Chapter 2

Literature Survey

2.1 Introduction

In this chapter, detailed and exhaustive study of available affix removal stemmers, statistical stemming approaches, lemmatizers and morphological analyzers are discussed. This literature study forms the base for the research that was conducted.

For the preprocessing of any Text Mining (TM) and Natural Language processing (NLP) related research work or application, the steps are: tokenization, stop-words removal and stemming/lemmatization.

Tokenization is done to separate out each word in the given text into words, punctuation marks and white space. The white space and punctuation marks are removed. Each word is called as token which can be further processed. Stop-words removal is related to removing the words that occur more commonly like articles, prepositions, and conjunctions etc. which by themselves carry very less information. This would focus on giving more importance to the words that remain in the input text now.

The next step deals with reducing the words that remains from the previous step to their word stem or their root word. In stemming the word stem is obtained by removing the affixes e.g. aborts, aborted, aborting become abort and sometimes words like abolish, abolished become abol which is not a dictionary word. In lemmatization unlike stemming, meaningful root words are generated from the inflected and derived words.

2.2 Approaches to Stemming

W.B Frake, renowned IR researcher: Software Engineering Guild, Sterling, categorized the stemming process into four major techniques as follows (W. B. Frakes: Stemming Algorithm, 1984, 1986):

1. Affix-based or Rule-based
2. Statistical or Successor Variety
3. Table lookup
4. N-gram Statistical Approach.

These four categories were further extended to a mixed approach which were: Corpus-Based, Context-Sensitive, K-stemming and Xerox stemming. Some other statistical techniques like Yet Another Suffix Stripper (YASS) and HMM based stemmers etc. were also incorporated into language-independent stemming methods in later stage. Fig. 2.1 depicts the different categories of stemming algorithms.

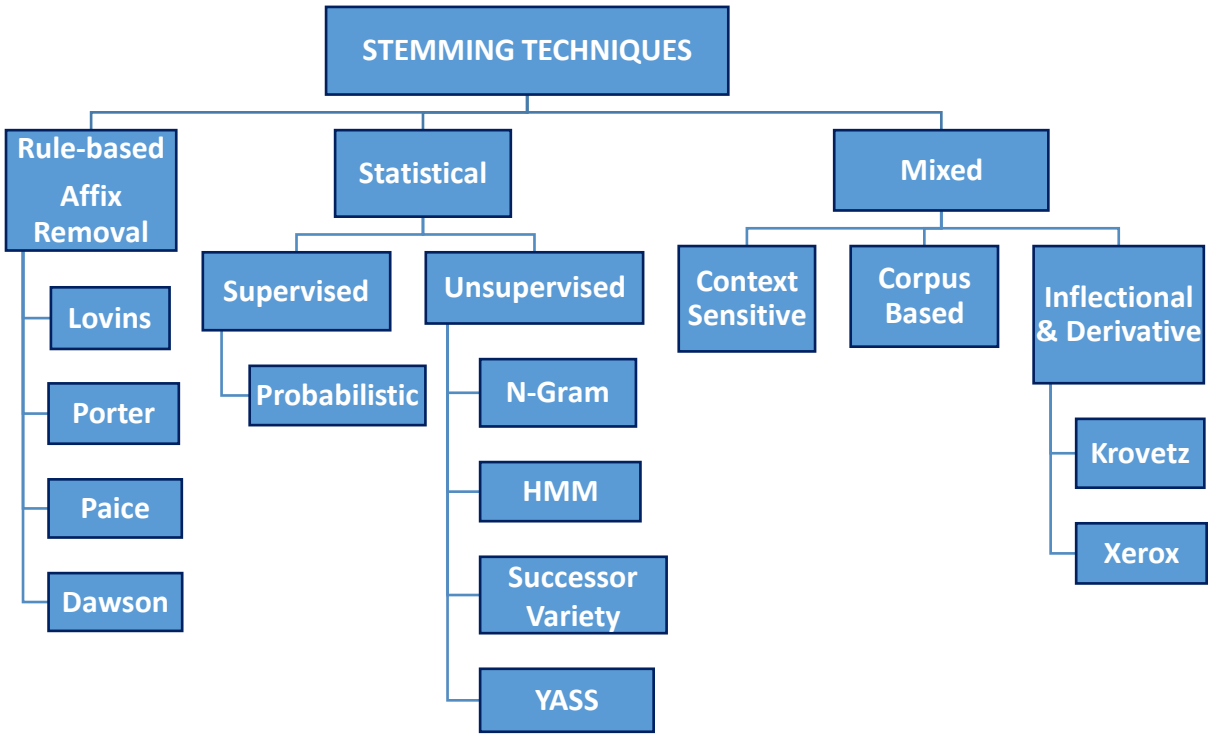

Figure 2.1 Stemming Techniques

Rule-based or truncating stemmers apply a set of English morphological pre-defined conditional and transformation rules on textual words to generate a single token from different inflectional and derivational variants after chopping off suffixes and prefixes without exploiting linguistic basis (e.g. schools to school/woman to wom /believe to belief/central, center to cent).

Julie Beth Lovins first developed (1968) a single pass, context sensitive stemmer which removes endings based on the longest-match principle. This stemmer was the first published

popular stemmer[1]. Lovins work was based on that of three earlier attempts by - Professor John W.Tukey from Princeton University, Michael Lesk from Harvard University who worked under the direction of Professor Gerard Salton and James L. Dolby of R and D Consultants from Los Altos, California[2].

Dawson Stemmer (1974) is an extension of the Lovins approach except that it covers a much more comprehensive list of about 1200 suffixes. Like Lovins, it 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 the Dawson algorithm. (Dawson, J. 1974, Anjali G. Jivani, 2011)

In 1980, Martin Porter developed a stemmer which is also widely used in the area of TM and NLP. (Porter M.F., 1980, 2001). This one is currently also very popular stemmer.

In late 1980, an iterative Paice-Husk stemming algorithm was designed and developed by Chris D Paice at Lancaster University[3]. This algorithm incorporated a compact set of stemming rules, specifying the removal or replacement of an ending of a word. Paice also developed a direct measurement for comparing stemmers based on counting the over-stemming and under-stemming errors. (Paice Chris D, 1990, 1994)

The Krovetz stemmer (KStem) was developed in 1993 by Robert Krovetz and is a linguistic lexical validation stemmer. It effectively and accurately removes inflectional suffixes.

2.3 Analysis of Affix Removal Stemming Algorithms

Lovins' Stemmer:

Lovins' stemming algorithm is developed based on two major concepts: iteration and longest-match. Iteration is handling suffixes of any morphed English word to generate a dictionary root word or stem token**.** It uses 294 endings, 29 conditions and 35 transformation rules to generate stem token form different morphed word variants. Lovins' algorithm handles inflectional suffices as well as irregular singular-plural suffices. Below mentioned steps broadly describes Lovins' stemming algorithm. (J. B. Lovins, 1968)

1. "Longest–match" technique:

[1] https://en.wikipedia.org/wiki/Julie_Beth_Lovins

[2] https://en.wikipedia.org/wiki/Stemming

[3] https://en.wikipedia.org/wiki/Christopher_D._Paice

This algorithm determines the position from where an ending list begins searching for a match, so that the stem is at least two characters long and then searches the ending list for a match to the last part of the word being stemmed (e.g., metallic to metall, metal to met, metallically to metall, magnetizable to magnet).

