1. 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.

2. 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 graph-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 DERIVATIVE 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. Derivative stemmer has average accuracy of 70.7%
6. Useful in dictionary search and data compression
7. Expansion using Named Entity Recognizer integration

3. 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. Tokenise 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.

4. 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

4.1. 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

4.1.1 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

ത്തിൽ = ◌ം

ത്ത് =◌ം

ത്ത് =◌ം

ത്ത =◌ം

ത്തെ = ◌ം

വുമായി = ◌ം

ത്തിനെ=◌ം

ത്തിലെ=◌ം

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

4.1.2 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.

4.1.3 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.

4.2. 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.

4.3. 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.

4.4. CALCULATING SIMILARITY

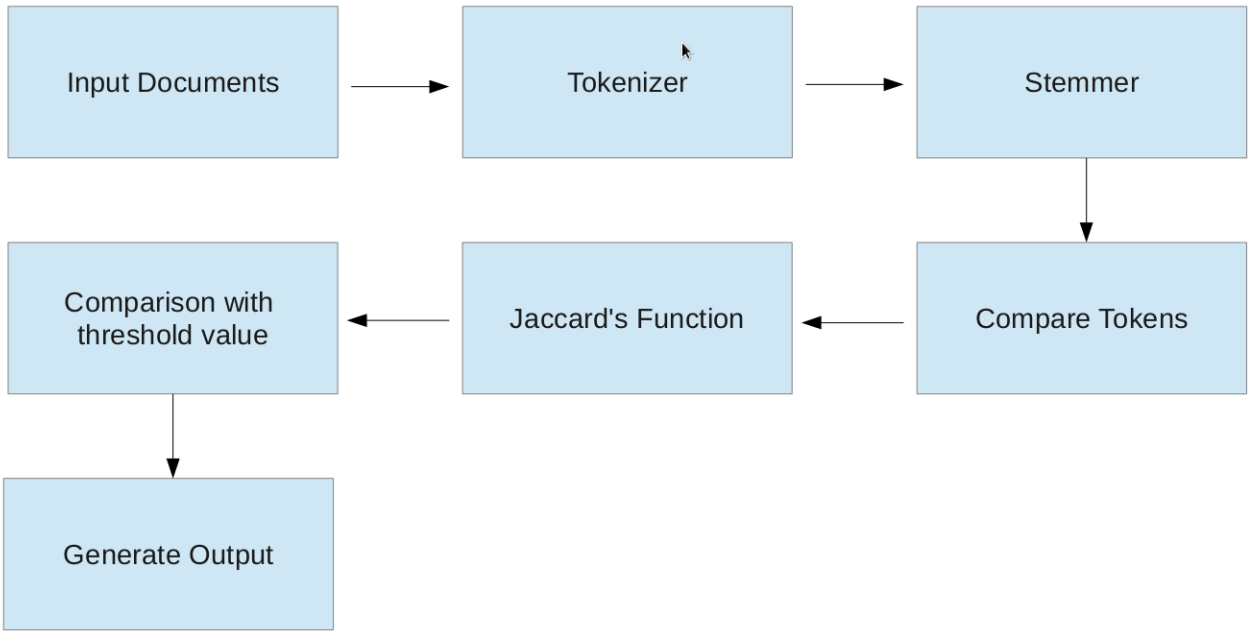
The similarity between documents are calculated using a function called Jaccard's similarity.

$$(d1, d2) \equiv J(d1, d2) \equiv \frac{f1 \cap f2}{f1 U f2}$$

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.

4.5. DISPLAYING RESULT

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



5. OBJECTIVES

The main objectives of this project are as follows

1. Improve the stemmer rules defined for Malayalam
2. Perform stemming for all the words in a document
3. Design an algorithm to compare token sets in two documents for near-duplication

6. HOW OBJECTIVES ARE TO BE TACKLED?

In order to tackle the specified objectives, the following steps are proposed

1. To improve the stemming rules, define rules for all Prathyaya forms and Sandhi rules of Malayalam and define how the suffix replacement has to be done.
2. Stemming has to be done over the complete document using any iterative statement.
3. Once stemming is done, the document is tokenized to trigrams. These form the token set of the document. Similarly, a token set is generated for the second document also. Then, corresponding token sets are compared for similarity. The number of similar tokens are recorded and used in Jaccard's formula to calculate the similarity. The result of Jaccard's function is converted to percentage similarity form and displayed to the user.

7. TIME SCHEDULE

WEEK 1(Sep 2- Sep 8)

- Different areas such as image processing, natural language processing etc were considered for the project.
- Finally natural language processing was selected as the area

WEEK 2 (Sep 9- Sep 15)

- Different topics in the area were discussed with the internal guide

WEEK 3 (Sep 16- Sep 22)

- Finalized the project topic "Document Similarity Detection for Indic Languages"
- Platform decision was taken
- Requirements for the project was found out.

