**ABSTRACT**

**Objective:**

To implement shingling algorithm of Document Similarity Detection for Indic Languages (Malayalam in particular). Due to the property of inflection and agglutination, Indic languages cannot be treated directly with word to word comparisons. So, a fuzzy search, indic soundex and edit distance technologies should be used. Edit distance technology is used for words which are written similarly, indic soundex technology is used for words which sound similar.

**Existing System :**

No system is in existence for Document Similarity for Malayalam Language. SILPA framework handles inflections partially, and we intend to improvise it.

**Proposed System :**

The texts are tokenized and each pair of corresponding token is treated with an approximate comparison algorithm which will take care of inflections (if any). Agglutionations are not in the scope of this project. The results of approximate comparison is used in Shingling Algorithm and the Jaccard's Similarity Formula to calculate the similarity. The words are arranged in ascending order and then tokenized to minimalize the problem of "word position similarity".

**Stakeholders :** Language Scientists, Media personnels, Literature Experts and students.

**Coding Backend :** Python and SILPA (Swathanthra Indic Language Processing Applications)

**GUI Frontend :** PyGTK/PyQt

**RELEVANCE OF PROJECT**

Holocene epoch has detected the enormous emergence of Internet document in the World Wide Web. Internet is now a vital factor for day today life in gathering information and extracting useful information from web pages thus becomes an important task. The performance and reliability of web search engines face huge problems due to the presence of extraordinarily large amount of web data. The voluminous amount of web documents has resulted in problems for search engines leading to the fact that the search results are of less relevance to the user. In addition to this, the presence of duplicate and near duplicate web documents has created an additional overhead for the search engines critically affecting their performance. The world is becoming a single global e-village and everything is going to the web. This, in turn, make sure that all the new developments and creations happen in the web. This induces a need for a system to check data theft and fraud. For both these scenarios – **reducing search engines' burden and checking for data theft** – similarity checking systems are required. Since, duplicate detection systems cannot work on Indic languages due to their grammatical and linguistic features like inflection and agglutination, near duplicate detection is only possible, which implements approximate search. Currently, there exists no full fledged document detection system for Indic languages. Hence, this project will be a starting step towards it. Moreover, the different features used in this project like ***stemming*** are important requirements in Natural Language Processing, where they are used for other applications. This near duplicate document detection system can be used to compare the ideas generated by two news paper articles, two reports or even two books and calculate their similarity. Since Natural Language Processing is a growing field in Computer Science, the different tools used in this project are useful to Language Experts, Language Scientists and other technologists.

The approximate search algorithm and fuzzy string calculation algorithm used in this project is useful for other projects like Search Engine implementation, context based searching etc.

**LITERATURE SURVEY**

The only advancements in the field of document comparison is the development of stemmers. In that also not much work has been reported for stemming for Indian languages compared to English and other European languages.

**2.1 Lightweight Stemmer by Ramanathan and Rao**

1. Hand crafted suffix list
2. Light stemming - stripping of a small set of either prefixes or suffixes or both, without trying to deal with infixes, or recognize patterns and find roots.
3. For Hindi – based on Hindi grammar – List of total 65 suffixes is generated manually.
4. Terms are conflated by stripping off word endings from a suffix list on a “longest match” basis
5. Advantage – computationally inexpensive

**2.2 Lightweight stemmer for Bengali**

1. Based on lightweight stemmer by Ramanathan and Rao
2. Strips suffixes using predetermined suffix list
3. Used in spelling checker
4. Does not derivational suffixes

**2.3 YASS – Yet Another Suffix Stripper**

1. Statistical approach
2. Clustering based approach
3. Based on string distance measured
4. Morphological stemming – classified into classes consisting of morphological variants of the root word.
5. Using grap-theoretic clustering algorithm – requires threshold θ

**2.4 Unsupervised morphological parsing for Bengali**

1. Segmenting words into prefixes, suffixes and stems
2. No prior knowledge of language specific morphotactics and morpho-phono logical rules.
3. Composed of two steps - (1) inducing prefixes, suffixes and roots from a vocabulary consisting of words taken from a large, unannotated corpus, and (2) segmenting a word based on these induced morphemes.
4. Better than Linguistica, widely used morphological parser.

