

4) Number of comparisons required

Speed of algorithm is very important, particularly when the software is run on a large number of documents. The computation time is heavily dependent on how many comparisons are required between documents. The greater the number of comparisons or operations required to find similar documents, the less efficient and slower the algorithm will be. The software should be able to find related documents with as few comparisons as possible. A similarity measure is awarded a mark between ++ and -- with ++ being a very low number of comparisons required.

5) Speed of comparisons

The number of comparisons required impacts speed but also the time taken for each of the comparisons. This criteria rates each approach on the computational complexity of each comparison required between the documents. The software should be able to do each comparison quickly as this will likely cause the entire software to run quickly. A similarity approach is awarded a mark between ++ and -- with ++ being a very quick time for each comparison.

3.2 Sim Hash

The first similarity approach looked at was the Simhash algorithm, described by Charikar [6]. They argue that the amount of information currently is large and we should not be looking to compare entire documents but rather creating smaller representations of each document, which will then be compared. Hashing is an example of this strategy as the content is represented by hashes which leads to greater disk space efficiency and speed. Simhash is calculated by applying a family of hash functions to each of the input phrases and the output is a value between 0 and 1. 1 shows that documents are identical and 0 shows that documents are very different. Charikar states that the similarity between two sets produced by using the family of hash functions can be estimated by counting the number of matching coordinates in their corresponding hash vectors. Similarity estimation is based on a test of equality of hash function values.

An implementation of the simhash algorithm is provided as on a separate website [7]. It also describes why a simhash algorithm is much more effective than a normal hashing algorithm. A normal hashing algorithm will give very different hashing values if the input phrase differs only slightly. Simhash will compute similar hash values for the first two paragraphs of documents 1 and 2. Normal hashing would not do this due to the few extra words and word reordering.

Simhash works well for finding different versions of the same document. Two different versions of documents are likely to be the same for large sections of text with one having additional text or text removed. Figure 4 shows the first portion of shingles (2 letter pairs) extracted from the first paragraph of documents 1 and 2 when ordered alphabetically. The simhash value is calculated using the sum of the hash values of each of the shingles. Despite the different word ordering and slightly different words, both documents share most of the shingles so the hash value will be very similar. The extra paragraph in document 2 cannot be matched with document 1 but two paragraphs is enough to show a version. Simhash also performs well when words have different endings. Document 1 uses *official* and document 2 *officially* yet

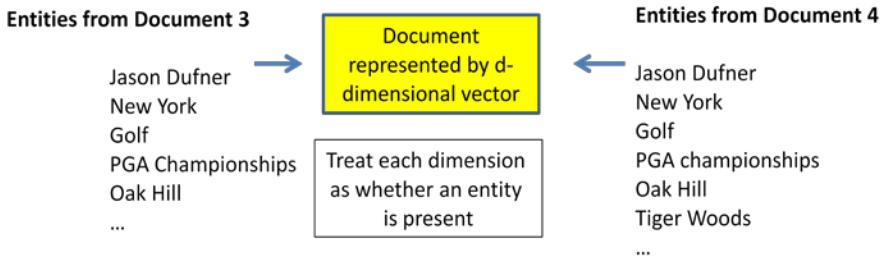

Figure 5: Clustering using documents 3 and 4

simhash can identify that the majority of the word is the same.

Simhash would not work so well in finding documents that are related semantically when used on the original text. When analysing the entire document simhash is unable to identify the key concepts to compare between documents. Also simhash is looking at the words themselves and not at the meaning of the words. Documents 3 and 4 are related semantically and share a number of common key words but simhash is unaware what the key words are. Document 3 uses the term *winning score* and document 4 uses *leading score* which will not be determined as the same by simhash which is looking at the letters and words. The Pingar API would make Sim Hash accurate in this area. Creating a hash value of the phrase created by joining all of the extracted entities together gives a very compact representation of the key concepts in the document. For Document 3 this would mean hashing *golf, Jason Dufner* etc. This one hash value is all that would be required to compare this document with others in the collection of documents. There are other variations that could be used such as calculating the simhash value for all the extracted concepts in each of the paragraphs in the document. This would again be compact as only one number per paragraph would be required.

Simhash performs well on the criteria required by the similarity measure. As stated previously it is able to find documents that are related in version. It is also able to identify documents that are semantically related but only by using the output of the Pingar API as an input to hashing. A major advantage of simhash is its small representation of a document. When identifying versions of documents then simhash will give a single number per chunk (likely a sentence). This means the entire document can be represented by a list of values such as document 5 being represented by 5 hash values. When identifying semantically related documents, a document can be represented by a single number being the hash of the extracted concepts from the document. The power of using numbers instead of text is really exploited in the next stage of the algorithm. Once the chunks of text have been turned into hash values, the chunks can be ordered based on the numerical value of the hash. This saves time in the search for related documents. Finding versions using simhash involves a number of comparisons with each of the chunks needing to be checked against each chunk in another document. With the ordering of the hashed chunks it reduces the number of comparisons required as each chunk only requires checking against hash values that are similar in value. The number of comparisons becomes efficient due to the ability to order numbers. The computation required in each comparison is also minimal. Each comparison involves finding the number of bits difference between two numbers which is easy and fast to compute.