1.1 If the longest ending is found with its associated given condition and, then it is removed to generate stem.

2. Suffix Removal:

If an ending suffix is found from the given 294 suffices ending list (Table 2.1),

2.1 Suffix is checked as to whether the pre-defined context sensitive rule is satisfied or not (e.g., induction to induct).

If that word's contextual suffix is satisfying the condition,

then the suffix will be removed to generate the 1st phase stem.

e.g., for the word "believed", context sensitive rule "ed E" is found and then condition rule "E" is applied to generate "believ".

3. Recoding of stem:

3.1 The 1st phase stem will be re-coded for generating the final one.

3.2 The final consonant will be un-doubled, if exists within the stem.

3.3 Stem is recoded based on a matched transformation rule retrieved from 35 word endings transformation rules (Table 2.1) to generate final stem.

(e.g., induct to induc, metal to metal, magnet to magnet).

3.4 If no transformation rule is matched,

Previously generated stem will be declared as the final stem.

According to Lovins' algorithm, below mentioned word variations in English occur in Latinate derivations. Stem is separated from the ending by a vertical bar.

produc|er : product|ion , invert|ed : invers|ion , induc|ed : induct|ion , adher|e : adhes|ion induct|ed : induct|ion, register|ing : registr|ation, consum|ed : consumpt|ion, resolv|ed :resolut|ion ; absorb|ing : absorpt|ion admitt|ed : admiss|ion ; attend|ing : attent|ion circl|e : circul|ar ; expand|ing : expans|ion matrix| : matric|es ; respond| : respons|ive lattic|e : lattic|es ; exclud|e : exclus|ion index| : indic|es ; collid|ing : collis|ion hypothes|ized : hypothet|ical ; analys|is : analyt|ic

Table 2.1 shows sample list of suffixes and their transformation rules.

Table 2.1 Sample List of Suffices and Recoding Rules of Lovins

Appendix A. The list of endings	Codes for context-sensitive rules associated with certain endings	Appendix C. Transformation rules used in recoding stem terminations
alistically B, arizability aizationally B, izationally B	A: No restrictions on stem B: Minimum stem length = 3	iev →ief rpt→ rb ix→ ic
ability A, aically A, alistic balities A, es E, ionality A, ionalize A, iousness A, izations A	C:Minimum stem length = 4 D:Minimum stem length = 5 E: Don't remove ending after e.	ax → ac ex → ec ix →ic urs→ur

Table 2.2 shows the sample output which is generated from Lovins stemming algorithm.

Table 2.2 Sample Set of Output of Lovins Stemmer

Word	Stemmed Term				Word	Stemmed Term			
	Porter	Lovins	Paice	KStem		Porter	Lovins	Paice	KStem
believe	believ	belief	believ	believe	index	index	indic	index	index
belief	belief	belief	believ	belief	indices	indic	indic	Ind	indices
running	run	run	run	running	formulae	formula	forml	formula	formulae
run	run	run	run	run	formula	formula	forml	formul	formula

Note: In LemmaChase and LemmaQuest model, concept of Lovins' suffix removal and transformation rules are implemented to generate root word from input word.

Porter's Stemmer:

This stemmer checks the presence of suffices within a word and then matches that suffix with already stored suffices. The algorithm's six steps are listed below:

1. This step identifies suffices and recodes them accordingly.
   - 1.1 It identifies and removes suffix ("-sses", "-ies", "-s") from words having plural part-of-speech and recodes to generate stem token (e.g. actresses to actress, abilities to ability, ponies to pony & pennies to penni).
   - 1.2 It also removes ending "–ed" used in the past tense and ending "-ing" used in the present continuous tense from words and recoded them. (e.g., abbreviated to abbreviate & abbreviate to abbreviate) and then, applying recode rule to convert "abbreviat" into "abbreviate".
2. The second step recodes terminal 'y' into 'i' in presence of another vowel within a stem or original word. e.g.," penny" to "penni", "ability" to "ability".

3. Third step eliminates double suffices and also does recoding based on a given list of suffices. E.g., from "running", -"ing" suffix is removed and then "runn" is recoded into "run".
   3.1 It indexes the 2nd last character of the stem token. It maps double suffices to a single suffix.
   3.2 This step also recodes "-ational", "-tional" into "-ate", "-tion" respectively. e.g., "application" into "applicate". e.g., "applicate" to "applic" using recoding rule "- icate>"-ic").
4. It eliminates "-ance", "-er" endings from stem words using <c>vcvc<v>. e.g. "Acceptance" and "acceptable" into "accept".
5. Final step removes "-e" ending from the input stem which is received from the 5th step. e.g., "assemble" to "assembl".

In this algorithm, a word "generalizations" is stemmed into a word "generalization" in step-1 and, then into a word "generalize" in step-2. Step-3 converts it into a word "general" and, then the final stem "gener" will be developed by step-5.

Table 2.3 shows the sample output which is generated from Porter stemming algorithm.

Table 2.3 Sample Set of Output of Porter Stemmer

Word	Stemmed terms				Word	Stemmed terms		
	Porter	Lovins	Paice			Porter	Lovins	Paice
reporters	report	reporter	Report		Applies	Appli	appl	apply
reporting	Report	report	Report		Apply	Appli	Ap	apply
acceptance	Accept	accept	acceiv		Application	Applic	applic	apply
acceptable	Accept	accept	acceiv		Abilities	Abil	Abil	abl
applicant	Applic	appl	appl		Ability	Abil	Abil	abl

Paice Stemmer:

Paice algorithm is working based on a rule table. This stemmer has quite similarity with Lovin's approach. Each rule directs either for deletion or for replacement of endings from a word to generate a stem token. The rules are clustered into sections corresponding to the final letter of the suffix. The rule table is searched quickly by looking up the final letter of the current word or truncated word. Within each section, the ordering of rules plays a meaningful role. Each rule comprises five elements, two of which are optional:

a) " An ending of one or more characters, held in reverse order;"

b) ” An optional intact flag "*";”

c) ” A digit specifying the remove total (may be zero);”

d) ” An optional append string of one or more characters;”

e) ” A continuation symbol, ">" or "."”

Paice stemming algorithm is given below:

1. “Selection of relevant section: First, the final letter of the form is inspected (For e.g. “center” has final letter ‘r’) ; if there is no section corresponding to that letter, then process will be terminated; otherwise, first rule (“re2 {-er>-}”) in the relevant section is applied (for “center”/ “hsiug5ct” {-guish>-ct}” for “distinguish”).”
2. “Check applicability of rule: If the final letters of the form are not matched with the reversed ending string in the rule, then process will jump into step 4; if the ending is matched, and the intact flag is set, and the form is not intact, then process will jump into 4; if the acceptability conditions are not satisfied, then process will move to 4.”
3. “Apply rule: The number of characters specified in the “remove total” (mentioned above in rule elements) are deleted from right hand of the form; if there is an “append string”, then it is appended to the form; (e.g. “-er” is removed from “center”/ “-guish” is removed from “distinguish”). If the continuation symbol "." exists into rule element, then the process will be terminated.”
4. ” Look for another rule: Process will move to the next rule in the table; if the section letter has been changed, then process will be terminated; otherwise process will move to 2 (ai*2. {-ia > - if intact}; a*1.{ -a > - if intact }).”

Table 2.4 shows the sample output which is generated from Paice stemming algorithm.