WEEK 4 (Sep 23-29)

- A discussion was conducted with HOD
- First phase of literature survey was completed

WEEK 5 (Sep 30- Oct 6)

- Second phase of literature survey was completed
- Found out the improvements to be done on the shingling algorithm

WEEK 6 & 7 (Oct 7- Oct 13)

- Familiarization of existing system – SILPA

WEEK 7 (Oct 14- Oct 21)

- Familiarization of existing system – SILPA

WEEK 8(Oct 22- Oct 30)

- First review of the project was done.

WEEK 9(Oct 31- Nov 7)

- Basic idea of implementation was decided

WEEK 10(Nov 8 - Nov 15)

- Python Training

WEEK 11(Nov 16 - Nov 23)

- Python Training

WEEK 12(Nov 24 – Dec 1)

- Python Training

WEEK 13(Dec 2 – Dec 9)

- Discussion with HOD about the updates

WEEK 14(Dec 10 - Dec 17)

- Python Training

WEEK 15(Dec 18 - Dec 25)

- Python Training

WEEK 16(Dec 26 – Jan 2)

- Python Training

WEEK 17(Jan 3 – Jan 10)

- Python Training

WEEK 18(Jan 11 – Jan 18)

- Improving existing rules for Stemmer – Defining Pratyaya And Sandhi Rules

WEEK 19(Jan 19 – Jan 26)

- Designing Stemmer algorithm using the defined rules

WEEK 20(Jan 27 – Feb 3)

- Implementing stemmer algorithm and existence checking algorithm.

WEEK 21(Feb 4 – Feb 11)

- Implementing Document Comparison algorithm and Jaccard's Function
- Discussion with HOD

WEEK 22(Feb 12 – Feb 19)

- Implementing Document Comparison algorithm and Jaccard's Function

WEEK 23(Feb 20 – Feb 27)

- Designing GUI for the application

WEEK 24(Feb 28 – Mar 7)

- Designing GUI for the application

WEEK 25(Mar 7 – Mar 14)

- Testing and Debugging

8. COST ESTIMATE

The expected expense for the complete implementation of the project is summarized as follows

1. Cost of Training	–	Rs. 2000 x 3	=	**Rs. 6000**
2. Electricity	–			**Rs. 2000**
3. Transportation and other miscellaneous cost			–	**Rs. 2000**
Total	**–**			**Rs. 10000**

9. RISK FACTOR

The potential risks involved in the project are

1. Risk of software crash and data loss.
2. Risk on accidental manipulation of input data.
3. Risk of incorrect identification of idea of the document from root words – Logical Risk

10. TEST PLAN AND TEST DATA

Software Testing is a critical element of the software development cycle. The testing is essential for ensuring the quality of the software developed and represents the ultimate view of specification, design and code generation. Software testing is defined as the process by which one detects the defects in the software. Testing begins as the module level and work towards the integration of the entire computer based system.

A good test case is one that has high probability of finding as a yet undiscovered error. A successful test is one such uncover errors in the software. It also demonstrates that a software functions are being performed according to specification and also behavioral and performance requirements are satisfied. For this, test plans have to be prepared .the implementation of a computer system requires that test data to be prepared. Nothing is complete without testing as it is vital success of the system.

10.1 LEVELS OF TESTING:

10.1.1 Unit testing:

In this, different modules are tested against the specification produced during design for the modules. Unit testing is essentially for verification of the code produced during the coding phase, and hence the goal is to test the internal logic of the modules. It is typically done by the programmer of the module. A module is considered for integration and use by others only after it has been unit tested satisfactorily.

10.1.2 Integrated testing:

In this, many unit tested modules are combined into subsystems, which are then tested. The goal here is to see if the modules can be integrated properly. Hence, the emphasis is on testing interfaces between modules. This testing activity can be considered testing the design.

10.1.3 System testing:

Here the entire software system is tested. The reference document for this process is the requirement document, and the goal is to see if the software meets its requirements. This is essentially a validation exercise, and in many situations it is the validation activity.

10.1.4 Acceptance testing:

It is sometimes performed with realistic data of the client to demonstrate that the software is working satisfactorily. Testing here focuses on the external behavior of the system; the internal logic of the program is not emphasized.

10.1.5 Regression testing:

It is performed when some changes are made to an existing system. Modification have been made to an existing system, testing also has to be done to make sure that modification has not had any undesired side effect of making some of the earlier services faulty. That is besides ensuring the desired behavior of the new services, testing has to ensure that the desired behavior of the old services is maintained, this is the task of regression testing.

10.2 TEST PLAN

Testing is done by providing two documents whose percentage similarity is known earlier. The difference from the actual result obtained and the expected result is taken as a measurement of efficiency of the project. An example of test data is as follows.