3.3 Clustering using Wikipedia guidance

The next method considered is to use Wikipedia to cluster the documents. A similar approach was used as by Huang in [8] and [9]. Wikipedia is becoming an increasingly useful resource for clustering documents as it is a huge, well structured collection of information. Knopp et al. state that Wikipedia has been explored in a number of natural language processing tasks in the last years [10]. Huang uses Wikipedia and the links to related pages to generate relationships between concepts. This information can then be used to supervise the normally unsupervised method of clustering. Using the extra semantic information found through Wikipedia analysis produced far more accurate clusters than the clustering algorithm could alone.

The Wikipedia part of the method would not be required as the Pingar API gives the same output. Huang used

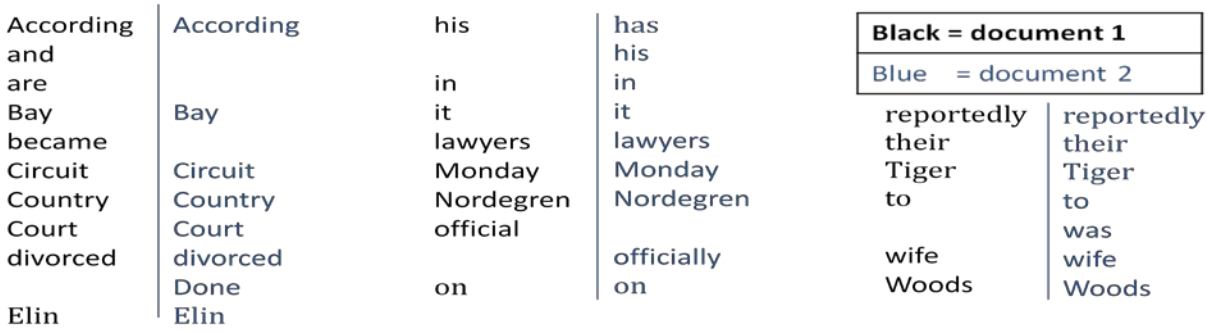

Figure 6: Words from documents 1 and 2

Wikipedia to find the relationship between concepts in a document which the Pingar API does also. Concepts from document 5 such as *physical exercise* and *obesity* would be found to be related using Wikipedia which the Pingar API will also find. This is a clever technique however it is not required in this software as the Pingar API does this.

Of more relevance is the technique that Huang used for finding the similarity of documents once Wikipedia was used for finding the relationship between concepts. Huang states that concepts are clustered according to their pair-wised semantic relatedness as computed from Wikipedia. Active learning is then applied to documents using the clustered concepts to derive a "keep together" or "separate" relationship. From our example documents it would find that *golf* and *PGA championships* are concepts that should be kept together and *golf* and *bikes* are likely to be separated. The report describes the method used to determine if a document was related to a cluster of concepts. The weighting of a cluster involves a calculation using the number of occurrences of a concept. This is one method that can be used to find the relationship between a document and concepts but it is essentially a word frequency approach which is addressed further in the next method on word frequency.

Huang uses Cop-Kmeans clustering algorithm (a variation of the k-means algorithm) to cluster the documents. Cop-kmeans uses the relationships as stated above to guide the clustering but underneath it still uses the simple k-means algorithm. Huang states that the clustering algorithm can only relate documents that use identical terminology, and important semantic relations between terms such as acronyms, synonyms, hypernyms, spelling variations and related terms are all ignored. This means that clustering is not able to find documents to be related unless they share exact words as the algorithm is not aware of synonyms and other important language characteristics. Clustering would not identify that *winning score* and *leading score* from documents 3 and 4 are related. Clustering could be extended to look at only the concepts extracted by the Pingar API and synonyms to resolve this issue.

Clustering would be able to identify semantically related documents if it used only the extracted concepts rather than the entire document. As stated by Huang [8], clustering can only relate documents that use identical terminology and will miss semantic relations. Huang also states that the terms used by clustering are usually single word terms which causes the algorithm to miss important concepts that are more than one word. These issues would be solved by using the extracted concepts from the Pingar API. Documents 3 and 4 are related semantically and Figure 5 shows how clustering can identify this relationship. The entities are extracted and clustering represents each document by a d-dimensional vector with each dimension being the presence of an extracted entity. These vectors will be compared and related documents grouped accordingly. Analysing the text alone will not find that concepts like *golf title* and *golf trophy* are related so synonyms from the Pingar API need to be used to find these concepts to be related.

Clustering is unable to find documents to be versions as this involves using far too many dimensions in the d-dimensional vector. When looking for versions, the entire document text needs to be analysed as it is not just the extracted entities that are important. This would mean thousands of dimensions in the vector which is impractical. It would also become similar to a word frequency approach which is discussed in the next section.

Clustering only meets some of the criteria. It is unable to identify versions as it cannot handle the high number of dimensions required in the vector representation. It is able to identify documents that are related semantically using the modification above that involves clustering using only the extracted concepts from the Pingar API. Clustering is able to represent the document in a more compact form as it considers a document as only containing a number of key terms. This is not quite as small as the single numerical value used by simhash but still an efficient use of disk space. Clustering treats the entire document as a whole and there is no implementation of clustering that involves breaking the document into chunks therefore the number of comparisons is small. The comparisons between documents are also fast and efficient as a value is calculated for each document and compared to the mean value of each cluster and grouped accordingly.

3.4 **Word Frequency**

The final method introduced in the literature review was a technique using word frequency [11]. Stanford University used a system that uses word frequencies to detect plagiarism. The technique could identify if a submitted document was a copy of one already in their databases of documents (i.e. version). This is similar to the technique using simhash in that it is looking at the original document text. This method compares documents based on how many times certain words appear and if two documents are versions of each other and are very similar then they will have similar word frequencies.