Table 2.4 Sample Set of Output of Paice Stemmer

Word	Stemmed terms				Word	Stemmed terms			
	Porter	Lovins	Paice	KStem		Porter	Lovins	Paice	KStem
center	center	center	cent	center	ox	ox	ox	ox	ox
central	central	centr	cent	central	oxen	oxen	oxen	ox	oxen
woman	woman	woman	wom	woman	distinguishing	distinguish	distinguish	distinct	distinguish
women	women	wom	wom	women	distinguish	distinguish	distingue	distinct	distinguish

Krovetz-Stemmer Stemming (KStem):

The primary task of KStem is briefly described in above mentioned section. Now, the algorithm is elaborated in the below-mentioned steps:

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

This algorithm uses a machine-readable dictionary. It uses inflectional affix removal and transformation rule as well as already stored dictionary. The broad steps of this **KStem** algorithm are described below.

1. A list of derivational endings from Longman dictionary and different term dictionaries are initially created containing some root words and derivational morphed words. Exceptional Words: {"bathe", "caste", "cute"}, direct Conflations :{"aging", "age"}, {"dying", "die“}}. Different dictionary words (a to z) are stored within 8 different files.
2. The word is either plural or not, is checked with the help of a plural inflectional rule function after passing through an exceptional word list.
3. Then, trimmed stem or original word is searched in a dictionary, if it exists, its root word is returned, otherwise the word will be passed through past Tense inflectional rule handler function.
4. The word in “verb” or “noun” part-of-speech form, ended with “-ing”, is stemmed in aspect handler function. But, function handles short words (aging → age) via a direct mapping. This prevents (thing → the) “thing” from being retrieved without stemming via dictionary. If the input document size is large, this stemmer will become weak at performance. This stemmer does not consistently produce a good recall and precision performance.

Table 2.5 shows the sample output which is generated from KStem stemming algorithm.

Table 2.5 Sample Output of KStem Stemmer

Word	**Stemmed terms**					**Word**	**Stemmed terms**			
	Porter	**Lovins**	**Paice**	**KStem**			**Porter**	**Lovins**	**Paice**	**KStem**
authority	author	author	auth	authority		state	state	st	stat	state
author	author	author	auth	author		statement	statement	stat	stat	statement
authorize	author	author	auth	authorize		station	station	stat	stat	station
factory	factori	factor	fact	factory		news	new	new	new	news
factor	factor	fact	fact	factor		new	new	new	new	new
factorial	factori	factor	fact	factorial		age	ag	ag	age	age
factorize	factor	factor	fact	factorize		aging	ag	aging	ag	age

Statistical Morphological Analyzer / Stemmer:

The statistical stemming is developed based on a novel string distance measure which leads to lexicon clustering. The results show that stemming significantly improves retrieval performance (as expected) by about 9-10%, and the performance of the statistical stemmer is comparable with that of the rule-based stemmer. In the absence of extensive linguistic resources for certain languages, statistical language processing tools have been successfully used to improve the performance of IR systems and of the IE system. Rule-based stemmers for resource-poor languages are either unavailable or lack comprehensive coverage, in all those cases, a statistical approach can successfully stem any word for that language. Two types of statistical approaches are applied to develop different stemmers, one approach is unsupervised and another one is supervised.

Unsupervised Stemming Approach:

In statistical stemming, unsupervised approach aims to identify morphological classes that share common roots based on different string similarity distance.

Prasenjit Majumder, Mandar Mitra, Swapan K. Parui, and Govinda Kole, ISI (2007) have proposed a stemming algorithm [YASS], independent of linguistic expertise. YASS algorithm is basically developed based on unsupervised statistical approach. This Stemming has a target to identify morphological classes that share common roots based on some statistical string similarity measures which is implemented in the proposed "LemmaQuest" algorithm. In this YASS stemming algorithm, average of summation of four String Distance Measures functions are used to cluster words into homogeneous groups. Each group is expected to represent an equivalence class consisting of morphological variants of a single root word.

The work of Adamson and Boreham (1974) was the first to report the use of string distance similarity measures in stemming–dice coefficient based on generation of bi-grams (Adamson and Boreham, 1974). Generation of N-gram token is statistical based stemmer. Though it is called "stemming method", but a stem token is not produced in this algorithm. This method has no requirement of grammatical knowledge of the respective natural language. It is applied for classification of text. String distance similarity measures have been used to estimate morphological relatedness of word pairs using shared consecutive sequences of characters or n-grams, as well as by applying directly on individual characters of words. String similarity

measures have also been used for other tasks such as determining words with the same root (Roeck and Al-Fares, 2000), query matching, and indexing for morphologically related languages.

The number of shared digram within a pair of character strings is used to compute Dice's Similarity Coefficient which helps to develop cluster from sets of character strings within text (Adamson and Boreham, 1974). A digram is a pair of successive letters. There is a little bit of perplexity in using the "stemming" term here because this method is not at all generating stem. Similarity-measure of two word forms is calculated based on the number of shared unique digram of a word with another word's unique digram which are generated in this method. Association measures between the pair of terms "phosphorus" & "phosphate" are calculated here.

Phosphorus=>ph, ho, os, sp, ph, ho, or, ru, us; Unique digrams = ph, ho, os, sp, or, ru, us ;

Phosphate=>ph, ho, os, sp, ph ha, at, te ; Unique digrams = ph, ho, os, sp, ha, at, te ;

Dice's coefficient (similarity) $S = \frac{2C}{A+B} = \frac{2*4}{7+7} = .57$

A=number of unique digrams in the 1st word B = number of unique digrams in the 2nd word.

C = number of unique digrams shared by A, B.

Table 2.6 shows similarity matrix for below mentioned morphed words based on Dice's similarity.

Table 2.6 Similarity Matrix Based on Dice's similarity

Words	statistics	statistic	statistical	statistician
statistics	0.00	1.08	0.93	0.87
statistic	0.00	0.00	1.00	0.93
statistical	0.00	0.00	0.00	0.81
statistician	0.00	0.00	0.00	0.00

The above mentioned similarity sparse matrix justifies the observation of Adamson and Boreham. Cluster is generated based on cut-off value 0.6 in the similarity matrix. So, this process helps to cluster the documents based on similarity measure in between adjacent words by creating virtual digram.

Haris' phonetic segmentation: Based on literature survey, it was studied that Zellig. S. Harris, University of Pennsylvania (1955) has first discussed about structural phonemes, utterance, and successor variety and word boundaries in the structural and linguistics domain. Z.

S. Harris' process focused on breaking phonetic text into its constituent morphemes. Z. S. Harris' process had broken phonetic text into its constituent morphemes.

Margaret. A. Hafer and Stephen. F. Weiss's segmentation process uses certain statistical properties of a corpus (successor and predecessor letter variety counts) to indicate where words should be divided (M. Hafer And Stephen F. Weiss, 1974). Consequently, this process is less reliant on human intervention than are other methods for automated stemming. The process is based on the notion of a letter successor variety.

Successor Variety Stemming: This stemmer is processed by decomposing word into segment and is developed based on structural morphology to identify the word and morpheme boundaries through phonemes' distribution in a large body of utterances. The word is segmented into "stems and affixes" by using certain statistical properties of a corpus (successor and predecessor variety counts) which decide the index from where the word should be parsed. Hafer's and Weiss's three approaches to generate stem from a word are discussed here.