**2.5 Unsupervised stemming algorithm for Hindi**

1. Based on split-all method
2. For training, words from EMILLE corpus
3. Words split to give n-gram suffix until the length of the word is reached.
4. Calculating stem and suffix probabilities and multiplied to get split probability
5. Optimal segment → maximum split probability

**2.6 Unsupervised stemmer for Marathi**

1. Three methods for suffix rules generation are used – Rule based, suffix stripping and statistical stripping.
2. Rule based stemmer uses a set of manually extracted suffix stripping rules whereas the unsupervised approach learns suffixes automatically from a set of words extracted from raw Marathi text
3. Uses set of words to learn suffixes
4. About 83% efficiency

**2.7 Lightweight inflectional stemmer and Heavyweight derivational stemmer for Gujarati**

1. Light weight stemmer uses parts-of-speech based stemming
2. Inflectional stemmer has accuracy of 90.7%
3. Boost due to POS based stemming – 9.6%
4. Boost due to inclusion of the language characteristics – 12.7
5. Derivational stemmer has average accuracy of 70.7%
6. Useful in dictionary search and data compression
7. Expansion using Named Entity Recognizer integration

**EXISTING SYSTEM AND FEATURES**

There are currently no existing systems for Malayalam Language. However, there exists some similar works which are intended for other Indic languages like Hindi, Marathi and Gujarathi. In Malayalam the only advancement is done by an organization name Swathanthra Malayalam Computing, through their Indic language framework called SILPA (Swathanthra Indic Language Processing Applications).

The features of stemmer and duplicate detection are as follows

1. Stemmer generated using predefined rule corpus which is in a partial state.
2. Suffix stripping is implemented and strip distance is used to compare words with rules dictionary.
3. Tokenize document to set of n-grams

The document comparison module of SILPA tokenized the documents and compare these tokens with their counterparts in the other document.

The disadvantage of SILPA project is that, it does not incorporate stemmer in document comparison. That is, inflections are not handled in the SILPA project. It performs only character to character comparison which is not at all useful for Indic languages.

**PROPOSED SYSTEM**

The proposed document comparison system is a modification to the SILPA project done by Swathanthra Malayalam Computing; This project, in short, enhances the Prathyaya rules, incorporates stemmer with document comparison module and perform comparison.

The step involved in document comparison are

1. Stemming
2. Tokenizing document to n-grams
3. Comparing
4. Calculating Similarity
5. Displaying result

**1. Stemming**

Stemming is the process of finding out root word of a token. It is a method to overcome the issue of ***inflection*** which is present in Indic languages, especially in Dravidian family of languages. Inflection is the

Inflection affects the character to character comparison in document similarity as different word forms may induce same meaning in Malayalam.

For eg:

അവൻ തൃശ്ശൂര് പോയി and

അവൻ തൃശ്ശൂരിൽ പോയി

both induce the same meaning “He went to Thrissur”. But when performing character to character comparison, the comparator will give the result that തൃശ്ശൂര് and തൃശ്ശൂരിൽ are different. So, we perform a rooting and we get the following root token set for both documents [അവൻ, തൃശ്ശൂർ, പോയി ]. When they are compared we can easily identify them as duplicates.

The steps involved designing stemmer are

1. Defining rules for stemmer in an external file
2. Word Suffixing
3. Check for existence

**Defining rules –** An external file is defined with the syntax <existing suffix> = <root word suffix>. A sample of the file is described below

#അനുസ്വാരത്തിലവസാനിക്കുന്ന ക്രിയ/നാമം

ത്തില്‍ = ം

ത്ത്=ം

ത്തു്=ം

ത്തു=ം

ത്തെ = ം

വുമായി = ം

ത്തിനെ=ം

ത്തിലെ=ം

This should be implemented for all existing Prathyaya rules and inflections.

**Word Suffixing –** The document is split to words and tokens are generated.. These tokens are sliced with variable length I starting from 1 to length of the word. At, each execution of the loop, the sliced word is checked for existence in the rules dictionary.