Word Frequency works well in finding different versions of the same document using only the original document text with a few modifications. Documents 1 and 2 are versions of each other as it is clear the first two paragraphs are very similar and Figure 6 shows how this works. The figure shows the words from the first paragraphs of documents 1 and 2. Even though the word ordering differs and there are a few different words, when ordered alphabetically it is clear that most of the words are common therefore the documents are versions. If the word frequency was analysed over the entire document text then two versions of the same document will not appear related if one document contains an extra section with completely different terms and word ratios. Document 2 has the extra paragraph on *Tiger Woods* and *Lindsey Vonn* and when analysed as an entire document the word frequencies would not be matching. Analysing the document by paragraph would show that the first two paragraphs of each document are versions.

This method would find documents that are semantically similar by using a method similar to that used by simhash. Word frequency using only the original text would not work as it focuses on words and not synonyms and

misspellings. Documents 3 and 4 are very similar semantically and a word frequency approach would not work as it is looking at the entire document and not just the concepts which is all that matters for semantically related documents. It is also unaware of terms like *winning score* and *leading score* which are used in the different documents but mean the same thing. When the Pingar API is combined with a word frequency method then these issues are resolved. The Pingar API extracts the concepts from a document and these can be compared to the concepts of another document. If one document contains the same concepts (or synonyms) as another document then they are likely related semantically.

A word frequency approach does not meet all the criteria. It does meet the major criteria of being able to find versions of a document as long as it is implemented by breaking the document into chunks. It also meets the major requirement of finding semantically related documents as it can compare the presence of key concepts between documents when combined with the Pingar API. This approach does not perform so well on the remaining three criteria. Representing the document in a more compact way is an important requirement of the software. When looking for versions, this method would represent a document by a word frequency for each chunk (likely a paragraph) in the text. This is not reducing the size of the document at all. When looking for semantic relatedness this method would be better in that a document would be related by only a small set of main concepts. This is still not as compact as the simhash method which only requires a single hash value for all the concepts combined. This method will involve a number of comparisons between the chunks in different documents. The difference between this and simhash is that the chunks are likely to be paragraphs in this algorithm and sentences in a simhash implementation. As a result less chunks need comparing. But then as the chunks are represented by word counts rather than numbers, every chunk will need comparing whereas simhash can be much smarter about which chunks need comparing by ordering the chunks numerically. There is little ordering that can be applied to word counts except for partial alphabetical ordering. As a result the total number of comparisons would be similar. The speed of each comparison would be slow as each chunk would need comparing on the presence of different words and the count for each.

3.5 Summary of approaches

The performance of the three similarity approaches is summarised in Figure 7 and simhash receives the best score based on the assessment criteria.

	Simhash	Clustering	Word frequency
Accurately find versions of documents	++	--	++
Accurately find semantically related documents	+	+	+
Create a smaller representation of a document	++	+	-
Efficient number of comparisons required.	+	++	+
Good speed of comparison	++	+	-
Total	**+8**	**+3**	**+2**

Figure 7: performance on required criteria

The Simhash method is the only one that performed positively in the 5 criteria. Clustering is unable to identify versions of a document which is a major requirement of the project and for this reason it is not considered. The major differences between simhash and word frequency comes in the way they process documents and the speed and space required. Simhash represents a document by a list of hash values when trying to identify versions which is a compact representation. Document 1 would be represented by 3 hash values, 1 for each sentence in the document. This list can then be sorted so the comparison of chunks between documents can be done efficiently. Word frequency does not use a compact representation and instead looks at a word count for each chunk in a document. Doc 1 would be represented by a list of word : frequency pairs. This increases the space required and each chunk has no natural ordering to aid computation speed. The algorithm would need to check every chunk against every other chunk.

Figure 8 shows the difference in disk space required by the two methods and figure 9 shows how these values were calculated. The values were calculated for document 2. Both the approach to finding semantically related documents and versions of the same document consume over 10 times the disk space using word frequency.

	Simhash	Word frequency
Finding versions	12 bytes	497 bytes
Finding semantically relatedness	4 bytes	48 bytes

Figure 8: disk space required for documents

Simhash	Word frequency
(32 bits per hash value = 4 bytes)	(Average word is 6 letters = 6 btyes)
Version:	**Version:**
= 3 sentences * 4bytes	Each paragraph is represented by a list of word-occurrences pairs i.e. golf-2. The word requires on average 6 bytes and the number 1 byte so 7 bytes per word. Document 1 has 24 different words in paragraph 1, 26 different words in paragraph 2 and 21 different words in paragraph 3.
	= 24*7 + 26*7 + 21 * 7
= 12 bytes	= 497 bytes
Semantically	**Semantically:**
= 8 extracted entities into 1 hash	6 bytes per each of the 8 entities extracted
= 4 bytes	= 6 * 8 = 48 bytes

Figure 9: calculation of bytes required

It was shown above that simhash performs better than word frequency in storage space required. The quality of the classification is also crucial in the decision making process. Simhash and word frequency are now compared on their accuracy. The statistics of interest are the precision and recall of the algorithms. The precision statistic shows the percentage of documents that are returned as similar that are in fact similar. i.e. true positives/ (true positives + false positives). Recall shows the percentage of similar documents that are returned by the algorithm. i.e. true positives / (true positives + false negatives). The f measure is the combination of these two statistics to give an overall accuracy reading.