I. Cut off method**:** -The simplest method to decompose a test word, is first to choose some cut off "k" and then decompose the word at a particular index where its successor (or predecessor or both) variety meets or over reaches k. This method was adopted by Haris in his phonetic analysis. In this method, careful decision to select 'k' is required.

"Test Word: readable

corpus: able, ape, beatable, fixable, read, readable, reading, reads, red, rope, ripe.

threshold=2 is considered and segmentation will be done when successor variety >= threshold" . in readable" word, "read" segment has 3 successor varieties (a, i, s) and next all segments (reada, readab, readabl, readable) have only one successor variety. so, from "readable, "read" segment is generated which will become a stem token.

II. complete word method: a word will be decomposed at a particular index before that, if that word-prefix or suffix is observed to be a whole word in the corpus. if a shortest substring "w'' of a word "w' " exists as a word in a document, is considered a stem of that particular word (w).

"read" is a broken part (segment) from "readable" word which is identified as a complete word in the corpus.

III. Entropy method: It exploits the benefits of the dissemination of successor variety characters.

The method's working process is as follows: Let $|D\alpha_i|$ be the number of words in a text body beginning with the 'i' length sequence of letters of a test word α.

Let $|D\alpha_{ij}|$ be the number of words in $|D\alpha_i|$ has the successor j.

Then probability that the successor letter of α_i is the j^{th} letter of the alphabet is given by $\frac{|D\alpha ij|}{|D\alpha i|}$

The entropy of successor system for a test word prefix α_i is $H\alpha_i = \sum_{\rho=1}^{26} -\frac{|D\alpha ij|}{|D\alpha i|} . \log_2 \frac{|D\alpha ij|}{|D\alpha i|}$

The importance of each successor letter is measured by entropy calculation within the segmentation process. Entropy is calculated by successor letter's probability of occurrence. Rare successor letters will have a smaller impact on taking decisions in segmentation with compare to highly probable successor variety.

Determination of stems: After segmentation of a word, which segment will be considered as a stem, will be determined. In most cases the first segment is chosen as the stem. When the segment appears in many different words, it is probably a prefix. Then, the second segment should be considered as a stem.

Stemming Process based on Supervised Approach

Supervised Stemming model describes the mutual reinforcement relationship between stems and derivations and then provides a probabilistic interpretation. Stemming algorithms based on statistical methods ensure no additional costs to add new languages to the system. This is an advantage that becomes crucial, especially for applications like digital libraries which are often constructed for a particular institution or nation, and are able to manage a great number of non-English documents as well as documents written in many different languages.

Benno Stein and Martin Potthast also introduced statistical stemming concept based on Marget A. Hafer & Stephen F. Weiss's Successor variety approach named "Successor Variety for Stemming" in 2007.

Massimo Melucci and Nicola Orioe developed a stemming approach based on statistical HMM concept by publishing paper "Algorithm for Probabilistic Stemming" in 2005.

Michela Bacchin, Nicola Ferro and Massimo Melucci (2005) developed a language-independent probabilistic stemmer. This model is applied into several languages. It describes the mutual reinforcement relationship between stems and derivations and then provides a probabilistic interpretation. This model splits each word of finite collection of words into all possible positions and then creates suffix list. The stemmer accepts word as an input and it tries

to determine the split which corresponds to a stem and a derivation. The stemmer uses the linguistic knowledge. It also considers only the pairs which lead to the word and not the whole collection. Table 2.7 shows the list of rule-based and statistical stemming techniques with their researchers name.

Table 2.7 Overview of popular Rule-based and Statistical Stemming Techniques

Stemmer Developer and Year	Stemming Model
Lovins, 1968	Suffix Stripping (rule-based)
Hafer and Weiss, 1974	Variety Successor (statistical)
Porter, 1980	Rule based suffix stripping
Harman, 1991	Light weight suffix stripping
Krovetz, 1993	Dictionary based lemmatization (rule-based)
Oard, Levow, and Cabezas, 2000	Character Co-Occurrence
Bacchin, Ferro, and Melucci, 2002	Graph-based (statistical)
Mayfield and McNamee, 2003	n-grams (statistical)
Bacchin, Ferro, and Melucci, 2005	Probability based stemming (statistical)
Majumder et al., 2007	String similarity distance (statistical)

Corpus-Based Stemming:

Jinxi Xu and W. Bruce Croft developed "Corpus-Based Stemming using Co-occurrence of Word" around year 1994/1995 under mixed category of stemming and they also followed statistical approach. They adapted to language characteristics of a domain as reflected in a corpus. e.g. "stocks" is plural form of "stock" in Wall Street Journal (WSJ), but primary meaning of "stocks" in corpus related with Medieval history are the devices to punish a prisoner for public humiliation. So, a good conflation for one corpus may be bad performer for another. Corpus-based statistics may guide algorithm to establish relationship between any two corpus words. Corpus-based stemmer helps to prevent to conflate "policy / police", "addition/ additive", university/universal" all these un-related words into single stem token. It is observed that such semantically un-related words rarely co-exist in a corpus. This concept is applied here through statistical approach. Here, affix removal stemmer's fails and wrongly conflate all above mentioned word pairs. Corpus-Based stemming process is as follows: Assumptions are: Word variants that should be conflated will occur in same documents or in some text window (100 words text window).

Following metric to measure the significance of "word form co-occurrence"

$em(a,b)= \max((n_{ab} - En(a, b))/(n_a + n_b)), 0)$ where $En(a, b) = kn_an_b$ is the expected number of co-occurrences assuming a and b are statistically independent where n_a, n_b are the number of occurrence of words a and b in the corpus. n_an_b is the number of times both a and b fall in given text window or a single document. K is the parameter which is estimated for corpus . This

metric is a variation of EMIM (Expected Mutual Information Measure) (Church and Hanks 1989; Van Rijsbergen 1979). It is used to measure significance of association. For word form co-occurrences, EMIM can be defined as EMIM (a,b) = $P(a,b)\log_{10}(P(a,b)/P(a)P(b))$ where $P(a,b)= n_{ab}/N$, [21]. $P(a)=n_a/N$, $P(b)=n_b/N$, N is the number of text windows "the" & "of" , "new" & "news" and "gas" & "gases". "Gas" indícate as "natural gas"/ HydroCarbon gas where as "gases" indícate "inert gas"/ "hot gas"/ "helium", "neon", "xenón" (Xu Jinxi and Croft Bruce W, 1998). "News" and "new" co-occur frequently, but they are un-related word pair. According to formula for estimating En(a,b), their expected co-occurrence is 50,000. but em score is 0, which suggests that they are not related. em scores of sample word pairs on WSJ with 100 words window.

Jiaul H. Paik, Swapan K. Parui (ISI, Kolkata) published a paper "A Novel Corpus-Based Stemming Algorithm using Co-occurrence Statistics" in 2011.

Context sensitive Stemming is a method where stemming is done before indexing a document. Context sensitive analysis is done using statistical modeling on the query side. This method was proposed by Fuchun Peng, Nawaaz Ahmed, Xin Li and Yumao Lu in 2007. This stemming process is divided into four steps after the query is triggered. Steps: Candidate generation, Query Segmentation and head word detection are implemented in this approach.

