

TOPIC- STEMMING TECHNOLOGY

SUBJECT – E BUSINESS TECHNOLOGIES

COURSE- MASTER OF SCIENCES- BUSINESS  
CONSULTING.

PROFESSOR- PROF Dr EDUARD HEINDL

STUDENT- AJAY SINGH

## **DECLARARTION**

I hereby affirm that all the work in this project paper has been done by me. No one else was involved in this paper in any form. Whatever resources, I have used in this work, I have mentioned in the references section.

**AJAY SINGH**

## Information Retrieval

Information Retrieval (IR) is essentially the requirement of documents in a collection that should be retrieved to satisfy a user's need for information. The user's information requirement is represented by a query or profile, and contains one or more search terms, plus perhaps some additional information such importance weights. Hence, the retrieval decision is made by comparing the terms of the query with the index terms (important words or phrases) appearing in the document itself. The decision may be binary (retrieve/reject), or it may involve estimating the degree of relevance that the document has to the query.

At times, the words that appear in documents and in queries often have many morphological variants. Thus, pairs of terms such as "computed" and "computation" will not be recognized as equivalent without some form of natural language processing (NLP).

## Stemming

The morphological variants of words have similar semantic interpretations and can be considered as equivalent for the purpose of IR applications. It is in this light that, a number of so-called stemming Algorithms, or stemmers, have been developed, which attempt to reduce a word to its stem or root form. In a way, the key terms of a query or document are represented by stems rather than by the original words. It implies that different variants of a term can be conflated to a single representative form; it also reduces the size of the dictionary, that is, the number of distinct terms needed for representing a set of documents. A smaller dictionary size results in a saving of storage space and processing time.

For the purpose of information retrieval, it doesn't usually matter whether the stems generated are genuine words or not – thus, "computation" might be stemmed to "comput" – provided that (a) different words with the same 'base meaning' are conflated to the same form, and (b) words with distinct meanings are kept separate. An algorithm which attempts to convert a word to its linguistically correct root ("compute" in this case) is sometimes called a lemmatiser.

Not all the search engines support this functionality. Examples of products using stemming algorithms would be search engines such as Lycos and Google, and also thesauruses and other products using NLP for the purpose of IR. Stemmers and lemmatizers also have applications more widely within the field of Computational Linguistics.

## Examples

A stemmer for English language for example, should identify the string "cats" (and possibly "catlike", "catty" etc.) as based on the root "cat", and "stemmer", "stemming", "stemmed" as based on "stem". A stemming algorithm reduces the words "fishing", "fished", "fish", and "fisher" to the root word, "fish".

## History

The first ever published stemmer was written by Julie Beth Lovins in 1968. This paper was remarkable for its early date and had great influence on later work in this area.

A later stemmer was written by Martin Porter and was published in the July 1980 issue of the journal *Program*. This stemmer was very widely used and became the de-facto standard algorithm used for English stemming. Dr. Porter received the Tony Kent Strix award in 2000 for his work on stemming and information retrieval.

Many implementations of the Porter stemming algorithm were written and freely distributed; however, many of these implementations had quite a number of flaws. As a result, these stemmers did not match their potential. To eliminate this source of error, Martin Porter released an official free software support of the algorithm around the year 2000. He extended this work over the next few years by building SNOWBALL, a framework for writing stemming algorithms, and implemented an improved English stemmer together with stemmers for several other languages.

## Algorithms

There are several types of stemming algorithms which differ in respect to performance and accuracy and how certain stemming obstacles are overcome.

### Brute Force Algorithms

These stemmers employ a lookup table which contains the relations between root forms and inflected forms. To stem a word, the table is queried to find out a matching inflection. If a matching inflection is found, the associated root form is returned.

Brute force approaches are criticized for their general lack of elegance in that no algorithm is applied that would more quickly converge on a solution. In other words, there are more operations performed during the search than should be necessary. Brute force searches consume immense amounts of storage to host the list of relations (relative to the task). The algorithm is only accurate to the extent that the inflected form already exists in the table. Given the number of words in a given language, like English, it is unrealistic to expect that all word forms can be captured and manually recorded by human action alone. Manual training of the algorithm is overly time-intensive and the ratio between the effort and the increase in accuracy is marginal at best.