Sood [12] states that simhash algorithms have good recall but tend to have a low precision. This means that the algorithm is able to identify almost all the documents that are related and return them as related. So it is likely documents 3 and 4 will be identified as related. The

algorithm can struggle in that it returns a number of false positives which are documents that it decides are related that should not be. It may find documents 4 and 5 to be related where they should not be. Sood worked with a recall percentage of 95% and found the algorithm to be fast but it did return a number of false-positives. Simhash receives a mid range f measure to reflect the high recall but lower precision. The precision can be improved with tightening up the criteria for documents to be related. If the recall percentage is lowered to 90% by only including documents that are more closely related such as documents 1 and 2, the number of false-positives will reduce. This is the concept of a ROC curve (Receiver Operating Characteristic) where the threshold used for deciding whether documents are related is altered and the changes in true positive and false positive rates are monitored. There are also ways of introducing the Pingar API which should increase the precision of the algorithm.

The recall for a word frequency algorithm is high as it will find documents to be related even if the words in the sentence are reordered or a few extra words are inserted. Like the simhash algorithm, the issues arise with word frequency having a lower precision. Paragraphs can have very similar word frequencies and thus appear as related but in fact not be versions at all. The f measure will be mid range again to reflect the high recall but lower precision.

Word frequency and simhash perform similarly in terms of recall and precision but the main difference between the two is in disc usage and also speed of comparisons. Simhash is able to work much quicker and with the use of less space so for these reasons it is chosen as the similarity measure to use. Testing can be done in the implementation to determine the threshold to use for accuracy to get the most useful results regarding recall and precision.

4. Sim hash implementation

The next stage is to work out exactly how to apply the simhash method for the highest accuracy. The question of this research is how to best combine the Pingar API and simhash method to find document similarity in semantics and in version. Matpalm [7] had an implementation of the simhash algorithm which this research is based on. This section outlines the initial implementation of the simhash algorithm

4.1 Initial algorithm

This is the algorithm used initially to calculate the simhash of chunks in a document. At each stage the idea is described as well as any experimentation done to find the best version. The simhash value used was the number of bits difference between two generated hash values, Figure 10 illustrates this difference in bits. The arrows represent the bits which differ in the numbers. The difference in bits in these two hash values is 8.

1537307734	=	0	1	0	1	1	0	1	1	1	0	1	0	0	0	0	1	0	1	1	1	0	1	0	0	0	1	0	1	0	1	1	0
1218804829	=	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	1	0	1	1	1	1	1	0	0	0	1	0	1	1	1	0	1

Figure 10: Difference in bits of sim hash values

Break the phrases up into features

This part of the algorithm involves breaking the input phrases into smaller chunks called shingles with a few letters. The example given broke the text into two letter shingles and this was the method created and used initially. The two letter shingles include spaces and no duplicate shingle is included. An example phrase is "Tiger Woods has reportedly divorced his wife Elin Nordegren" from document 1 and figure 11 shows this broken into shingles.

ti	ig	ge	er	r␣	␣w	w	oo	od	ds	s␣	␣h	ha	as	␣r	re	ep	po	or	rt
te	ed	dl	ly	y␣	␣d	di	iv	vo	or	rc	ce	d␣	hi	is	wi	if	fe	e␣	␣e
el	in	n␣	␣n	no	rd	de	eg	gr	re	en									

'Figure 11: Document 1 in 2 letter shingles

Hash each feature

It was suggested that a 32-bit hashing algorithm was used. This length was chosen to be long enough so that clashes did not occur with different input being hashed to the same output. The length was also short enough to be computationally efficient. Testing was done with 16 and 64 bit algorithms also. The larger the hash value the greater the bit difference between related phrases but it increased linearly compared to the difference between two other phrases. This means the bit difference between two different sets of phrases may have been 4 and 8 with 32 bit and 8 and 16 with 64 bit so the results did not reveal any further information.

The Java [13] hashing function was giving fairly small hash values even though it was a 32-bit hashing function. This appeared to be as the input to hash was small. As a result some of the phrases were found to be very similar even if they were in fact completely different. A new hashing algorithm implementation was written to resolve this. The hashing method is simple and gets the lowest 32 bits when the byte value of the two letters is multiplied by a large prime number. The prime number used was 27644437 [14] and it gave good hash values. The algorithm works much better with this new hashing algorithm. Figure 12 shows this hashing algorithm.

Input 2 letter shingle	= "ti"
Numerical value	= byte value of t * byte value of i = 116 * 105 = 12180
Hash value	= lowest 32 bits of (12180 * 27644437) = 1701793572

Figure 12: calculating hash value

Keep a 32 value array to modify

For each of the hashed shingles described above, if a bit *i* is set then add 1 to the value at position *i* in the array. If bit *i* is not set in the hashed value, then subtract one from the value at position *i* in the array. This was created fairly quickly and there was little room for different implementations.

Calculate 32-bit simhash value

Set bit *i* to 1 if the value at position *i* in the array above is > 0. Set the value to 0 otherwise. Again there is little room for variation.

Difference in bits.

At this point the simhash value has been calculated for each of the phrases input to the algorithm. The next step is to find the difference between each hashed phrase in terms of bits. Each of the phrases is analysed and the algorithm outputs the number of bits difference between each one.

4.2 Output from initial algorithm

A first implementation of the algorithm had been created and was tested on a few phrases. Each phrase represents a chunk that would be extracted from a complete document. Figure 13 shows the three input phrases from the example documents and Figure 14 shows the bit difference of the hashed values.