Most of the statistical stemmers (Table 2.8) are designed based on string similarity measures. For most of the statistical stemmers, the morphologically related words are grouped into different morphological classes. Morphological classes are created using a clustering algorithm, such as Hierarchical Agglomerative Clustering (HAC) with string distance as a distance metric for splitting clusters. Several classes of string distance measures have been proposed for different linguistic typology (Majumder et al., 2007). Below mentioned table shows the listing of string distance similarity measures for clustering morphologically related words which is beneficial to develop stemming algorithms or IE related tasks.

Table 2.8 mentions the name of researchers with their studies related to string distance measures.

Table 2.8 Research Studies related to String Distance Measures

Name of Researcher and Year	String Distance Measure
Adamson and Boreham, 1974	Dice based clustering (N-gram)
Jacquemin, 1997	Simple truncation based clustering
Gaussier, 1999	Simple truncation based clustering
Roeck and Al-Fares, 2000	Jaccard Better clusters
David Yarowsky and Wicentowski, 2000	Edit distance word alignment
Schone and Jurafsky, 2000	Edit Distance
Baroni, Matiasek, and Trost, 2002	Edit Distance

Hu et al., 2005	Edit distance
Majumder et al., 2007	D1, D2, D3, D4
Snajder and Basic, 2009	D3, D4, Dice and Levenshtein
Bhat, 2013	D2, D4 D2

Adamson and Boreham's Dice based clustering is developed by N-gram segmentation which is already described in above section.

Jacquemin, 1997 proposed an algorithm for automatically acquiring morphological links between words. The four steps of the algorithm are (1) single-word truncation, (2) conflation of multi-word terms, (3) classification and filtering, and (4) clustering of conflation classes.

Jacquemin proposed word k-similarity measure for connecting more than one allied morphed words.

Definition 1: Two words w and w' are said to be k-similar if and only if the following equation is true:

$p = \max(\min(|wl - k, |w'| - k), 1)$

$p = trunc(w, p) = trunc(w', p)$ where trunc(w, i) is composed of the first i characters of w and where |w| is the length of w. Using this definition, "immunization" and "immunized" are 3-similar: $p = \max(\min(12 - 3, 9 - 3), 1) = \max(6,1) = 6$ trunc(w, 6) = trunc(w', 6) = imrnuni .

Definition 2: Let k_0 be the minimal value of k such that two words w and w' are k-similar. The corresponding truncation is called Maximal Common String and the corresponding suffixes are called Minimal Truncation Suffixes.

For the preceding example, $k_o = 2$, the Maximal Common String is c=immuniz and minimal truncation suffixes are s= ation and s' = ed.

This algorithm extracts morphological relations from corpora and lists of terms. This algorithm relies on approximate string matching and does not need any prior linguistic knowledge. Author mentioned that most natural application of this algorithm is text simplification.

Eric Gaussier, 1999 proposed an unsupervised method to learn suffixes and suffixation operations from an inflectional lexicon of a language. The elements acquired with this method are used to build stemming procedures and can assist lexicographers in the development of new lexical resources. They adopted the following probabilistic model to account for such a derivation process:

$P(w1 \rightarrow w2) = \sum_{op} p(w2(G) = Op_G(w1(G)),$

$w_2(F) = Op_F(w1(F)), w2(M) = O_{pM}(w_1(M)),$

$w_2(SX)=O_{psx}(w_1(SX))$, $w_2(S)=O_{ps}(w_1(S))$ where

O_p is a derivation process, Op_G is the component of Op which operates on the graphemic layer, and w(G) is the graphemic layer associated to word w.

They adopted the following form for a suffixation operation S:

$$S = \begin{pmatrix} G_d = concat(G_o, s) \\ MS_o \rightarrow MS_d \end{pmatrix}$$

Where $G_d(MS_d)$ stands for the graphemic (morpho-syntactic) form of the derived word produced by the Suffixation operation, G_0 (MS_0) for the graphemic (morpho-syntactic) form of the original word on which the suffixation operation operates.

Author depicted an unsupervised method to acquire derivational rules from an inflectional lexicon. The interesting points of this method lie in its ability to automatically acquire suffixes, as well as to induce a linguistically motivated structure in a lexicon. This structure, together with the elements extracted, can easily be revised and corrected by a lexicographer. Morphologically Sensitive Clustering Algorithm was developed by Anne N. Deroeck and Waleed Al-Fares for Identifying Arabic Roots. The clusters are developed using Jaccard distance measure for Arabic words sharing the same root.

David Yarowsky and Richard Wicentowski have developed corpus based algorithm (Minimally Supervised Morphological Analysis by Multimodal Alignment) which is capable to handle inflectional morphological form and irregular Verb using weighted Levenshtein distance measure. This algorithm combines four original Alignment models based on relative corpus frequency, contextual frequency, contextual similarity, weighted string similarity and incrementally retrained inflectional transduction probabilities [Minimally Supervised Morphological Analysis by Multimodal Alignment]. This algorithm focused on highly irregular forms, acts as an unsupervised inflectional morphological Analyzer. E.g.(take → took/ skip → skipped/ defy → defies/ defy → defying).

Schone and Jurafsky, 2000 developed a knowledge-free algorithm “Knowledge-Free Induction of Inflectional Morphologies” to automatically induce the morphology structures of a language. The algorithm takes as input a large corpus and produces as output a set of conflation sets indicating the various inflected and derived forms for each word in the language. As an example, the conflation set of the word “abuse” would contain “abuse”, “abused”, “abuses”, “abusive”, “abusively”, and so forth. The algorithm automatically induces the morphology of

inflectional languages using only text corpora and no human input. The algorithm combines cues from orthography, semantics, and syntactic distributions to induce morphological relationships in German, Dutch, and English. Many NLP tasks, such as building machine-readable dictionaries, are dependent on the results of morphological analysis.

Baroni, Matiasek, and Trost, 2002, developed designed an algorithm "Unsupervised discovery of morphologically related words based on orthographic and semantic similarity" that takes an un-annotated corpus as its input, and returns a ranked list of probable morphologically related pairs as its output. The algorithm tries to discover morphologically related pairs by looking for pairs that are both orthographically (measured by minimum edit distance) and semantically similar (measured by mutual information), where orthographic similarity is measured in terms of minimum edit distance, and semantic similarity is measured in terms of mutual information.

The algorithm "A heuristic for morpheme discovery based on string edit distance" was developed by John Goldsmith, Yu Hu, Irina Matveeva, and Colin Sprague in 2005**.** It derives from work which explores unsupervised learning of the morphology of languages with rich morphologies, that is, with a high average number of morphemes per word. This algorithm works with Swahili, a major Bantu language of East Africa, and goal of the algorithm is the development of a system that can automatically produce a morphological analyzer of a text on the basis of a large corpus. Table 2.9 shows the list of popular stemming algorithms.

Table 2.9 Summarization of Existing Popular Stemming Algorithms

Stemmer	**Method**	**External Resource**	**Stem/ Lemma**	**Application**	**Advantage**	**Disadvantage**
Lovins, Paice, Porter	Affix Removal	Affix removal & transformation rule	Stem token	Information Retrieval/Extraction	Fast-single Pass Algo., Handle many irregular plurals.	Not very Reliable and fails to form words from stem in many times.
KStem	Dictionary LookUp	Affix removal & transformation rule, stored dictionary.	Lemma	Information Retrieval/Extraction	Generate dictionary word instead of invalid stem most of the times	Cannot consistently produce a good recall and precision
N-gram	Statistical	NIL	Di-gram	Clustering	Language Independent	Require significant amount of space for creating and indexing n-gram
Successor Variety	Statistical	Threshold value	Stem token	Language independent extraction/ retrieval	Language Independent	
Corpus-Based	Statistical	EM, window size	Word grouping	Clustering/ grouping	Appropriate for a given corpus, Valid word is generated after stemming	Need to develop individual statistical measure for every corpus separately.

