ISSN:2229-6093



# A Comparative Study of Stemming Algorithms

Ms. Anjali Ganesh Jivani
*Department of Computer Science & Engineering*
*The Maharaja Sayajirao University of Baroda*
*Vadodara, Gujarat, India*
*anjali_jivani@yahoo.com*

## Abstract

*Stemming is a pre-processing step in Text Mining applications as well as a very common requirement of Natural Language processing functions. In fact it is very important in most of the Information Retrieval systems. The main purpose of stemming is to reduce different grammatical forms / word forms of a word like its noun, adjective, verb, adverb etc. to its root form. We can say that the goal of stemming is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. In this paper we have discussed different methods of stemming and their comparisons in terms of usage, advantages as well as limitations. The basic difference between stemming and lemmatization is also discussed.*

***Keywords-*** *stemming; text mining; NLP; IR; suffix*

## 1. Introduction

Word stemming is an important feature supported by present day indexing and search systems. Indexing and searching are in turn part of Text Mining applications, Natural Language Processing (NLP) systems and Information Retrieval (IR) systems. The main idea is to improve recall by automatic handling of word endings by reducing the words to their word roots, at the time of indexing and searching. Recall in increased without compromising on the precision of the documents fetched. Stemming is usually done by removing any attached suffixes and prefixes (affixes) from index terms before the actual assignment of the term to the index. Since the stem of a term represents a broader concept than the original term, the stemming process eventually increases the number of retrieved documents in an IR system. Text clustering, categorization and summarization also require this conversion as part of the pre-processing before actually applying any related algorithm.

## 2. Working of a Stemmer

It has been seen that most of the times the morphological variants of words have similar semantic interpretations and can be considered as equivalent for the purpose of IR applications. Since the meaning is same but the word form is different it is necessary to identify each word form with its base form. To do this a variety of stemming algorithms have been developed. Each algorithm attempts to convert the morphological variants of a word like introduction, introducing, introduces etc. to get mapped to the word 'introduce'. Some algorithms may map them to just 'introduc', but that is allowed as long as all of them map to the same word form or more popularly known as the stem form. Thus, the key terms of a query or document are represented by stems rather than by the original words. The idea is to reduce the total number of distinct terms in a document or a query which in turn will reduce the processing time of the final output.

## 3. Stemming and Lemmatizing

The basic function of both the methods – stemming and lemmatizing is similar. Both of them reduce a word variant to its 'stem' in stemming and 'lemma' in lemmatizing. There is a very subtle difference between both the concepts. In stemming the 'stem' is obtaining after applying a set of rules but without bothering about the part of speech (POS) or the context of the word occurrence. In contrast, lemmatizing deals with obtaining the 'lemma' of a word which involves reducing the word forms to its root form after understanding the POS and the context of the word in the given sentence.

In stemming, conversion of morphological forms of a word to its stem is done assuming each one is semantically related. The stem need not be an existing word in the dictionary but all its variants should map to this form after the stemming has been completed. There are two points to be considered while using a stemmer:

- Morphological forms of a word are assumed to have the same base meaning and hence should be mapped to the same stem
- Words that do not have the same meaning should be kept separate

These two rules are good enough as long as the resultant stems are useful for our text mining or language processing applications. Stemming is generally considered as a recall-enhancing device. For languages with relatively simple morphology, the influence of stemming is less than for those with a more complex morphology. Most of the stemming experiments done so far are for English and other west European languages.

Lemmatizing deals with the complex process of first understanding the context, then determining the POS of a word in a sentence and then finally finding the 'lemma'. In fact an algorithm that converts a word to its linguistically correct root is called a lemmatizer. A lemma in morphology is the canonical form of a lexeme. Lexeme, in this context, refers to the set of all the forms that have the same meaning, and lemma refers to the particular form that is chosen by convention to represent the lexeme.

In computational linguistics, a stem is the part of the word that never changes even when morphologically inflected, whilst a lemma is the base form of the verb. Stemmers are typically easier to implement and run faster, and the reduced accuracy may not matter for some applications. Lemmatizers are difficult to implement because they are related to the semantics and the POS of a sentence. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. The results are not always morphologically right forms of words. Nevertheless, since document index and queries are stemmed "invisibly" for a user, this peculiarity should not be considered as a flaw, but rather as a feature distinguishing stemming from lemmatization. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the lemma.