1) Tiger Woods and his wife, Elin Nordegren, are reportedly divorced
2) Tiger Woods has reportedly divorced his wife Elin Nordegren
3) It is now 5 years since Tiger won his last major title.

Figure 137: Three input phrases to demonstrate similar text

Phrases	Number of bits different
1,2	3
1,3	10
2,3	10

Figure 84: bit difference in example for versions

From this example it appears that the number of bits difference did show which of the phrases were the most similar. The output did not show very well just how different each of the phrases were. The difference between phrase 3 and the other two is huge but it is difficult to get a read on how different using just the value output.

This implementation works fairly well and can identify phrases that are similar in terms of words. This can be used for finding documents that are versions of one another. When one word in a phrase was changed, or removed, the simhash implementation would still find the phrases to be related.

The current implementation is not very good at finding phrases that are related in terms of semantics but not words. Figures 15 and 16 show an illustration of this.

1) a leading score of 270 the best in five years
2) The winning score was a 10-under 270, four shots better than the lowest score in the five previous majors
3) Woods finished well down the field

Figure 15: Input phrases for semantic relation

Phrases	Number of bits different
1,2	8
1,3	9
2,3	10

Figure 16: Bits different in semantic example

The first two phrases should be shown as slightly related but the output does not appear to show anything. The first two are shown as similar, likely because of common words such as *score* and *270* but the difference in the other phrases is not much more. The simhash algorithm is trying to find similarities in the phrases based on common 2 letter pairs in the phrases. It is incapable of recognising related terms such as *leading* and *winning* in the first two phrases. The method needs to be adjusted to find semantic similarities and is where the Pingar API is useful.

5. Incorporating the Pingar API

The implementation described above is a good introduction to simhash but alone it will not give a high level of accuracy for the classification software. The Pingar API can be incorporated to particularly help with finding semantic relatedness but also related versions. This section is broken into two parts, the first is looking at combining simhash with the Pingar API to find versions of a document, the second is looking at finding semantically related documents.

5.1 Finding document versions

This is the part where simhash already performs well. Simhash analyses each chunk or sentence in a document and creates a hash value which can be compared against every chunk in other documents. Each method is applied to documents 1 and 2 to show their effectiveness.

5.1.1. Using simhash on original document text

When looking for versions, it is possible that using simhash on the original document text is the most accurate way to find relationships. The document is broken into chunks (likely sentences) and each chunk has its simhash value calculated. Each document is represented by a list of hash values which can then be compared against every other document. If documents share a number of chunks that are within a fixed number of bits of each other then the documents are found to be versions of each other. Documents 1 and 2 are shown to be related as the sentences contain similar words and structure in the first two paragraphs as in Figure 17.

Document 1
Tiger Woods **and** his wife, Elin Nordegren, are reportedly divorced. According to their lawyers it **became** official in Bay County Circuit Court on Monday.

Woods and Nordegren **have** already commented on their divorce: "We are sad that our marriage is over and we wish each other the very best for the future."

Document 2
Tiger Woods **has** reportedly divorced his wife Elin Nordegren. According to their lawyers it **was done** officially on Monday in Bay County Circuit Court.

Woods and Nordegren commented already on their **unfortunate** divorce: "We are sad that our marriage is over and we wish each other the very best for the future."

Keywords	Jason Dufner, Oak Hill Country Club, PGA Championship, golf, Keegan Bradley, bogey,Atlanta, major, winning score
People	Keegan Bradley, Jason Dufner
Locations	Oak Hill Country Club, New York, Atlanta

Figure 17: Differences in documents 1 and 2

The first sentences are the same but with different word ordering and slightly different words such as *became* in document 1 and *was done* in document 2. The hash values will be very similar as simhash looks at 2 letter pairs. The hash values of the 2nd sentences will be very similar for the same reasons. Document 2 has a 3rd sentence which

document 1 does not but this is fine as the first part of each document is shown to be versions as 2/2 of the sentences from document 1 appear in document 2.

5.1.2. Using simhash on text with synonyms

The previous approach can be modified to incorporate the Pingar API. Figure 18 shows how the method will work on the first sentence of documents 1 and 2. This approach would find phrases to be versions if the author has swapped the concept for a synonym. The example finds *official* to be a concept and finds the synonyms to include in the input to hash. This example shows why looking at the concepts is unnecessary. Including the synonym has added no further accuracy to the algorithm. In documents 1 and 2 there are no concepts that are in one document but not the other. All the differences in the documents that make them versions are in word ordering and insertion/deletion of minor words such as *it became* rather than *was done*. The sentences in Figure 18 differ in that they have 2 different words and 1 word where it has a different ending. Using synonyms has not helped to improve this situation.

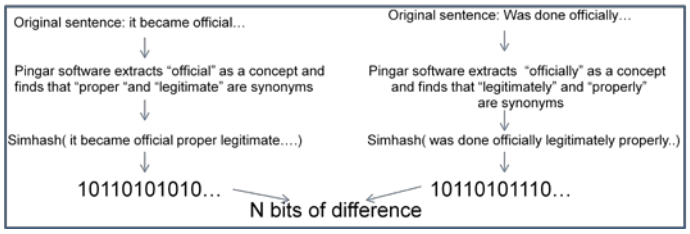

Figure 18: simhash on chunks with synonyms

5.1.3. Summary of document version approaches

Using synonyms is not necessary for finding document versions. When a person is writing a document, they are just as likely to change the smaller words in a sentence as the concept words. The example showed that using the synonyms did not improve accuracy as in general the synonyms will be a very small amount of the words that differ in versions. Simhash will find sentences to be versions if only 1 or 2 words differ so it is not necessary to introduce the synonyms for finding document versions.