Context-sensitive	Head word detection, Document matching	Document	Lemma	Web search	Improves selective word expansion on the query side.	High processing time, complex algorithm

2.4 Lemmatization

Unlike stemming, where a lot of work has been done and, many stemmers have been popularized, there is still scope in designing of a lemmatizer as discussed before. It has been observed from literature survey that broadly seven different categorical approaches have been used in designing of lemmatizers. The detail functionality of all above mentioned Lemmatization approaches are discussed in this chapter. Fig 2.2 shows broad categorization of Lemmatization techniques.

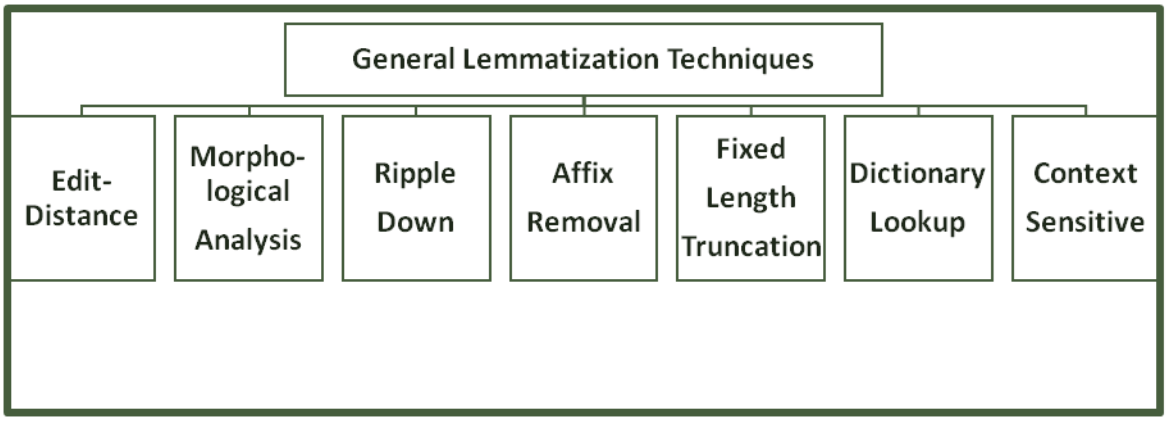

Figure 2.2 Lemmatization Techniques

2.5 Analysis of Lemmatizers

First Approach of Lemmatization: In lemmatization, the first approach, "Edit Distance" (also known as Levenshtein Distance) has been implemented on a dictionary-based algorithm in order to get the automatic induction of the normalized form (lemma) of regular and blandly irregular words without direct supervision. The algorithm is a composition of two alignment approaches based on the string similarity and the presence of the most frequent inflexional suffixes. In the "Edit Distance" algorithm, Levenshtein distances in-between strings are stored in a simple metric which can be used as an effective string approximation tool (Dimitrios P. Lyras, Kyriakos N. Sgarbas, Nikolaos D. Fakotakis, 2007).

The Edit Distance is calculated based on a simple dynamic programming algorithm which focuses on a primitive string edit operation. This operation is always possible to modify a source string A into a target string B (provided that both strings are subject to the same alphabet).

$d_L[i, j] = \min (d_L[i,j-1]+c[\varepsilon,B_j], d_L[i-1,j]+c[A_i,\varepsilon], d_L[i-1,j-1]+c[A_i,B_j])$ for $i \geq 1$ and $j \geq 1$,
with $d_L[0, 0] = 0$ and $d_L[i,-1] = d_L[-1, j] = \infty$ where: A_i is the i^{th} element of string A,
B_j is the j^{th} element of string B

Levenshtein Distance Calculation

The Levenshtein Distance of two strings A and B (denoted as d_L (A, B)) is computed on the minimum number of single character insertions, deletions and substitutions required to convert string A to string B. The d_L(A, B) can be easily calculated, which is mentioned above. However, greater the Levenshtein distance, more mismatched strings are found. The algorithm provides a choice to select the value of the approximation that the tool accepts as desired similarity distance (e.g. if the user accepts zero as the desired approximation, then only one target words with the minimum edit distance will be returned, whereas if any user enters e.g. 2 as the desired approximation, then a set of target words is returned having a distance <= (minimum + 2) from the source word. e.g., close → closing, stir → stirred.

Second Approach of Lemmatization:

The second approach is the Morphological Analyzer which is based on "finite state automata (FSA)". A Finite State Transducer is a more generic term than a finite-state automaton (FSA). A FSA provides a formal language by defining a set of accepted strings, while a FST provides relations in-between sets of strings. A FST will read a set of strings on the input tape**,** generate a set of relations on the output tape and translate the contents of its input tape to its output tape, to generate another string on its output tape. (Kimmo Koskenniemi, 1984, Lauri Karttunen and Kenneth R. Beesley, 2005)

The Concept of Morphological Analysis:

M.P. Schützenberger (1961) explained and implemented two sequential transducers (where the output of the first form is the input of the second) for generating a new word and for extracting morphemes (M.P. Schützenberger. 1961). “Morphology” concept had been first incorporated into the generative linguistics’ domain with Noam Chomsky, Morris Halle and Aronoff. Chomsky and Halle (1968, 1970, 1973) proposed ordered context-sensitive linguistic

rules for describing the phonological component as a system of rules which maps surface structures into phonetic representations. They formulated many phonological rules based on which, words are segmented into morphemes. The 7th rule of Chomsky described a derivational word's segmentation phonological rule which stresses a primary-stressed vowel in the context. e.g. $[CoVCo]_N/[erase]_V r]_N$. According to linguistic experts' observations, highly irregular relationships exist in-between English derived nominal and their supposedly corresponding verbs (profess → professor → profession → professional; social → socialist → socialite) and but for some nouns, no verb exists in the lexicon (tuition but no *tut, motion but no *mote).

C. Douglas Johnson (1972) discovered that ordered phonological rules can be implemented with a cascade of FSTs (Finite State Transducer) if the rules are never applied to their own output. Based on Harris's phonetic concept (1974), model segmented lexical text into stems and affixes with minimal human intervention which is described in statistical stemming approach section.

Kaplan & Kay (1980) rediscovered the findings of Johnson and Schützenberger. Ronald M. Kaplan and Martin Kay (1980) were putting rules into a more algebraic perspective than Johnson. According to Kaplan and Kay, formal languages were constructed with sets of strings and mathematical objects which are built from a finite alphabet E by the associative concatenation operation. Formal language theory has classified string sets and the subsets of E*. Ordered n tuples of strings: $X=<x_1,x_2\ldots x_n>$ & $Y=<y_1,y_2\ldots y_n>$ are shown here. Concatenation of X & Y: $X.Y=_{df} <x_1y_1,x_2y_2,\ldots x_ny_n>$ and this has the expected property that $|X.Y| = |X| + |Y|$, even if they have different length and $|X|=_{df} \sum_i | xi |$.