For example, the word inflations like gone, goes, going will map to the stem 'go'. The word 'went' will not map to the same stem. However a lemmatizer will map even the word 'went' to the lemma 'go'.

**Stemming:**
introduction, introducing, introduces – introduc
gone, going, goes – go

**Lemmatizing:**
introduction, introducing, introduces – introduce
gone, going, goes, went – go

## 4. Errors in Stemming

There are mainly two errors in stemming – over stemming and under stemming. Over-stemming is when two words with different stems are stemmed to the same root. This is also known as a false positive. Under-stemming is when two words that should be stemmed to the same root are not. This is also known as a false negative. Paice has proved that light-stemming reduces the over-stemming errors but increases the under-stemming errors. On the other hand, heavy stemmers reduce the under-stemming errors while increasing the over-stemming errors [14, 15].

## 5. Classification of Stemming Algorithms

Broadly, stemming algorithms can be classified in three groups: truncating methods, statistical methods, and mixed methods. Each of these groups has a typical way of finding the stems of the word variants. These methods and the algorithms discussed in this paper under them are shown in the Fig. 1.

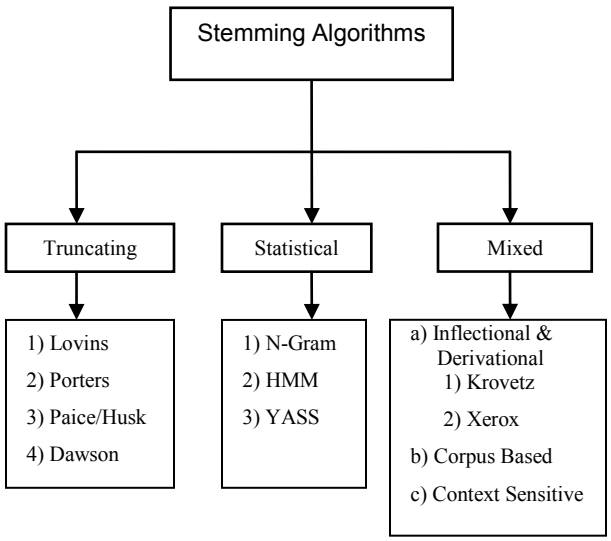


**Figure 1. Types of stemming algorithms**

### 5.1. Truncating Methods (Affix Removal)

As the name clearly suggests these methods are related to removing the suffixes or prefixes (commonly known as affixes) of a word. The most basic stemmer

was the Truncate (n) stemmer which truncated a word at the nth symbol i.e. keep n letters and remove the rest. In this method words shorter than n are kept as it is. The chances of over stemming increases when the word length is small.

Another simple approach was the S-stemmer – an algorithm conflating singular and plural forms of English nouns. This algorithm was proposed by Donna Harman. The algorithm has rules to remove suffixes in plurals so as to convert them to the singular forms [7].

**Lovins Stemmer**

This was the first popular and effective stemmer proposed by Lovins in 1968. It performs a lookup on a table of 294 endings, 29 conditions and 35 transformation rules, which have been arranged on a longest match principle [6]. The Lovins stemmer removes the longest suffix from a word. Once the ending is removed, the word is recoded using a different table that makes various adjustments to convert these stems into valid words. It always removes a maximum of one suffix from a word, due to its nature as single pass algorithm.

The advantages of this algorithm is it is very fast and can handle removal of double letters in words like 'getting' being transformed to 'get' and also handles many irregular plurals like – mouse and mice, index and indices etc.

Drawbacks of the Lovins approach are that it is time and data consuming. Furthermore, many suffixes are not available in the table of endings. It is sometimes highly unreliable and frequently fails to form words from the stems or to match the stems of like-meaning words. The reason being the technical vocabulary being used by the author.

**Porters Stemmer**

Porters stemming algorithm [17, 18] is as of now one of the most popular stemming methods proposed in 1980. Many modifications and enhancements have been done and suggested on the basic algorithm. It is based on the idea that the suffixes in the English language (approximately 1200) are mostly made up of a combination of smaller and simpler suffixes. It has five steps, and within each step, rules are applied until one of them passes the conditions. If a rule is accepted, the suffix is removed accordingly, and the next step is performed. The resultant stem at the end of the fifth step is returned.