Brute force algorithms do overcome some of the challenges faced by the other approaches. Not all inflected word forms in a given language “follow the rules” appropriately. While “running” might be easy to stem to “run” in a suffix stripping approach, the alternate inflection, “ran”, is not. Suffix stripping algorithms are

somewhat powerless to overcome this problem, short of increasing the number and complexity of the rules, but brute force algorithms only require storing a single extra relation between “run” and “ran”. While that is true, this assumes someone bothered to store the relation in the first place, and one of the major problems of improving brute force algorithms is the coverage of the language.

Brute force algorithms are initially very difficult to design given the immense amount of relations that must be initially stored to produce an acceptable level of accuracy (the number can span well into the millions). However, brute force algorithms are easy to improve in that decreasing the stemming error is only a matter of adding more relations to the table. Someone with only a minor experience in linguistics is capable of improving the algorithm, unlike the suffix stripping approaches which require a solid background in linguistics.

For technical accuracy, some programs may use suffix stripping to generate the lookup table given a text corpus, and then only consult the lookup table when stemming. This is not regarded as a brute force approach, although a lookup table is involved.

### **Suffix Stripping Algorithms**

Suffix stripping algorithms do not rely on a lookup table that consists of inflected forms and root form relations. Instead, a typically smaller list of “rules” is stored which provide a path for the algorithm, given an input word form, to find its root form. Some examples of the rules include:

if the word ends in ‘ed’, remove the ‘ed’  
if the word ends in ‘ing’, remove the ‘ing’  
if the word ends in ‘ly’, remove the ‘ly’

Suffix stripping approaches enjoy the benefit of being much simpler to maintain than brute force algorithms, assuming the maintainer is sufficiently knowledgeable in the challenges of linguistics and morphology and encoding suffix stripping rules. Suffix stripping algorithms are sometimes regarded as crude given the poor performance when dealing with exceptional relations (like ‘ran’ and ‘run’). The solutions produced by suffix stripping algorithms are limited to those lexical categories which have well known suffixes with few exceptions. This, however, is a problem, as not all parts of speech have such a well formulated set of rules. Lemmatization attempts to improve upon this challenge.

### **Lemmatization Algorithms**

A more complex approach to the problem of determining a stem of a word is lemmatization. This process involves first determining the part of speech of a word, and applying different normalization rules for each part of speech. The part of speech is first detected prior to attempting to find the root since for some languages, the stemming rules change depending on a word’s part of speech.

This approach is highly conditional upon obtaining the correct lexical category (part of speech). While there is overlap between the normalization rules for certain categories, identifying the wrong category or being unable to produce the right category limits the added benefit of this approach over suffix stripping algorithms. The basic idea is that, if we are able to grasp more information about the word to be stemmed, then we are able to more accurately apply normalization rules (which are, more or less, suffix stripping rules).

### **Stochastic Algorithms**

Stochastic algorithms involve using probability to identify the root form of a word. Stochastic algorithms are trained (they “learn”) on a table of root form to inflected form relations to develop a probabilistic model. This model is typically expressed in the form of complex linguistic rules, similar in nature to those in suffix stripping or lemmatisation. Stemming is performed by inputting an inflected form to the trained model and having the model produce the root form according to its internal rule set, which again is similar to suffix stripping and lemmatisation, except that the decisions involved in applying the most appropriate rule, or whether or not to stem the word and just return the same word, or whether to apply two different rules sequentially, are applied on the grounds that the output word will have the highest probability of being correct (which is to say, the smallest probability of being incorrect, which is how it is typically measured).

Some lemmatisation algorithms are stochastic in that, given a word which may belong to multiple parts of speech, a probability is assigned to each possible part. This may take into account the surrounding words, called the context, or not. Context-free grammars do not take into account any additional information. In either case, after assigning the probabilities to each possible part of speech, the most likely part of speech is chosen, and from there the appropriate normalization rules are applied to the input word to produce the normalized (root) form.