10.3 TEST DATA

INPUT	EXPECTED OUTPUT
Exact duplicate documents. Eg: Document 1 : അവൻ തൃശ്ശൂരിൽ പോയി Document 2 : അവൻ തൃശ്ശൂരിൽ പോയി	Document 1 Before Stemming – ['അവൻ', തൃശ്ശൂരിൽ', 'പോയി'] Tokens – ['അവൻ', 'തൃശ്ശൂർ' ,'പോയി'] Document 2 Before Stemming – ['അവൻ', തൃശ്ശൂരിൽ', 'പോയി'] Tokens – ['അവൻ', 'തൃശ്ശൂർ' ,'പോയി'] Output Similarity : 100%

Near Duplicate Documents Eg : Document 1 : അവൻ തൃശ്ശൂർ പോയി Document 2 : അവൻ തൃശ്ശൂരിൽ പോയി	Document 1 Before Stemming – ['അവൻ', 'തൃശ്ശൂർ' ,'പോയി'] Tokens – ['അവൻ', 'തൃശ്ശൂർ' ,'പോയി'] Document 2 Before Stemming – ['അവൻ', തൃശ്ശൂരിൽ', 'പോയി'] Tokens – ['അവൻ', 'തൃശ്ശൂർ' ,'പോയി'] Output Similarity : 100%
Distinct Documents Eg: Document 1 : അവൻ കാലടിയിൽ നിന്നും വന്നു Document 2 : അവൻ തൃശ്ശൂരിൽ പോയി	Document 1 Before Stemming – ['അവൻ', 'കാലടി' ,'നിന്നും' 'വന്നു'] Tokens – ['അവൻ', 'തൃശ്ശൂർ' ,'പോയി'] Document 2 Before Stemming – ['അവൻ', തൃശ്ശൂരിൽ', 'പോയി'] Tokens – ['അവൻ', 'തൃശ്ശൂർ' ,'പോയി'] Output Similarity : 20%

11. CONCLUSION

This project presents a document similarity comparison mechanism for Malayalam text using trigrams and Shingling Algorithm. A stemmer is used to handle the problem of inflection that exists in all Dravidian languages including Malayalam. This has been a major hurdle for the existing systems. In this project, an existing stemmer in improvised by adding more Pratyaya and Sandhi rules and other use cases. This stemmer is used to identify the root words of each tokens of the document and then these root words are compared. Jaccard's similarity function is used to calculate the similarity between documents and the result is displayed in the percentage form. Trigrams are used because they are found out to be the most efficient n-gram models in text comparison. Nevertheless the existing systems can provide significant speedup gain comparing to semantic-based methods especially for large data sets since the comparison does not involve deeper analysis of the structure and/or the semantics of terms.

In future, we plan to extend this work by

- Checking for similarity where words have been replaced by similar words
- Highlighting similar parts of the document
- Implementing an algorithm based stemmer rather than a rule based one

12. REFERENCES

[1].A Copy detection Method for Malayalam Text Documents using N-grams Model - Sindhu.L, Bindu Baby Thomas and Sumam Mary Idicula

[2].A Simple Stemmer for Inflectional Languages - Jiaul H. Paik and Swapan K. Parui

[3].YASS: Yet Another Suffix Stripper - Prasenjit Majumder, Mandar Mitra, Swapan K. Parui, And Gobinda Kole

[4].Malayalam Stemmer - Vijay Sundar Ram R, Pattabhi R K Rao T and Sobha Lalitha Devi

[5].An Unsupervised Approach To Develop Stemmer - Mohd. Shahid Husain - International Journal on Natural Language Computing (IJNLC) Vol. 1, No.2, August 2012

[6].A Literature Review: Stemming Algorithms for Indian Languages - M.Thangarasu, Dr.R.Manavalan - International Journal of Computer Trends and Technology (IJCTT) – volume 4 Issue 8–August 2013

[7].Stemmers for Tamil Language: Performance Analysis - M.Thangarasu, Dr.R.Manavalan - International Journal of Computer Science & Engineering Technology (IJCSET)

[8].A Survey of Common Stemming Techniques and Existing Stemmers for Indian Languages - Vishal Gupta , Gurpreet Singh Lehal - Journal Of Emerging Technologies In Web Intelligence, Vol. 5, No. 2, May 2013

[9].A Computational Phonetic Model for Indian Language Scripts - Anil Kumar Singh

[10]. Pairwise Document Similarity in Large Collections with MapReduce - Tamer Elsayed, Jimmy Lin and Douglas W. Oard - Proceedings of ACL-08: HLT, Short Papers (Companion Volume), pages 265–268, Columbus, Ohio, USA, June 2008. c 2008 Association for Computational Linguistics

[11]. Hybrid Inflectional Stemmer and Rule-based Derivational Stemmer for Gujarati - Kartik Suba, Dipti Jiandani and Pushpak Bhattacharyya

[12]. A Lightweight Stemmer for Hindi - Ananthakrishnan Ramanathan and Durgesh D Rao