The rule looks like the following:

<condition> <suffix> → <new suffix>

For example, a rule (m>0) EED → EE means "if the word has at least one vowel and consonant plus EED ending, change the ending to EE". So "agreed" becomes "agree" while "feed" remains unchanged. This algorithm has about 60 rules and is very easy to comprehend.

Porter designed a detailed framework of stemming which is known as 'Snowball'. The main purpose of the framework is to allow programmers to develop their own stemmers for other character sets or languages. Currently there are implementations for many Romance, Germanic, Uralic and Scandinavian languages as well as English, Russian and Turkish languages.

Based on the stemming errors, Paice reached to a conclusion that the Porter stemmer produces less error rate than the Lovins stemmer. However it was noted that Lovins stemmer is a heavier stemmer that produces a better data reduction [1]. The Lovins algorithm is noticeably bigger than the Porter algorithm, because of its very extensive endings list. But in one way that is used to advantage: it is faster. It has effectively traded space for time, and with its large suffix set it needs just two major steps to remove a suffix, compared with the five of the Porter algorithm.

**Paice/Husk Stemmer**

The Paice/Husk stemmer is an iterative algorithm with one table containing about 120 rules indexed by the last letter of a suffix [14]. On each iteration, it tries to find an applicable rule by the last character of the word. Each rule specifies either a deletion or replacement of an ending. If there is no such rule, it terminates. It also terminates if a word starts with a vowel and there are only two letters left or if a word starts with a consonant and there are only three characters left. Otherwise, the rule is applied and the process repeats.

The advantage is its simple form and every iteration taking care of both deletion and replacement as per the rule applied.

The disadvantage is it is a very heavy algorithm and over stemming may occur.

**Dawson Stemmer**

This stemmer is an extension of the Lovins approach except that it covers a much more comprehensive list of about 1200 suffixes. Like Lovins it too is a single pass stemmer and hence is pretty fast. The suffixes are stored in the reversed order indexed by their length and last letter. In fact they are organized as a set of branched character trees for rapid access.

The advantage is that it covers more suffixes than Lovins and is fast in execution.

The disadvantage is it is very complex and lacks a standard reusable implementation.

### 5.2. Statistical Methods

These are the stemmers who are based on statistical analysis and techniques. Most of the methods remove the affixes but after implementing some statistical procedure.

**N-Gram Stemmer**

This is a very interesting method and it is language independent. Over here string-similarity approach is used to convert word inflation to its stem. An n-gram is a string of n, usually adjacent, characters extracted from a section of continuous text. To be precise an n-gram is a set of n consecutive characters extracted from a word. The main idea behind this approach is that, similar words will have a high proportion of n-grams in common. For n equals to 2 or 3, the words extracted are called digrams or trigrams, respectively. For example, the word 'INTRODUCTIONS' results in the generation of the digrams:

*I, IN, NT, TR, RO, OD, DU, UC, CT, TI, IO, ON, NS, S*

and the trigrams:

**I, *IN, INT, NTR, TRO, ROD, ODU, DUC, UCT, CTI, TIO, ION, ONS, NS*, S**

Where '*' denotes a padding space. There are n+1 such digrams and n+2 such trigrams in a word containing n characters.

Most stemmers are language-specific. Generally a value of 4 or 5 is selected for n. After that a textual data or document is analyzed for all the n-grams. It is obvious that a word root generally occurs less frequently than its morphological form. This means a word generally has an affix associated with it. A typical statistical analysis based on the inverse document frequency (IDF) can be used to identify them.

This stemmer has an advantage that it is language independent and hence very useful in many applications.

The disadvantage is it requires a significant amount of memory and storage for creating and storing the n-grams and indexes and hence is not a very practical approach.

**HMM Stemmer**

This stemmer is based on the concept of the Hidden Markov Model (HMMs) which are finite-state automata where transitions between states are ruled by probability functions. At each transition, the new state emits a symbol with a given probability. This model was proposed by Melucci and Orio [12].

This method is based on unsupervised learning and does not need a prior linguistic knowledge of the dataset. In this method the probability of each path can be computed and the most probable path is found using the Viterbi coding in the automata graph.