5.2 Finding semantic relationships

This is the part which requires a lot of assistance from the Pingar API. Simhash alone is incapable of finding semantically related documents as it does not consider synonyms or misspelling. The concepts extracted by the Pingar API can be used as an input to hash to generate more accurate results. To find the effectiveness, each method is shown on documents 3 and 4. Figures 19 and 20 show the entities extracted from documents 3 and 4. These will be used to illustrate the upcoming approaches.

Keywords	Jason Dufner, Oak Hill Country Club, PGA Championship, golf, Keegan Bradley, bogey,Atlanta, major, winning score
People	Keegan Bradley, Jason Dufner
Locations	Oak Hill Country Club, New York, Atlanta

Figure19: Entities extracted from document 3

Keywords	Jason Dufner, Jim Furyk, Tiger Woods, golf, Oak Hill, American, major, PGA
People	Jason Dufner, Tiger Woods, Jim Furyk
Locations	New York, Oak Hill, Torrey Pines

Figure 20: Entities extracted from document 4

5.2.1. Hashing concepts from an entire document

The first approach to using the Pingar API is to extract all the concepts from the document and generate a simhash value for the concatenation of these concepts. Figure 21 demonstrates this process. Each document would be represented by a single number representing the hash value of each of the concepts combined. Comparisons between documents are fast and easy with just a single number being compared for each document. Documents that share mainly the same concepts will have a similar simhash value and be shown as semantically related.

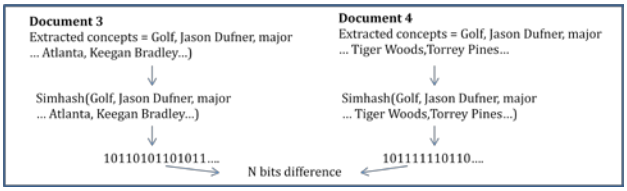

Figure 21: hashing of extracted concepts

5.2.2. Hashing concepts from each paragraph

The next approach is similar to the previous except the document is broken into sections. A document may contain two fairly separate sections and this method would identify this and still find related documents. If the first half of document a is on an identical topic to document b but then document a discusses a different topic in the second half, the previous approach would not find these documents related. The number of shared concepts would not be high enough to produce similar simhash values. Figure 22 shows an example of this from documents 3 and 4. The majority of each document is on the same topic however the last paragraph of document 3 is on the tournament 2 years ago whereas the last paragraph in document 4 is on *Tiger Woods*. The paragraph has a completely different simhash value which can be discarded as the rest of the paragraphs are similar.

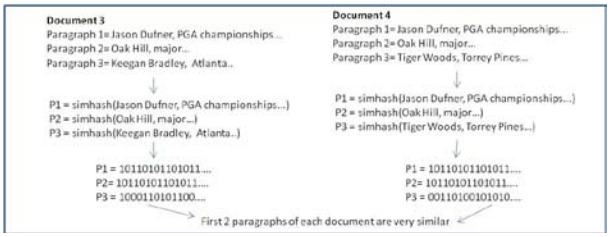

Figure 22: Hashing concepts per paragraph

5.2.3. Include frequency of entities

This adjustment to the simhash method for semantics uses the number of occurrences of each of the entities and can be applied to either of the approaches above to improve accuracy. If two documents are closely related in terms of topic, not only will they use common words or concepts throughout but they will use these key terms many times.

Figure 23 displays an example of two documents with the entities extracted and their frequency. The first 3 documents are all adaptations of document 3 and the last is document 5 to illustrate the introduction of entity frequency. One approach would be to include the term golf numerous times in the input to hash. i.e. for document 3a it would be simhash(golf golf golf golf Jason Dufner PGA championship). This is essentially a variation on the tf-idf algorithm (Term Frequency-Inverse Document Frequency) [15]. This algorithm calculates the importance of each term in a document and goes up when a term is included many times in a document and goes down when the term is used many times in the entire document corpus. The trouble with this is the simhash implementation looks at 2 letter shingles within the phrase and discards duplicates. So including golf many times in the simhash has the same effect as including it once.

It was then investigated what would happen if the simhash algorithm was altered to include duplicates. The result was an algorithm that places a very high weighting on the entities that occur many times. Documents 3a and 3b were found to be no more similar than documents 3a and 5 and 5 is completely different. The reason for this is that *golf* has such a high weighting in the calculation of simhash that the other terms are essentially ignored. To reinforce this the similarity between documents 3a and 3c was calculated and found to be really high. This shows that the simhash value for document 3a is so highly influenced by the entity golf that it is almost the same as hashing only golf. This tf-idf algorithm can still be used but it should be used carefully. An entities frequency is important as two documents that both mention *golf* 20 times should be shown as related regardless of the content of the rest of the documents which this algorithm would show. The semantic similarity should be calculated twice, once with the tf-idf included and once without. Calculating the semantic similarity without including entities many times will find documents 3a and 3b to be semantically related which is also true. When these two approaches are used carefully, documents can be more accurately analysed that using either approach alone.