### **Hybrid Approaches**

Hybrid approaches use two or more of the approaches described above in unison. A simple example is a suffix tree algorithm which first consults a lookup table using brute force. However, instead of trying to store the entire set of relations between words in a given language, the lookup table is kept small and is only used to store a minute amount of “frequent exceptions” like “ran => run”. If the word is not in the exception list, apply suffix stripping or lemmatisation and output the result.

### **Affix Stemmers**

In linguistic, the term affix refers to either a prefix or suffix. In addition to dealing with suffixes, several approaches also attempt to remove common prefixes. For example, given the word indefinitely, identify that the leading “in” is a prefix that can be removed. Many of the same approaches mentioned earlier apply, but go by the name **affix stripping**.

## Matching Algorithms

These algorithms use a stem database (for example a set of documents that contain stem words). These stems, as mentioned above, are not necessarily valid words themselves (but rather common sub-strings, as the “brows” in “browse” and in “browsing”). In order to stem a word the algorithm tries to match it with stems from the database, applying various constraints, such as on the relative length of the candidate stem within the word (so that, for example, the short prefix “be”, which is the stem of such words as “be”, “been” and “being”, would not be considered as the stem of the word “beside”).

## Language Challenges

While much of the early academic work in this area was focused on the English language (with significant use of the Porter Stemmer algorithm), many other languages have been investigated.

Hebrew and Arabic are still considered difficult research languages for stemming. English stemmers are fairly trivial (with only occasional problems, such as “dries” being the third-person singular present form of the verb “dry”, “axes” being the plural of “axe” as well as “axis”); but stemmers become harder to design as the morphology, orthography, and character encoding of the target language becomes more complex. For example, an Italian stemmer is more complex than an English one (because of more possible verb inflections), a Russian one is more complex (more possible noun declensions), a Hebrew one is even more complex (due to non-catenative morphology and a writing system without vowels), and so on, while a stemmer for Hungarian would be easier to implement due to the precise rules in the language for flexion

## Multilingual Stemming

Multilingual stemming applies morphological rules of two or more languages simultaneously instead of rules for only a single language when interpreting a search query. Commercial systems using multilingual stemming exist.

# Applications

## Information Retrieval

Stemmers are common elements in IR systems, since a user who runs a query on “daffodils” would probably also be interested in documents that contain the word “daffodil” (without the s). The effectiveness of stemming for English query systems was soon found to be rather limited, however, and this has led early information retrieval researchers to deem stemming irrelevant in general. An alternative approach, based on

searching for n -grams rather than stems, may be used instead. Also, recent research has shown greater benefits for retrieval in other languages.

### **Commercial Uses.**

Many commercial companies have been using stemming since at least the 1980's and have produced algorithmic and lexical stemmers in many languages.

The Snowball stemmers have been compared with commercial lexical stemmers with varying results.

Google adopted word stemming in 2003. For example initially a search for "fish" would not have returned "fishing". Other software search algorithms vary in their use of word stemming. Programs that simply search for substrings obviously will find "fish" in "fishing" but when searching for "fishes" will not find occurrences of the word "fish".

## **Stemming Errors**

### **Two Types of Errors**

Natural languages are not completely regular constructs, and therefore stemmers operating on natural words inevitably make mistakes. On the one hand, words which ought to be merged together (such as "adhere" and "adhesion") may remain distinct after stemming; on the other, words which are really distinct may be wrongly conflated (e.g., "experiment" and "experience"). These are known as under stemming errors and over stemming errors respectively. By counting these errors for a sample of words, we can gain an insight into the operation of a stemmer, and compare different stemmers one with another.

To enable the errors to be counted, the words in the collection must already be organized into 'conceptual groups', containing words (such as "adhere", "adheres", "adhering", "adhesion", "adhesive") which ought all to be merged to the same stem. The research refers to two papers (Paice, 1994 and Paice, 1996) which describe this approach to stemmer evaluation in some detail.