Kimmo Koskenniemi (1984) invented 2-level-morphology. They explained that morphology of a language is a set of rules which start from an underlying lexical representation, and transform it step by step until the surface representation is reached. Koskenniemi's two-level morphological process was the first practical general model in computational linguistics for the analysis of morphological complex languages.

In continuation of Kimmo's model, Lauri Karttunen and Kent Wittenburg (2003) have developed a "Two-Level Morphological Analyzer" (first FST compiler) based on Kaplan's implementation of the finite-state calculus. This analyzer can handle low level English morphological variants such as plural in nouns and present continuous, past and past participle in a verb and some comparative (-er), superlative (-est), singular & plural genitive ('s) and verb-to-

adjective marker (-able). But most of the derivative and nominalized morphed words are still unexplored.

GoldSmith (2001) developed Linguistica5 software for unsupervised learning of linguistic structure. Unsupervised learning revolves around morphological signature's objects in Linguistica. For example, {Ø, s} is a morphological signature set which is used in any sizable English datasets, with possible linked stems such as walk-, jump- (which entails that the words walk/walks, jump/jumps occur in the data).

Krister Lindén (2008, 2009) attempted to model the transformation between base and inflected form part by part, but adopted a simpler, three-way split into prefix, stem and suffix. Linde´n mentioned that the model was implemented as a cascade of Finite state.

Third Approach of Lemmatization:

This approach is called "Ripple-Down Rule". It is a data structure, used by popular Lemmagen , which allows retrieving a possible lemma of a given inflected or derivational form. Ripple-down rules are an incremental approach to knowledge acquisition. Ripple-down rules are an incremental approach to knowledge acquisition. (Matjaž Juršič, Igor Mozetič, Tomaž Erjavec, Nada Lavrač, 2010)

RDR ::= IF rule.condition THEN rule.class EXCEPT rule.exceptions

Where – rule.condition is a wordform suffix, – rule.class is a transformation to be applied to a wordform, and – rule.exceptions ::= nil | RDR+ (list of RDRs) .

Fourth Approach of Lemmatization:

The fourth approach is Affix lemmatizer which is a combination of a rule based and supervised training approach and the last approach is fixed-length truncation approach. German and Dutch need more advanced methods than suffix replacement since their affixing of words (inflection of words) can include both prefixing, infixing and suffixing. Therefore, a trainable lemmatizer is required to create that handles pre- and infixes in addition to suffixes.

Dutch full form lemma pair "Afgevraagd → afvragen" conversion was explained in this lemmatization process (Translation: wondered, to wonder). Input given to the training program, it should produce a transformation rule like this:*ge*a*d → ***en. (Bart Jongejan, Hercules Dalianis, 2009)

Fifth Approach of Lemmatization:

The fifth one is Fixed Length Truncation approach which has been successfully executed for the Turkish language. In this approach, word is simply truncated and the first 5 and 7 characters of each word are considered as its lemma. In this approach, words with less than n characters are used as a lemma with no truncation. This fixed prefix truncation technique gives good results in the Turkish language. can et.al investigated the fixed-length truncation for 3 through 7 characters; they found that 5 were the best result. According to other experiments carried out on approximately 24K Turkish words, the length of Turkish words are composed of an average of 7.07 letters. So this technique uses the first 5 and 7 characters. (Fazli Can, Seyit Kocberber, Erman Balcik, Cihan Kaynak and Onur M Vursavas, 2008)

Sixth Approach of Lemmatization:

The sixth one is Context Sensitive lemmatization technique**.** Abhisek Chakrabarty Onkar Arun Pandit and Utpal Garain, ISI (2017) developed "Context Sensitive lemmatization model using Two Successive Bidirectional Gated Recurrent Networks" which is based on supervised deep neural network architecture and language- independent context sensitive lemmatization technique. This model learns the transformation patterns between word-lemma pairs (sang → sing, achieving → achieve) and hence, can handle the unknown word forms too.

Toms Bergmanis and Sharon Goldwater (2018) developed Context Sensitive Neural Lemmatization with Lematus which was executed on inflected English word variants to determine their dictionary root word (e.g., swims, swimming, swam, swum) and also on data from 20 typologically varied languages. They presented Lematus, a simple sequence-to-sequence neural machine translation framework that uses character-level context.

Abhisek Chakrabarty, Akshay Chaturvedi and Utpal Garain, CVPR Unit, ISI (2019) developed language independent CNN-based Context Sensitive Lemmatization which was executed on two indic languages (Hindi, Marathi) and two European languages (French and Spanish). For the English language, experiment was only done for verb in different tenses.

Seventh Approach of Lemmatization:

The seventh one is a Dictionary-based Turkish Lemmatizer (DTL) (Richard Wicentowski, 2002) DTL is based on "Grand Turkish Dictionary" as a dictionary. For the sake of efficiency, DTL uses radix-trie, which is one of the most efficient variants of the trie data

structure. Another dictionary-based approach is WordNet Lemmatizer (Jorge Morato, Miguel Ángel Marzal, Juan Lloréns, and José Moreiro, 2004) which is also popularly used for text processing applications.

Limitation of existing Stemmers and Lemmatizers:

After studying of morphological analyzer, it is observed that the nominalized and derived words are being treated as separate independent words without connecting them to their actual root words or lemmas. Derived words and their root words also exist independently in English dictionary. A derivational affix is added for a word (Verb) to convert it to a noun form. e.g., the noun "legalization" has been produced from the verb "legal". Existing lemmatizers also treat the derived and nominalized words as separate words. This is a challenge to connect nominalized word with its base word (lemma). So, LemmaChase has tried to meet this challenge. Examples of Nouns formed from adjectives are "applicability" (from "applicable"), "carelessness" (from "careless"), "difficulty" (from "difficult") and "intensity" (from "intense"). Examples of Nouns formed from verbs are "failure" (from "fail"), "nominalization" (from "nominalize"), and "investigation" (from investigate"). The nominalized word shows only a single Part of Speech (POS) in any English dictionary, though its root word basically exists in different POS forms and the root word's morphological structure also differs from nominalized word (e.g. "judgmental" to "judge"/ "application" to "apply"/ "angrily to angry").

2.6 String Distance Similarity Measures

After studying research papers, it was observed that string similarity distance calculation is the key measure to develop clusters for morphed-allied words. This is going to be used further, when the designed and developed lemmatization model is discussed.

Different String Similarity distance measures are explored for identifying similar class based derived words and to create group for them. Table 2.10 shows different string measures. These measures are used to identify the strings which are syntactically similar in construction (inflected and derivative words)[4]. Based on the observation of the output generated from

[4] https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50

different similarity measure, suitable string similarity measure can be incorporated within proposed algorithm[5].