In order to apply HMMs to stemming, a sequence of letters that forms a word can be considered the result of a concatenation of two subsequences: a prefix and a suffix. A way to model this process is through an HMM where the states are divided in two disjoint sets: initial can be the stems only and the later can be the stems or suffixes. Transitions between states define word building process. There are some assumptions that can be made in this method:

1. Initial states belong only to the stem-set - a word always starts with a stem
2. Transitions from states of the suffix-set to states of the stem-set always have a null probabiliy - a word can be only a concatenation of a stem and a suffix.
3. Final states belong to both sets - a stem can have a number of different derivations, but it may also have no suffix.

For any given word, the most probable path from initial to final states will produce the split point (a transition from roots to suffixes). Then the sequence of characters before this point can be considered as a stem.

The advantage of this method is it is unsupervised and hence knowledge of the language is not required.

The disadvantage is it is a little complex and may over stem the words sometimes.

**YASS Stemmer**

The name is an acronym for Yet Another Suffix Striper. This stemmer was proposed by Prasenjit Majumder, et. al. [20]. According to the authors the performance of a stemmer generated by clustering a lexicon without any linguistic input is comparable to that obtained using standard, rule-based stemmers such as Porter's. This stemmer comes under the category of statistical as well as corpus based. It does not rely on linguistic expertise. Retrieval experiments by the authors on English, French, and Bengali datasets show that the proposed approach is effective for languages that are primarily suffixing in nature.

The clusters are created using hierarchical approach and distance measures. Then the resulting clusters are considered as equivalence classes and their centroids as the stems. As per the details given in [20], the edit distance and YASS distance calculations for two string comparisons is shown in Fig. 2 and Fig. 3 of [20]. The YASS distance measures $D_1$, $D_2$, $D_3$ and $D_4$ are based on a Boolean function pi for penalty. It is defined as:

$$p_i = \begin{cases} 0 \text{ if } x_i = y_i \;\; 0 \le i \le \min(n, n') \\ 1 \text{ otherwise} \end{cases}$$

Where X and Y are two strings, $X = x_0x_1x_2 \ldots x_n$ and $Y = y_0y_1y_2 \ldots y_n$. If the strings are of unequal lengths we pad the shorter string with null characters to make the strings lengths equal. Smaller the distance measure indicates greater similarity between the strings. The edit distance between two strings of characters is the number of operations required to transform one of them into the other.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | s | t | r | o | n | o | m | e | r | x | x | x | x |
| a | s | t | r | o | n | o | m | i | c | a | l | l | y |

$D_1 = \frac{1}{2^8} + \frac{1}{2^9} + \ldots + \frac{1}{2^{13}} = 0.0077$

$D_2 = \frac{1}{8} \times (\frac{1}{2^0} + \ldots + \frac{1}{2^{13-8}}) = 0.2461$

$D_3 = \frac{6}{8} \times (\frac{1}{2^0} + \ldots + \frac{1}{2^{13-8}}) = 1.4766$

$D_4 = \frac{6}{14} \times (\frac{1}{2^0} + \ldots + \frac{1}{2^{13-8}}) = 0.8438$

Edit distance = 6

**Figure 2. Calculation of distance measures - 1**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| a | s | t | r | o | n | o | m | e | r |
| a | s | t | o | n | i | s | h | x | x |

$D_1 = \frac{1}{2^3} + \ldots + \frac{1}{2^9} = 0.2480$

$D_2 = \frac{1}{3} \times (\frac{1}{2^0} + \ldots + \frac{1}{2^{9-3}}) = 0.6615$

$D_3 = \frac{7}{3} \times (\frac{1}{2^0} + \ldots + \frac{1}{2^{9-3}}) = 4.6302$

$D_4 = \frac{7}{10} \times (\frac{1}{2^0} + \ldots + \frac{1}{2^{9-3}}) = 1.3891$

Edit distance = 5

**Figure 3. Calculation of distance measures - 2**

As per the distances $D_1$, $D_2$, $D_3$ and $D_4$ it can be seen that astronomer and astronomically are more similar than astronomer and astonish. The edit distance shows exactly opposite which means the new distance measures are more accurate.

### 5.3. Inflectional and Derivational Methods

This is another approach to stemming and it involves both the inflectional as well as the derivational morphology analysis. The corpus should be very large to develop these types of stemmers and hence they are part of corpus base stemmers too. In case of inflectional the word variants are related to the language specific syntactic variations like plural, gender, case, etc whereas in derivational the word variants are related to the part-of-speech (POS) of a sentence where the word occurs.