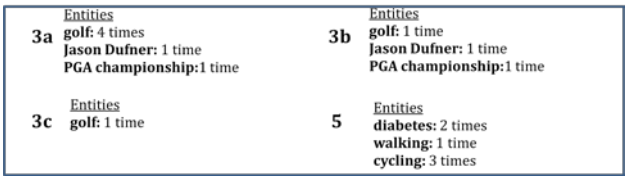

Figure 23: frequency of concepts in documents

5.2.4. Summary of semantic approaches

The second approach would find documents to be related even if only a few of the paragraphs are related. This is the desired output and hashing the concepts from the entire document would not find these documents which have only some related paragraphs. The example showed a situation where new entities are introduced in a paragraph of document 3 with different entities in the last paragraph of document 4. The hash per paragraph approach finds the simhash value for the first few paragraphs to be identical with only the last paragraphs being different which it then discards.

Care will have to be taken when using this method as the paragraphs in the documents may not always line up so well. One document may talk about topic a in the first paragraph and topic b in the second and a second document may do it vice versa. The comparisons between documents will need to check each paragraph with every paragraph in the other documents. The simhash values will be quite different between two paragraphs if one has an extra two entities with the rest of the paragraphs being very similar. This can cause an issue and for this reason it is probably only necessary to find one or two paragraphs that have similar hash values between two documents as it is fairly hard to achieve. Often the first paragraphs of documents sum up the main topics and these will be found to have similar hash values.

The frequency of entities will be considered by calculating semantic relatedness in two different ways. It was found that introducing an entity into the simhash calculation many times if it occurred many times in the document had a very large impact on the overall simhash value calculated. The documents will be compared semantically using both tf-idf and without.

A possible extension is to introduce broader/narrow concepts as determined by the pingar API and taxonomy generator. This would be similar to the frequency issue in that the stronger relationships between concepts can be used with greater weighting in the simhash value. This will be considered as an extension to the project to achieve greater accuracy.

6. Optimizing comparing simhash chunks

The simhash implementation involves comparing a number of hash values between documents to find numbers that differ by a small number of bits. This is a process which uses a high level of computation and should be designed to be as efficient as possible. Simhash can be efficient in that the hash values can be ordered so that the minimum number of comparisons are carried out to find the related chunks. If every chunk is compared against every other chunk then the algorithm runs in $O(n^2)$. Documents 1 and 2 will need 6 comparisons to

determine relatedness as document 1 has 2 sentences and document 2 has 3 sentences. This suggested optimization will reduce this.

6.1 Method for optimization

This is based on the method described in matpalm [7] but adjusted to fit this software. Each of the hashed chunks in a document must be checked against every other chunk in another document but the number of comparisons can be reduced.

Remove chunks from document 2 that are very different

First step is to count the number of bits set in the chunk from the first document currently being compared, call this x. Remove any chunks from the list of chunks from the second document that have more than x+n or less than x-n bits set with n being the threshold for number of bits difference that is considered related. If for example a hashed chunk in document 2 has 26 bits set and the chunk from document 1 has 10 bits set then the number of bits difference is always going to be at least 16.

Order the list of hashed chunks for the second document

If the bits different between two hashed chunks are in the lowest few bits then ordering the hash values will result in the similar chunks appearing next to each other in the list. Insert the hashed chunk from the first document into this list and remember its position. Figure 24 shows an ordered list of hash values and phrases (3,6) and (8,5) have ended up close together and both have a small bit difference.

Phrase number	value	Bit value	Bit difference from phrase above
4	934	0000001110100110	
3	2648	0000101001011000	9
6	2650	0000101001011010	1
1	37586	1001001011010010	5
8	40955	1001111111111011	6
5	40957	1001111111111101	2
2	50086	1100001110100110	9
7	64475	1111101111011011	9

Figure 24: ordered list of hash values [7]

Find chunks close to hashed chunk from the first document

Navigate through the list of chunks to find those that are within n bits difference from the hashed chunk from the first document that has been inserted into the list. These are chunks that are closely enough related to the first chunk. Remove these chunks from the list of chunks from the second document.

Rotate each chunk one bit to the left

So far the method has only found chunks that differ in the lower bits of the hash value. By rotating all of the values one bit to the left, the difference between each of the values will still remain intact. Figure 25 shows the rotated hash values with the bit difference the same as in the previous figure.

Phrase number	value	Bit value	Bit difference from phrase above
4	3736	0000111010011000	
3	10592	0010100101100000	9
6	10600	0010100101101000	1
1	19274	0100101101001010	5
8	32750	0111111111101110	6
5	32758	0111111111110110	2
2	3739	0000111010011011	9
7	61295	1110111101101111	9

Figure 25: Bit difference of rotated hash values[3]

Repeat step of ordering and finding related chunks

Now repeat the steps of ordering the list of hashed chunks from document 2 and then removing the ones within n bits difference to the hashed chunk from document 1. These chunks have a close enough relation to the hashed chunk from the first document.

Repeat steps 4 and 5

Rotate the bits in every chunk in the list by another bit to the left, sort the list and remove the closely related hash values. Repeat these steps as many times as there are bits in the hash values. i.e. rotate 32 times for 32 bit hash values.

Repeat steps 1-6 for each chunk from document 1

Repeat the process for each of the hashed chunks from the first document.

Repeat steps 1-7 for every other document in the collection

Repeat the earlier steps for every other document in the collection. So compare each chunk from document 1 with chunks in documents 2 till m with m being the number of documents in the collection. Then compare every chunks in document 2 with chunks in documents three till m. Continue until chunks in documents m-1 and m have been compared.