Table 2.10 Different String Similarity Measures

	String Similarity Measure	Overview	
1.	Longest common subsequence (LCS) distance	It finds the longest subsequence common to all sequences in a set of sequences (often just two sequences). It differs from the longest common substring problem: unlike substrings, subsequences are not required to occupy consecutive positions within the original sequences.	
2.	Longest common substring problem	The longest common substring problem is to find the longest string that is a substring of two or more strings. The problem may have multiple solutions. Applications include data de-duplication and plagiarism detection.	
3.	Levenshtein edit distance	In the string correction problem, one string is converted into another using a set of prescribed edit operations. Formula is already mentioned in above section.	
4.	Damerau–Levenshtein Distance	In this edit operation algorithm, the permissible operations are: substitution, insertion, deletion and transposition which are applied for string to string conversion. Formulation of this distance is mentioned below this table.	
5.	Hamming Distance	The Hamming distance is the number of positions at which the corresponding symbols are different. In other words, it is the number of substitutions required to transform one string into another. Hamming distance is computed for equal length strings; otherwise shorted string is padded with space. Formula is given below.	
6.	Jaccard Distance	Jaccard Distance is a measure of degree of dissimilarity of two sets. The lower the distance, the more similar the two strings. Jaccard Distance depends on another concept called "Jaccard Similarity Index" which is (the number in both sets) / (the number in either set) * 100	$J(X,Y) = \|X \cap Y\| / \|X \cup Y\|$ $D(X,Y) = 1 - J(X,Y)$
7.	Cosine similarity	Cosine similarity is a metric used to determine how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. In this context, the two vectors are the arrays containing the word counts of two documents. Formulation is described below this table.	
8.	N-gram	In general, n-gram means splitting a string in sequences with the length n. So if we have this string "abcde", then bigrams are: ab, bc, cd, and de while trigrams will be: abc, bcd, and cde. n-grams can be used with Jaccard Distance. n-grams segmentation are beneficial in the applications such as machine translation when you want to find out which phrase in one language usually comes as the translation of another phrase in the target language. Bi-gram formulation was already described in Adamson and Boreham's dice formula.	
9.	Jaro-winkle	Jaro–Winkler distance is a string metric measuring an edit distance between two sequences. It is a variant proposed in 1990 by William E. Winkler of the Jaro distance metric (1989, Matthew A. Jaro). The Jaro–Winkler distance uses a prefix scale p which gives more favorable ratings to strings that match from the beginning for a set prefix length.	**Jaro-Winkler Similarity:** $sim_w = sim_j + l_p(1 - sim_j)$, sim_j is the Jaro similarity for strings s_1 and s_2. l is the length of common prefix at the start of the string up to a maximum of 4 characters p is a constant scaling factor for how much the score is adjusted upwards for having common

[5] https://medium.com/@adriensieg/text-similarities-da019229c894

		Jaro Similarity: Sim_j= {0 if m=0 Sim_j= { 1/3 (m/$|s_1|$ + m/$|s_2|$ + (m-t)/ m) otherwise. Where: |si| is the length of the string ; m is the number of matching characters ; t is the number of transpositions.	prefixes. The Jaro–Winkler distance d_w is defined as d_w=1- sim_w.
10.	The Sørensen–Dice coefficient	The Sørensen–Dice coefficient is a statistic which calculates to gauge the similarity of two samples. It was independently developed by the botanists Thorvald Sørensen and Lee Raymond Dice, who published in 1948 and 1945 respectively.	$x = \frac{2\|X \cap Y\|}{\|X\| + \|Y\|}$ where |X| and |Y| are the cardinalities of the two sets (i.e. the number of elements in each set). The Sørensen index equals twice the number of elements common to both sets divided by the sum of the number of elements in each set.
11.	Ratcliff/Obershelp Pattern Recognition	Gestalt Pattern Matching, also Ratcliff/ Obershelp Pattern Recognition, is a string-matching algorithm for determining the similarity of two strings. It was developed in 1983 by John W. Ratcliff and John A. Obershelp and it was published in the Dr. Dobb's Journal in July 1988. The similarity of two strings S_1 and S_2 is determined by the formula, calculating twice the number of matching characters Km divided by the total number of characters of both strings.	The matching characters are defined as the longest common substring (LCS) plus recursively the number of matching characters in the non-matching regions on both sides of the LCS: $Dro = \frac{2\text{Km}}{\|S1\| + \|S2\|}$ where the similarity metric can take a value between zero and one: 0<= D_{ro} <= 1

Longest common subsequence (LCS) distance: LCS is formulated for two strings in below mentioned equations.

$LCS(Xi, Yi) = \{\Theta$ if i=0 or j=0

{LCS($X_{i-1,}$ Y_{j-1})^ x_i if i, j>0 and $x_i = y_i$

{max{LCS($X_{i-1,}$ Y_{j-1}), LCS($X_{i-1,}$ Y_{j-1})} if i, j>0 and $x_i = y_i$

To find the LCS of X_i and Y_j, compare x_i and y_j. If they are equal, then the sequence LCS(X_{i-1}, Y_{j-1}) is extended by that element, x_i. If they are not equal, then the longer of the two sequences, LCS(X_i, Y_{j-1}) and LCS(X_{i-1}, Y_j) is retained. (If they are the same length, but not identical, then both are retained).

Hamming Distance:

```
def hamming_distance(string1, string2):
dist_counter = 0
        for n in range(len(string1)):
                if string1[n] != string2[n]:        dist_counter += 1       return dist_counter;
```

Cosine Similarity: [6]

The cosine of two non-zero vectors can be derived by using the Euclidean dot product formula:

$a.b = ||a|| \, ||b|| \cos\theta$

Given two vectors of attributes, *A* and *B*, the cosine similarity, cos(θ), is represented using a dot product and magnitude as

$$\text{Similarity} = \cos\theta = \frac{a.b}{||a|| \, ||b||} = \sum_{i=i}^{n} aibi \; / \; (\sum a2i \; \sqrt{\sum b2i})$$

- If the angle is small (they share many tokens in common), the cosine is large.
- If the angle is large (and they have few tokens in common), the cosine is small.
- Cosine similarity is a metric used to determine how similar the documents are irrespective of their size.

Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. In this context, the two vectors I am talking about are arrays containing the word counts of two documents.

Sample Output of Word-pair Distance:

Based on the observation of the value of String-similarity distance in between derived morphed word-pair/ inflected word-pair, Levenshtein edit distance algorithm generates best value for below mentioned word-pairs. Table 2.11 shows the distance value for different set of word-pair [Output generated through python function call].

Table 2.11 Word Pairs with Their Distance Measures

Name of the Distance	**Word-Pair**						
	Applicant-Application	Applicants-Applicable	Algebra-Algeria	Eat-Ate	Police-Polish	Abscond-Absorb	Academic-Academy
Longest-common-subsequence (LCS) distance	4	6	2	2	4	5	3
Levenshtein edit distance	0.33	0.7	0.7	0.333	0.66	0.5	0.75
Damerau–Levenshtein Distance	4	3	2	2	2	3	2
Hamming Distance (miss-matched index)	7	7	5	0	4	3	6
Jaccard Distance	0.818	0.538	0.75	1.0	0.5		0.66
Cosine similarity	0.90	0.7	0.857	1.0		0.61	0.81
N-gram							
Jaro-winkle	0.94	0.88	0.94	0.0	0.66	0.822	0.92
The Sørensen–Dice coefficient	0.9	0.7	0.857	1.0	0.66	0.615	0.8
Ratcliff/Obershelp Pattern Recognition	0.8	0.7	0.857	0.66	0.66	0.615	0.8

[6] https://www.machinelearningplus.com/nlp/cosine-similarity/

2.7 Conclusion and Summary

In this chapter, the detailed literature study has been discussed related to stemming, lemmatizers and morphological analyzers. Most of them have been implemented to compare their output and to identify the limitations. Different string distance measure were also studied and implemented and their comparative chart has been shown.

The literature study of affix-removal stemmers has been published as a paper titled "Empirical Analysis of Affix Removal Stemmers" in the journal "Int.J. Computer Technology & Applications, Vol 5(2), 393-399, ISSN: 2229-6093".