**Krovetz Stemmer (KSTEM)**

The Krovetz stemmer was presented in 1993 by Robert Krovetz [10] and is a linguistic lexical validation stemmer. Since it is based on the inflectional property of words and the language syntax, it is very complicated in nature. It effectively and accurately removes inflectional suffixes in three steps:

1. Transforming the plurals of a word to its singular form
2. Converting the past tense of a word to its present tense
3. Removing the suffix 'ing'

The conversion process first removes the suffix and then through the process of checking in a dictionary for any recoding, returns the stem to a word. The dictionary lookup also performs any transformations that are required due to spelling exception and also converts any stem produced into a real word, whose meaning can be understood.

The strength of derivational and inflectional analysis is in their ability to produce morphologically correct stems, cope with exceptions, processing prefixes as well as suffixes. Since this stemmer does not find the stems for all word variants, it can be used as a pre-stemmer before actually applying a stemming algorithm. This would increase the speed and effectiveness of the main stemmer. Compared to Porter and Paice / Husk, this is a very light stemmer. The Krovetz stemmer attempts to increase accuracy and robustness by treating spelling errors and meaningless stems.

If the input document size is large this stemmer becomes weak and does not perform very effectively. The major and obvious flaw in dictionary-based algorithms is their inability to cope with words, which are not in the lexicon. Also, a lexicon must be manually created in advance, which requires significant efforts. This stemmer does not consistently produce a good recall and precision performance.

**Xerox Inflectional and Derivational Analyzer**

The linguistics groups at Xerox have developed a number of linguistic tools for English which can be used in information retrieval. In particular, they have produced English lexical database which provides a morphological analysis of any word in the lexicon and identifies the base form. Xerox linguists have developed a lexical database for English and some other languages also which can analyze and generate inflectional and derivational morphology. The inflectional database reduces each surface word to the form which can be found in the dictionary, as follows [23]:

- nouns singular (e.g. children child)
- verbs infinitive (e.g. understood understand)
- adjectives positive form (e.g. best good)
- pronoun nominative (e.g. whom who)

The derivational database reduces surface forms to stems which are related to the original in both form and

semantics. For example, 'government' stems to 'govern' while 'department' is not reduced to 'depart' since the two forms have different meanings. All stems are valid English terms, and irregular forms are handled correctly. The derivational process uses both suffix and prefix removal, unlike most conventional stemming algorithms which rely solely on suffix removal. A sample of the suffixes and prefixes that are removed is given below [23]:

- Suffixes: ly, ness, ion, ize, ant, ent, ic, al, Ic, ical, able, ance, ary, ate, ce, y, dom, ee, eer, ence, ency, ery, ess, ful, hood, ible, icity, ify, ing, ish, ism, ist, istic, ity, ive, less, let, like, ment, ory, ous, ty, ship, some, ure

- Prefixes: anti, bi, co, contra, counter, de, di, dis, en, extra, in, inter, intra, micro, mid, mini, multi, non, over, para, poly, post, pre, pro, re, semi, sub, super, supra, sur, trans, tn, ultra, un

The databases are constructed using finite state transducers, which promotes very efficient storage and access. This technology also allows the conflation process to act in reverse, generating all conceivable surface forms from a single base form. The database starts with a lexicon of about 77 thousand base forms from which it can generate roughly half a million surface forms.

The advantages of this stemmer are that it works well with a large document also and removes the prefixes also where ever applicable. All stems are valid words since a lexical database which provides a morphological analysis of any word in the lexicon is available for stemming. It has proved to work better than the Krovetz stemmer for a large corpus.

The disadvantage is that the output depends on the lexical database which may not be exhaustive. Since this method is based on a lexicon, it cannot correctly stem words which are not part of the lexicon. This stemmer has not been implemented successfully on many other languages. Dependence on the lexicon makes it a language dependent stemmer.

### 5.4. Corpus Based Stemmer

This method of stemming was proposed by Xu and Croft in their paper "Corpus-based stemming using co-occurrence of word variants" [22]. They have suggested an approach which tries to overcome some of the drawbacks of Porter stemmer. For example, the words 'policy' and 'police' are conflated though they have a different meaning but the words 'index' and 'indices' are not conflated though they have the same root. Porter stemmer also generates stems which are not real words like 'iteration' becomes 'iter' and 'general' becomes 'gener'. Another problem is that while some stemming algorithms may be suitable for one corpus, they will produce too many errors on another.