**Check for existence –** If the slice is found in the dictionary, the corresponding value from the dictionary is used to replace the sliced word's suffix. Else, the original word is copied to the token set.

This stem basically depends on the trained rules dictionary.

**2. Tokenizing document to words**

In the fields of computational linguistics and probability, an ***n*-gram** is a contiguous sequence of *n* items from a given sequence of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application. The *n*-grams typically are collected from a text or speech corpus. “n” denotes the number of tokens to be considered in each token set.

For eg: consider the string

കേരളത്തിൽ ധാരാളം നദികളും കുളങ്ങളും ജലാശയങ്ങളും ഉണ്ട്.

A bi-gram of this string will be

[{കേരളത്തിൽ ധാരാളം}, {ധാരാളം നദികളും}, {നദികളും കുളങ്ങളും}, {കുളങ്ങളും ജലാശയങ്ങളും} {ജലാശയങ്ങളും ഉണ്ട്}]

where each token contains two words.

Similarly, tri-grams will contain three words per token.

Sindhu et al. after comparing different n-grams calculated that tri-grams provide the most efficient result when performing document comparison [1].

After tokenizing, we sort the tokens in the ascending order of characters. It is done to overcome the problem of order of words in a sentence.

**3. Comparing**

After sorting the tokens, we run a loop over all the token sets, in each loop comparing them with their counterparts in the second document. The number of similar tokens are stored in a variable.

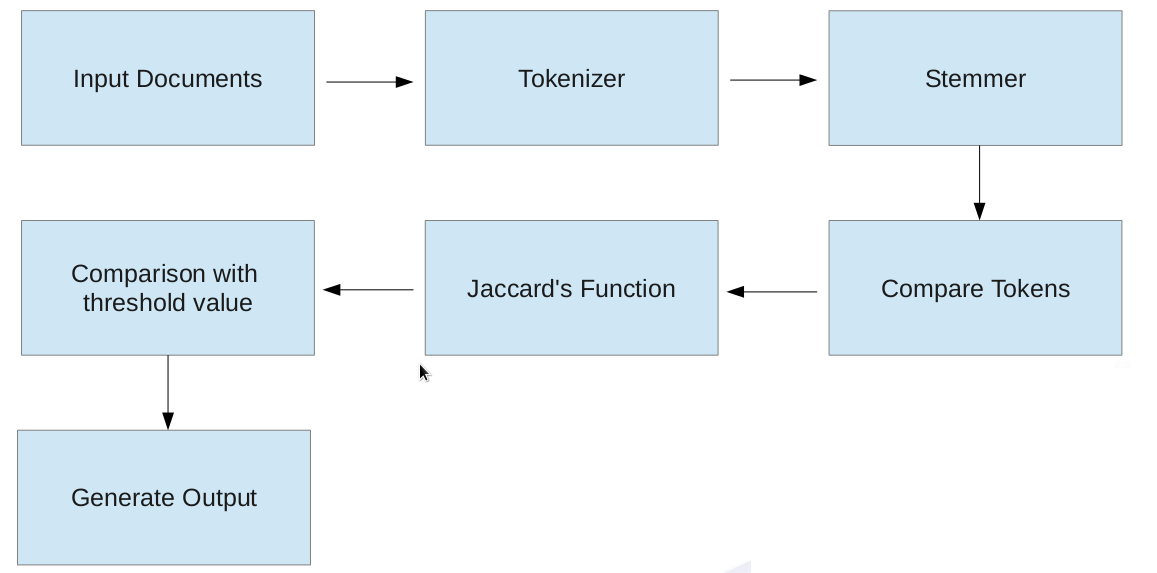
**4. Calculating Similarity**

The similarity between documents are calculated using a function called Jaccard's similarity.

f1 ∩ f2 denoted the token sets which are common in both documents and f1 U f2 denotes the union of unique tokens in both sets.

**5. Displaying Result**

The result of the Jaccard's function is displayed in the percentage form to denote the percentage similarity of both documents.

****

**OBJECTIVES**