In this document, we give simple examples to show how over stemming and under stemming indices can be computed. In principle, the idea is to compare every pair of words in the sample—though in fact, to avoid wasting lots of time, only certain pairs are actually considered. For each comparison, we will know whether the pair of words belongs to the same conceptual group, and whether they were in fact converted to the same stem:

If the two words belong to the same conceptual group, and are converted to the same stem, then the conflation is correct; if however they are converted to different stems, this is counted as an under stemming error.

If the two words belong to different conceptual groups, and remain distinct after stemming, then the stemmer has behaved correctly. If however they are converted to the same stem, this is counted as an over stemming error.

### Example

The following examples use the so-called truncate (n) stemmer, which simply retains the first n letters of the word, where n is a suitable integer, such as 4, 5 or 6. If the word has less than n letters to start with, it is returned unchanged.

Consider the seven words shown in the first column of the table below, which fall into two distinct conceptual groups. We will look at the effect of applying Truncate (5) and Truncate (4) to these words.

### Truncate(N)

FULL WORD	Truncate (5)	Truncate (4)
Divide	Divid	Divi
Dividing	Divid	Divi
Divided	Divid	Divi
Division	Divis	Divi
Divisor	Divis	Divi
Divine	Divin	Divi
Divination	Divin	Divi

### Under-Stemming Truncate(5)

	Divide	Dividing	Divided	Division	Divisor	Divine	Divination
Divide		1	1	0	0	X	X
Dividing	1		1	0	0	X	X
Divided	1	1		0	0	X	X
Division	0	0	0		1	X	X
Divisor	0	0	0	1		X	X
Divine	X	X	X	X	X		1
Divination	X	X	X	X	X	1	

1 = A pair of group able words with identical stems. This represents a successful stemming operation.

0 = A pair of group able words with non-identical stems. This represents an Under stemming error.

X = A pair of non-group able words. These pairs are not considered when counting under stemming errors.

In this case, the proportion of word pairs successfully merged is 10 out of 22, hence  $UI = 1 - (10/22) = 0.545$ .

### Over-Stemming Truncate(5)

	Divide	Dividing	Divided	Division	Divisor	Divine	Divination
Divide		X	X	X	X	1	1
Dividing	X		X	X	X	1	1
Divided	X	X		X	X	1	1
Division	X	X	X		X	1	1
Divisor	X	X	X	X		1	1
Divine	1	1	1	1	1		X
Divination	1	1	1	1	1	X	

1 = A pair of non-group able words with non-identical stems. This represents a successful stemming operation.

0 = A pair of non-group able words with identical stems. This represents an Over stemming error.

X = A pair of group able words. These pairs are not considered when counting Over stemming errors.

In this case, the proportion of word pairs which were correctly not merged to the same stem is 20 out of 20, hence  $OI = 1 - (20/20)$ ,  $OI = 0$ .

### Under-Stemming Truncate(4)

	Divide	Dividing	Divided	Division	Divisor	Divine	Divination
Divide		1	1	1	1	X	X
Dividing	1		1	1	1	X	X
Divided	1	1		1	1	X	X
Division	1	1	1		1	X	X
Divisor	1	1	1	1		X	X
Divine	X	X	X	X	X		1
Divination	X	X	X	X	X	1	

In this case, the proportion of word pairs correctly merged is 22 out of 22, hence  $UI = 1 - (22/22) = 0$ .

### Over-Stemming Truncate(4)

	Divide	Dividing	Divided	Division	Divisor	Divine	Divination
Divide		X	X	X	X	0	0
Dividing	X		X	X	X	0	0
Divided	X	X		X	X	0	0
Division	X	X	X		X	0	0
Divisor	X	X	X	X		0	0
Divine	0	0	0	0	0		X
Divination	0	0	0	0	0	X	

Here, the proportion of word pairs which were correctly not conflated is 0 out of 20, so that  $OI = 1 - (20/20) = 1$ .

### Discussion

There are some problems with this approach. Firstly, the manual construction of the grouped word collection is time-consuming, which limits the size of the word collections which can be used. Secondly, it is sometimes unclear whether or not two words should be grouped together - for instance, whether “different” and “differentiate” should be merged depends very much on the topic of the original text. Moreover, the question of whether or not a group will contain all of the words it should and none of the words that it should not is highly relative to the document being inspected.