Corpus based stemming refers to automatic modification of conflation classes – words that have resulted in a common stem, to suit the characteristics of a given text corpus using statistical methods. The basic hypothesis is that word forms that should be conflated for a given corpus will co-occur in documents from that corpus. Using this concept some of the over stemming or under stemming drawbacks are resolved e.g. 'policy' and 'police' will no longer be conflated.

To determine the significance of word form co-occurrence, the statistical measure used in [22],

$$Em(a, b) = nab / (na + nb)$$

Where, a and b are a pair of words, na and nb are the number of occurrences of a and b in the corpus, nab is the number of times both a and b fall in a text window of size win in the corpus (they co-occur).

The way this stemmer works is to first use the Porter stemmer to identify the stems of conflated words and then the next step is to use the corpus statistics to redefine the conflation. Sometimes the Krovetz stemmer (KSTEM) along with Porter stemmer is used in the initial stem to make sure that word conflations are not missed out.

The advantage of this method is it can potentially avoid making conflations that are not appropriate for a given corpus and the result is an actual word and not an incomplete stem.

The disadvantage is that you need to develop the statistical measure for every corpus separately and the processing time increases as in the first step two stemming algorithms are first used before using this method.

### 5.5. Context Sensitive Stemmer

This is a very interesting method of stemming unlike the usual method where stemming is done before indexing a document, over here for a Web Search, context sensitive analysis is done using statistical modeling on the query side. This method was proposed by Funchun Peng et. al. [4].

Basically for the words of the input query, the morphological variants which would be useful for the search are predicted before the query is submitted to the search engine. This dramatically reduces the number of bad expansions, which in turn reduces the cost of additional computation and improves the precision at the same time.

After the predicted word variants from the query have been derived, a context sensitive document matching is done for these variants. This conservative strategy serves as a safeguard against spurious

stemming, and it turns out to be very important for improving precision.

This stemming process is divided into four steps [4] after the query is fired:

**Candidate generation:**

Over here the Porter stemmer is used generate the stems from the query words. This has absolutely no relation to the semantics of the words. For a better output the corpus-based analysis based on distributional similarity is used. The rationale of using distributional word similarity is that true variants tend to be used in similar contexts. In the distributional word similarity calculation, each word is represented with a vector of features derived from the context of the word. We use the bigrams to the left and right of the word as its context features, by mining a huge Web corpus. The similarity between two words is the cosine similarity between the two corresponding feature vectors.

**Query Segmentation and head word detection:**

When the queries are long, it is important to detect the main concept of the query. The query is broken into segments which are generally the noun phrases. For each noun phrase the most important word is detected which is the head word. Sometimes a word is split to know the context. The mutual information of two adjacent words is found and if it passes a threshold value, they are kept in the same segment. Finding the headword is by using a syntactical parser.

**Context sensitive word expansion:**

Now that the head words are obtained, using probability measures it is decided which word variants would be most useful – generally they are the plural forms of the words. This is done using the simplest and most successful approach to language modeling which is the one based on the *n*-gram model which uses the chain rule of probability. In this step all the important head word variants are obtained. The traditional way of using stemming for Web search, is referred as the naive model. This is to treat every word variant equivalent for all possible words in the query. The query "book store" will be transformed into "*(book OR books)(store OR stores)*" when limiting stemming to pluralization handling only, where *OR* is an operator that denotes the equivalence of the left and right arguments.

**Context sensitive document matching:**

Now that we have the word variants, in this step a variant match is considered valid only if the variant occurs in the same context in the document as the original word does in the query. The context is the left or the right non-stop segments of the original word. Considering the fact that queries and documents may not represent the intent in exactly the same way, this proximity constraint is to allow variant occurrences within a window of some fixed size. The smaller the window size is, the more restrictive the matching.

The advantage of this stemmer is it improves selective word expansion on the query side and conservative word occurrence matching on the document side.

The disadvantage is the processing time and the complex nature of the stemmer. There can be errors in finding the noun phrases in the query and the proximity words.

## 6. Comparison between the Algorithms

As per all the methods and the related stemming algorithms discussed so far, a comparative of them related to their advantages and limitations is shown in Table 4, Table 5 and Table 6. It is clearly deduced that none of the stemmers are totally exhaustive but more or less the purpose of stemming is resolved. As of now the Porter's Stemmer is the most popular and researchers make their own changes in the basic algorithm to cater to their requirements.