7. Conclusion

Through its performance on a set of criteria, simhash was found to be the best performing of the similarity measurements. Word frequency and simhash were both accurate in their classification however simhash was far more efficient in time taken and disk space used. A simple first version of simhash was introduced to discover how it worked and where it broke down. Pingar had provided an API and taxonomy generator which can be combined with the simhash method to fix most of the areas where the simhash algorithm struggled.

For document versions it was found that the best solution was to calculate the simhash value for each sentence within the original document and the document summarised by a list of these hash values. A method introducing synonyms into the document text was tested and discovered to be of no great benefit to the algorithm as the initial document text was satisfactory for finding versions of a document.

When classifying documents related semantically it was found that the best approach is to combine the extracted entities and their synonyms into a single simhash value for each paragraph within the document. Introducing synonyms helps the algorithm to find related paragraphs that use different terms for the same ideas. Analysing the

document by paragraph rather than as a whole meant a section of a document on a completely different topic would not stop the document being found to be related to another if the majority of the documents were similar. Documents will be analysed semantically twice, once with tf-idf and once without to improve the accuracy by including entity frequency. Broader/narrower relationships were considered but are considered extensions time permitting.

An efficient method for comparing chunks was then introduced. This method was based on an algorithm introduced in literature but modified to work with the nature of this simhash application. This improvement in efficiency helps make simhash a quick algorithm for classifying related documents.

8. References

[1] Fowke, M. (2013). Text categorization and analysis based on document history. Literaure Review. Waikato University.

[2] (2010, August 23). Divorce of Tiger Woods and wife finalized, Short News

[3] (2013, August 12). Golf: Jason Dufner claims first major title, Short News

[4] (2013, August 7). Study: Walking, cycling to work may lower diabetes risk, Short News

[5] Tjong Kim Sang, E. F., & De Meulder, F. (2003, May). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In*Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4* (pp. 142-147). Association for Computational Linguistics.

[6] Charikar, M. S. (2002, May). Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing* (pp. 380-388). ACM.

[7]Matpalm. The simhash algorithm. Retrieved from http://matpalm.com/resemblance/simhash

[8]Huang, A., Milne, D., Frank, E., & Witten, I. H. (2008, December). Clustering documents with active learning using Wikipedia. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on* (pp. 839-844). IEEE.

[9] Huang, A., Milne, D., Frank, E., & Witten, I. H. (2009). Clustering documents using a Wikipedia-based concept representation. In *Advances in Knowledge Discovery and Data Mining* (pp. 628-636). Springer Berlin Heidelberg

[10]Knopp, J., Frank, A., & Riezler, S. (2010). *Classification of named entities in a large multilingual resource using the Wikipedia category system* (Master's thesis, University of Heidelberg).

[11] García-Molina, H., Gravano, L., & Shivakumar, N. (1996, December). dSCAM: Finding document copies across multiple databases. In *Parallel and Distributed Information Systems, 1996.,Fourth International Conference on* (pp. 68-79). IEEE.

[12]Sood, S. (2011). *Probabilistic Simhash Matching* (Doctoral dissertation, Texas A&M University).

[13]Java. (2013). What is Java. Retrieved from www.java.com

[14] Wolfram Math World. (2013) Bell Number. Retrieved from http://mathworld.wolfram.com

[15]Ramos, J. (2003, December). Using tf-idf to determine word relevance in document queries. In *Proceedings of the First Instructional Conference on Machine Learning*.

9. Appendix: Example documents

Document 1

Tiger Woods and his wife, Elin Nordegren, are reportedly divorced. According to their lawyers it became official in Bay County Circuit Court on Monday.

Woods and Nordegren have already commented on their divorce: "We are sad that our marriage is over and we wish each other the very best for the future."

Document 2

Tiger Woods has reportedly divorced his wife Elin Nordegren. According to their lawyers it was done officially on Monday in Bay County Circuit Court.

Woods and Nordegren commented already on their unfortunate divorce: "We are sad that our marriage is over and we wish each other the very best for the future."

Tiger has since been linked to another Blonde woman in skier Lindey Vonn. Vonn was spotted course side during the BMW championships.

Document 3

Jason Dufner finished bogey-bogey on the two most difficult holes on the Oak Hill Country Club course in New York to claim his first major golf title at the PGA Championship on Sunday.

The winning score was a 10-under 270, four shots better than the lowest score in the five previous majors at Oak Hill.

Two years ago in Atlanta, the 36-year-old had blown a five-shot lead and Keegan Bradley ended up winning the title.

Document 4

The PGA championship concluded in New York on Sunday with Jason Dufner winning his first major golf trophy.

Dufner won the tournament by 2 strokes over American Jim Furyk at the Oak Hill course with a leading score of 270 the best in five years.

Tiger Woods finished well down the field which was frustrating for the hot favourite going into the event. It is now 5 years since Tiger won his last major title at the US open in Torrey Pines.

Document 5

People who walk to work are 40 percent less likely to develop diabetes and 17 percent less likely to develop high blood pressure than those who drive, a new study by UK researchers suggests.

Of the adults who used private transport such as cars, motorbikes and taxis to get to work, 19 per cent were obese, compared to only 13 percent of those who cycled to work and 15 percent of those who walked.

"This study highlights that building physical activity into the daily routine by walking, cycling or using public transport to get to work is good for personal health," states study co-author Anthony Laverty.