An example of this would be an IR system containing two documents, one containing no words from the ‘Divine’ group, but various words from ‘Divide’ group, and other containing words from ‘Divine’ group, but none from the ‘Divide’ group. On testing for under stemming and over stemming errors using the Truncate (4) Stemmer on each document there are no errors, yet when a query is performed with the term ‘Divide’ both documents would be returned. This is obviously detrimental to precision due over stemming, but analysis based on the individual documents would not pick this up. It would be necessary for the entire database to be checked to find all such errors

## Stemming Performance

### Direct Assessment

The most primitive method for assessing the performance of a stemmer is to examine its behaviour when applied to samples of words - especially words which have already been arranged into ‘conflation groups’. This way, specific errors (e.g., failing to merge “maintained” with “maintenance”, or wrongly merging “experiment” with “experience”) can be identified, and the rules adjusted accordingly. This approach is of very limited

utility on its own, but can be used to complement other methods, such as the error-counting approach outlined later.

### **Information Retrieval**

The most obvious method for comparing the usefulness of Stemmers for the field of IR is by their impact on IR performance, using a testing system and a 'test collection' of documents, queries and relevance judgments. This involves substituting different Stemmers to see which gives the best results in terms of performance metrics such as Precision and Recall. However, there are problems with using such a technique for deciding which stemmer to use in an IR system, since the results are frequently indecisive. Thus, the 'best' Stemmer may be different for different databases and different searches.

### **Error Counting**

It is possible to evaluate stemming by counting the numbers of two kinds of errors that occur during stemming, namely;

#### **Under-Stemming.**

This refers to words that should be grouped together by stemming, but aren't. This causes a single concept to be spread over various different stems, which will tend to decrease the Recall in an IR search.

#### **Over-Stemming**

This refers to words that shouldn't be grouped together by stemming, but are. This causes the meanings of the stems to be diluted, which will affect Precision of IR.

Using a sample file of grouped words, these errors are then counted. This evaluation method was described by Chris Paice in a paper entitled 'Method for Evaluation of Stemming Algorithms Based on Error Counting' JASIS 47(8), August 1996, 632-649. This method returns a value for an Under-Stemming (or Conflation) index;

UI = Under-Stemming Index

CI = Conflation Index: proportion of equivalent word pairs which were successfully grouped to the same stem.

$$UI = 1 - CI$$

Also a value for an Over-Stemming (or Distinctness) index;

OI = Over-Stemming index

DI = Distinctness Index: proportion of non-equivalent word pairs which remained distinct after stemming.

OI= 1 - DI

The purpose of this error-counting approach is that, although it is advantageous to have the index of terms compressed, this is only useful up to a point. This is because, as conflation becomes 'heavier', the merging of distinct concepts becomes increasingly frequent. At this point, small increases in Recall are gained at the expense of a major loss of Precision.

One question mark over this approach concerns the validity of the grouped file against which the errors are assessed. In Paice (1996), these grouped files were constructed by human judgment, during scrutiny of sample word lists.

## Stemmer Strength

A 'weak' or 'light' stemmer is one which only merges a few of the most highly related words together - for example, just singular and plural forms, or inflective variants of verbs ("react", "reacts", "reacting", "reacted"). A 'strong' or 'heavy' stemmer, on the other hand, merges a much wider variety of forms (... "reaction", "reactions", "reactive", "reactivity", "reactant" etc.). In IR searching, light stemming seems to be most effective; heavy stemming increases the chances of ambiguity and confusion (for instance, merging "author" with "authority" is generally not helpful). In other situations - e.g., for displaying terms in a user interface - a heavier stemmer may be quite useful.

Whether the impact of a stemmer is positive depends not only on its strength, but also on whether it performs its task accurately - stemmer strength metrics do not represent actual stemming accuracy.