**Table 1. Truncating (Affix Removal) Methods**

| **Lovins Stemmer** | |
|---|---|
| Advantages | Limitations |
| 1) Fast – single pass algorithm.<br>2) Handles removal of double letters in words like 'getting' being transformed to 'get'.<br>3) Handles many irregular plurals like – mouse and mice etc. | 1) Time consuming.<br>2) Not all suffixes available.<br>3) Not very reliable and frequently fails to form words from the stems .<br>4) Dependent on the technical vocabulary being used by the author. |
| **Porters Stemmer** | |
| Advantages | Limitations |
| 1) Produces the best output as compared to other stemmers.<br>2) Less error rate.<br>3) Compared to Lovins it's a light stemmer.<br>4) The Snowball stemmer framework designed by Porter is language independent approach to stemming. | 1) The stems produced are not always real words.<br>2) It has at least five steps and sixty rules and hence is time consuming. |

| **Paice / Husk Stemmer** | |
|---|---|
| Advantages | Limitations |
| 1) Simple form.<br>2) Each iteration takes care of deletion and replacement. | 1) Heavy algorithm.<br>2) Over stemming may occur. |
| **Dawson Stemmer** | |
| Advantages | Limitations |
| 1) Covers more suffixes than Lovins<br>2) Fast in execution | 1) Very complex<br>2) Lacks a standard implementation |

**Table 2. Statistical Methods**

| **N-Gram Stemmer** | |
|---|---|
| Advantages | Limitations |
| 1) Based on the concept of n-grams and string comparisons.<br>2) Language independent. | 1) Not time efficient.<br>2) Requires significant amount of space for creating and indexing the n-grams.<br>3) Not a very practical method. |
| **HMM Stemmer** | |
| Advantages | Limitations |
| 1) Based on the concept of Hidden Markov Model.<br>2) Unsupervised method and so is language independent. | 1) A complex method for implementation.<br>2) Over stemming may occur in this method. |
| **YASS Stemmer** | |
| Advantages | Limitations |
| 1) Based on hierarchical clustering approach and distance measures.<br>2) It is also a corpus based method.<br>3) Can be used for any language without knowing its morphology. | 1) Difficult to decide a threshold for creating clusters.<br>2) Requires significant computing power. |

**Table 3. Inflectional & Derivational Methods**

| **Krovetz Stemmer** | |
|---|---|
| Advantages | Limitations |
| 1) It is a light stemmer.<br>2) Can be used as a pre-stemmer for other stemmers. | 1) For large documents, this stemmer is not efficient.<br>2) Inability to cope with words outside the lexicon.<br>3) Does not consistently produce a good recall and precision.<br>4) Lexicon to be created in advance. |
| **Xerox Stemmer** | |
| Advantages | Limitations |
| 1) Works well for a large document also.<br>2) Removes the prefixes where ever applicable.<br>3) All stems are valid words. | 1) Inability to cope with words outside the lexicon.<br>2) Not implemented successfully on language other than English. Over stemming may occur in this method.<br>3) Dependence on the lexicon makes it language dependent. |

## 7. Conclusion

As can be seen from all the algorithms that have been discussed so far, there is a lot of similarity between the stemming algorithms and if one algorithm scores better in one area, the other does better in some other area. In fact, none of them give 100% output but are good enough to be applied to the text mining, NLP or IR applications.

The main difference lies in using either a rule-based approach or a linguistic one. A rule based approach may not always give correct output and the stems generated may not always be correct words. As far as the linguistic approach is concerned, since these methods are based on a lexicon, words outside the lexicon are not stemmed properly. It is of utmost importance that the lexicon being used is totally exhaustive which is a matter of language study. A statistical stemmer may be language independent but does not always give a reliable and correct stem.

The problem of over stemming and under stemming can be reduced only if the syntax as well as the semantics of the words and their POS is taken into

consideration. This in conjunction with a dictionary look-up can help in reducing the errors and converting stems to words. However no perfect stemmer has been designed so far to match all the requirements.

## 8. Future Enhancements

Although a lot of research work has already been done in developing stemmers there still remains a lot to be done to improve recall as well as precision.