### Measurement of stemmer strength.

There are the following ways to measure stemmer strength:

#### 1. Number of words per conflation class

This is the average size of the groups of words converted to a particular stem (regardless of whether they are all correct). Thus, if the words "engineer", "engineering" and "engineered", and no others, were all stemmed to the stem "engineer", then the size of that conflation class would be 3. If the conflation of 1,000 different words resulted in 250 distinct stems, then the mean number of words per conflation class would be 4.

This metric is obviously dependent on the number of words processed, but for a word collection of given size, a higher value indicates a heavier stemmer. The value is easily calculated as follows:

$$WC = \text{Mean number of words per conflation class}$$

N = Number of unique words before Stemming

S = Number of unique stems after Stemming

$$\text{MEAN WC} = N/S$$

## 2. Index Compression

The Index Compression Factor represents the extent to which a collection of unique words is reduced (compressed) by stemming, the idea being that the heavier the Stemmer, the greater the Index Compression Factor. This can be calculated by;

IC = Index Compression Factor

N = Number of unique words before Stemming

S = Number of unique stems after Stemming

$$\text{IC} = (N - S)/N$$

## 3. The Word Change Factor

This is a simply the proportion of the words in a sample that have been changed in any way by the stemming process, the idea being that the larger the number of words that are altered, the greater the strength of the Stemmer.

## 4. Number of Characters Removed

There are various metrics which use the principle that strong stemmers remove more characters from words than weaker stemmers. One way is to compute the average number of letters removed when a stemmer is applied to a text collection. Thus, suppose that the nine words “react”, “reacts”, “reacting”, “reacted”, “reaction”, “reactions”, “reactive”, “reactivity” and “reactivities” are all reduced to “react”; the numbers of characters removed are then 0, 1, 3, 2, 3, 4, 3, 5 and 7, respectively. This gives a mean removal rate of  $28/9 = 3.11$ .

Obtaining such data for a large collection of words enables us to compute not only the mean but also the mode and the median of the resulting distribution, in case these seem more useful.

The meaning of ‘number of characters removed’ is not clear-cut for those stemmers who replace endings rather than just removing them. Thus, a stemmer may replace the endings “-ies” and “-ied” by “-y”. We could count this as 3 (number of letters removed),

or 2 (reduction in length), or even 4 (letters removed + letters added). Alternatively, we could use a metric such as the Hamming Distance, or Modified Hamming Distance, as discussed next.

## 5. Hamming Distance

The Hamming Distance takes two strings of equal length and counts the number of corresponding positions where the characters are different. If the strings are of different lengths, we can use the Modified Hamming Distance, MHD. Thus, suppose the string lengths are  $P$  and  $Q$ , where  $P < Q$ ,

we use the formula  $MHD = HD(1,P) + (Q-P)$

where  $HD(1, P)$  is the Hamming Distance for the first  $P$  characters of both strings. Applying this to a stemmer, suppose that the word “parties” is converted to “party”. In this case,  $P=5$  and  $Q=7$ , so that  $HD(1,P) = 1$  by comparing “parti” with “party”, and  $(Q-P) = 2$ , giving  $MHD = 3$ . Clearly, we can compute the average MHD value for every word in the original sample.

## 6. Inter-Stemmer Similarity

It is possible to compare two separate stemming algorithms by comparing the output they produce. This provides a measure of the similarity (or conversely, the distance) between the two algorithms. The approach is to take a set of words and apply both algorithms in turn, thus producing two output lists. Corresponding stems in the two output lists are then compared to give a measure of similarity between the stemmers. This could be useful, for example, for deciding whether two separate implementations of the same algorithm really do the same thing (for example, it can be used for comparing the various available implementations of Porter’s algorithm).

A simple way to measure inter-stemmer similarity is by simply counting the proportion of the stem-pairs which are identical. A more sensitive measure, however, would be one based on the average Modified Hamming Distance between the pairs of stems. This will give a measure of inter-stemmer distance, so (as suggested by Frakes & Fox) it may be more useful to use the inverse of the average MHD as a similarity value.

As with stemmer strength, inter-stemmer similarity is not directly related to accuracy. Thus, you could have two stemmers which are very dissimilar and yet which are virtually identical in their ability to conflate related words. To see this, suppose you take a particular stemmer, and modify it so that it systematically recodes the last 3 letters of each stem (e.g., “A” to “B”, “B” to “C”, “C” to “D”, etc.). The modified stemmer would group a collection of words in exactly the same way as the original, but the similarity between the two would be quite low. A different kind of metric would be needed to notice the similarity in this case!

## 7. Similarity Metric

The following metric was developed by Chris O Neill during the course of a project produced under the supervision of him during 2001.

This approach starts by considering the average Hamming distance SSM between the corresponding stems in two lists, where:

A and B are the stemmers being compared,

HD is the Modified Hamming distance between two strings, and

N is the number of words in the word list.

$$SSM(A, B) = \frac{\sum_{i=1}^N HD(A_i, B_i)}{N}$$

However, this approach leaves the following problems:

It would be better if the result could be normalized to give a value between 1 and 0. The above metric fails to take into consideration the overall lengths of the two strings. Thus, removing 's' from the word 'reds' to give 'red', is a removal of 25% of the word, and would have a Hamming distance of 1. Removing the suffix's' from the word 'methylene dioxymethamphetamins' to give 'methylenedioxymethamphetamin' is a removal of 3.4% of the word, yet this also gives a modified Hamming distance of 1.

A new metric was therefore developed which, though still based on the Hamming distance, also takes into account the lengths of the words being compared. It uses the idea that there is a maximum and minimum modified Hamming distance that can obtain between two strings. The minimum distance is zero, when both strings are identical, whereas the maximum distance MD is the length of the longer string. The relative distance RD is obtained from the modified hamming distance MHD thus:

$$RD(X, Y) = MHD(X, Y) / MD(XY)$$

Therefore, by calculating the sum of the relative distances for all pairs of stems, then dividing this by the number of words, an average distance metric is obtained. This was then further developed by subtracting the value from one, then multiplying it by 100 to give a percentage, so that 0% represents no similarity and 100% total similarity. Thus, the new stemmer similarity metric is given by the formula:

$$SSM(U, V) = 100 \left[ \left( \frac{\sum_w \left( \frac{MHD(Uw, Vw)}{MD(Uw, Vw)} \right)}{N} \right) - 1 \right]$$

Where

U & V are the Stemmers being compared,

N = Number of words in the sample

MHD = Modified Hamming Distance

MD = Maximum Distance

## REFERENCES

- Adamson, G.W. & Boreham, J., 1974: "The use of an association measure based on character structure to identify semantically related pairs of words and document titles," *Information Processing & Management* **10**(7/8), 253-260.
- Arlberger and V. Kann. 1999. Implementing an efficient part-of-speech tagger, *Software Practice and Experience*, 29, 815-832, 1999.
- Church, K.W., 1995: "One term or two?" in E.A. Fox, P. Ingwersen & R. Fidel (eds.), *Proceedings of the 18<sup>th</sup> ACM SIGIR conference held at Seattle, WA, July 9-13, 1995*; pp.310-318.
- D. Harman. 1991. How effective is suffixing? *Journal of the American Society for Information Science*, 42(1): 7-15.
- Dawson, J.L., 1974: "Suffix removal for word conflation," *Bulletin of the Association for Literary & Linguistic Computing*, **2**(3), 33-46.
- Frakes, W.B. & Baeza-Yates, R., 1992: *Information Retrieval: Data Structures & Algorithms*. Englewood Cliffs, NJ: Prentice-Hall. Chapter 8.
- D.A. Hull. 1996. Stemming Algorithms - A Case Study for Detailed Evaluation. *Journal of the American Society for Information Science*, 47(1): 70-84
- Hafer, M.A. & Weiss, S.F., 1974: "Word segmentation by letter successor varieties", *Information Processing & Management* 10(11/12), 371-386.
- R. Krovetz. 1993. Viewing Morphology as an Inference Process. In *Proceedings of the 16<sup>th</sup> Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, New York, pp 191-202.
- Harman, D., 1991: "How effective is suffixing?" *Journal of the American Society for Information Science* **42** (1), 7-15.
- W. Kraaij and R.Pohlmann. 1994. Porter's stemming algorithm for Dutch. In L.G.M. Noordman and W.A.M. de Vroomen, editors, *Informatie wetenschap 1994: Wetenschappelijke bijdragen aan de derde STINFON Conferentie*, pp. 167-180.
- Hull, D.A., 1996: "Stemming algorithms: a case study for detailed evaluation", *Journal of the American Society for Information Science*, **47**(1), 70-84.
- M.F. Porter. 1980. An algorithm for suffix stripping. *Program*, vol 14, no 3, pp 130-130.
- Xu and W. B. Croft. 1998. Corpus-based Stemming using Co-occurrence of Word Variants. *ACM Transactions on Information Systems*, Volume 16, Number 1, pp 61-81, January 1998.