There is a need for a method and a system for efficient stemming that reduces the heavy tradeoff between false positives and false negatives. A stemmer that uses the syntactical as well as the semantical knowledge to reduce stemming errors should be developed. Perhaps developing good lemmatizer could help in achieving the goal.

## 9. References

[1] Eiman Tamah Al-Shammari, "Towards An Error-Free Stemming", in Proceedings of ADIS European Conference Data Mining 2008, pp. 160-163.

[2] Frakes W.B. "Term conflation for information retrieval". Proceedings of the 7th annual international ACM SIGIR conference on Research and development in information retrieval. 1984, 383-389.

[3] Frakes William B. "Strength and similarity of affix removal stemming algorithms". ACM SIGIR Forum, Volume 37, No. 1. 2003, 26-30.

[4] Funchun Peng, Nawaaz Ahmed, Xin Li and Yumao Lu. "Context sensitive stemming for web search". Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval. 2007, 639-646.

[5] Galvez Carmen and Moya-Aneg•n F˜lix. "An evaluation of conflation accuracy using finite-state transducers". Journal of Documentation 62(3). 2006, 328-349

[6] J. B. Lovins, *"Development of a stemming algorithm,"* Mechanical Translation and Computer Linguistic., vol.11, no.1/2, pp. 22-31, 1968.

[7] Harman Donna. "How effective is suffixing?" Journal of the American Society for Information Science. 1991; 42, 7-15 7.

[8] Hull David A. and Grefenstette Gregory. "A detailed analysis of English stemming algorithms". Rank Xerox ResearchCenter Technical Report. 1996.

[9] Kraaij Wessel and Pohlmann Renee. "Viewing stemming as recall enhancement". Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval. 1996, 40-48.

[10] Krovetz Robert. "Viewing morphology as an inference process". Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval. 1993, 191-202.

[11] Mayfield James and McNamee Paul. "Single N-gram stemming". Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval. 2003, 415-416.

[12] Melucci Massimo and Orio Nicola. "A novel method for stemmer generation based on hidden Markov models". Proceedings of the twelfth international conference on Information and knowledge management. 2003, 131-138.

[13] Mladenic Dunja. "Automatic word lemmatization". Proceedings B of the 5th International Multi-Conference Information Society IS. 2002, 153-159.

[14] Paice Chris D. "Another stemmer". ACM SIGIR Forum, Volume 24, No. 3. 1990, 56-61.

[15] Paice Chris D. "An evaluation method for stemming algorithms". Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval. 1994, 42-50.

[16] Plisson Joel, Lavrac Nada and Mladenic Dunja. "A rule based approach to word lemmatization". Proceedings C of the 7th International Multi-Conference Information Society IS. 2004.

[17] Porter M.F. "An algorithm for suffix stripping". Program. 1980; 14, 130-137.

[18] Porter M.F. "Snowball: A language for stemming algorithms". 2001.

[19] R. Sun, C.-H. Ong, and T.-S. Chua. "Mining Dependency Relations for Query Expansion in Passage Retrieval". In *SIGIR*, 2006

[20] Prasenjit Majumder, Mandar Mitra, Swapan K. Parui, Gobinda Kole, Pabitra Mitra and Kalyankumar Datta. "YASS: Yet another suffix stripper". ACM Transactions on Information Systems. Volume 25, Issue 4. 2007, Article No. 18.

[21] Toman Michal, Tesar Roman and Jezek Karel. "Influence of word normalization on text classification". The 1st International Conference on Multidisciplinary Information Sciences & Technologies. 2006, 354-358.

[22] Xu Jinxi and Croft Bruce W. "Corpus-based stemming using co-occurrence of word variants". ACM Transactions on Information Systems. Volume 16, Issue 1. 1998, 61-81.

[23] Hull D. A. and Grefenstette,. " A detailed analysis of English Stemming Algorithms", XEROX Technical Report, http://www.xrce.xerox.

AUTHORS PROFILE

Ms. Anjali Ganesh Jivani is an Associate Professor in the Department of Computer Science and Engineering, The Maharaja Sayajirao University of Baroda. She is pursuing her doctorate in Computer Science and her research area is Text Mining. She has published a number of National and International level research papers related to Text Mining. She has also co-authored a book titled 'SQL & PL/SQL Practice Book', ISBN 978-81-910796-0-9.