- Kraaij, W. & Pohlmann, R., 1996: "Viewing stemming as recall enhancement," in H-P. Frei, D. Harman, P. Schauble & R. Wilkinson (eds.), Proceedings of the 17<sup>th</sup> ACM SIGIR conference held at Zurich, August 18-22, 1996; pp.40-48.
- M. Hassel. 2001. Internet as Corpus – Automatic Construction of a Swedish News Corpus. NODALIDA '01 - 13<sup>th</sup> Nordic Conference on Computational Linguistics, May 21-22 2001, Uppsala, Sweden
- Krovetz, R., 1993: "Viewing morphology as an inference process", in R. Korfhage, E. Rasmussen & P. Willett (eds.), and Proceedings of the 16<sup>th</sup> ACM SIGIR conference held at Pittsburgh, PA, June 27-July 1, 1993; pp.191-202.
- M. Popovic and P. Willett. 1992. The effectiveness of stemming for natural-language access to Slovene textual data. Journal of the American Society for Information Science, 43(5): 384-390.
- Lennon, M., Pierce, D.S., Tarry, B.D. and Willett, P., 1981: "An evaluation of some conflation algorithms for information retrieval". Journal of Information Science **3**, 177-183.
- Lovins, J.B., 1968: "Development of a stemming algorithm". Mechanical Translation and Computational Linguistics **11**, 22-31
- Lovins, J.B., 1971: "Error evaluation for stemming algorithms as clustering algorithms," Journal of the American Society for Information Science, **22**(1), 28-40.
- Paice, C.D., 1990: "Another stemmer", SIGIR Forum, **24**(3), 56-61 (Fall 1990).
- Paice, C.D., 1994: "An evaluation method for stemming algorithms", in Croft, W.B. & van Rijsbergen, C.J. (eds.), Proceedings of the 17<sup>th</sup> ACM SIGIR conference held at Dublin, July 3-6, 1994; pp. 42-50.
- Paice, C.D., 1996: "A method for the evaluation of stemming algorithms based on error counting," Journal of the American Society for Information Science, **47**(8), 632-649.
- Popovic, M. and Willett, P., 1992: "The effectiveness of stemming for natural language access to Slovene textual data," Journal of the American Society for Information Science, 43(5), 384-390.
- Porter, M.F., 1980: "An algorithm for suffix stripping". Program **14**,130-137.
- Savoy, J., 1993: "Stemming of French words based on grammatical categories" Journal of the American Society for Information Science, 44(1), 1-9.
- Ulmschneider, J.E. & Doszkocs, 1983: "A practical stemming algorithm for online search assistance," Online Review, **7**(4), 301-318.
- Xu, J. & Croft, W.B., 1998: "Corpus-based stemming using cooccurrence of word variants," ACM Transactions on Information Systems, **16**(1), 61-81.
- Stemming algorithms research. [www.cranfield.ac.uk/research](http://www.cranfield.ac.uk/research).
- Key word stemming. [www.cybertouch.info](http://www.cybertouch.info)
- Stemming technology research. [www.vlex.be](http://www.vlex.be)
- Lovins, Julie B. Development of a Stemming Algorithm, Electronic Systems Laboratory, MIT, USA.