INFORMATION RETRIEVAL ACROSS INDIC SCRIPTS

A PROJECT REPORT

Submitted by

ARVINDKUMAR K 2012115517

SOUNDHARYA M 2012115569

SHAHNAZ S 2012115605

to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION SCIENCE AND TECHNOLOGY

COLLEGE OF ENGINEERING,GUINDY

ANNA UNIVERSITY

CHENNAI 600 025

MAY 2016

ANNA UNIVERSITY CHENNAI

CHENNAI - 600 025

BONA FIDE CERTIFICATE

Certified that this project report titled INFORMATION RETRIEVAL ACROSS INDIC SCRIPTS is the bona fide work of ARVINDKUMAR K, SOUNDHARYA M and SHAHNAZ S who carried out project work under my supervision. Certified further that to the best of my knowledge and belief, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion on this or any other candidate.

PLACE: Chennai
DATE:

DR. RANJANI PARTHASARATHI
PROJECT GUIDE
DEPARTMENT OF IST, CEG
ANNA UNIVERSITY
CHENNAI 600025

COUNTERSIGNED

DR. SASWATI MUKHERJEE
HEAD OF THE DEPARTMENT
DEPARTMENT OF IST, CEG
ANNA UNIVERSITY
CHENNAI - 600025

ABSTRACT

With the rapid growth of search engines, detecting multilingual intent underlying search queries has become a critical challenge to serve international users with diverse language requirements. A country like India, with 22 official languages including English needs a search engine that can understand the complexity and nuances of multi-script search.

Information Retrieval across Indic Scripts (IRIS) deals with asking questions in one or more languages and retrieving documents in one or more different languages. With an increasingly globalized economy, there is a strong necessity to find information in other languages.

This project presents our research in developing and evaluating a multilingual Indic script web search engine concentrating on English, Hindi and Tamil. A query pre-processing approach that uses Natural Language Processing (NLP) techniques has been adopted that combines Multiword Expression Recognition, Named Entity Recognition, Query Transliteration and Query Translation.

To improve web search quality, the project also proposes a set of new ranking features based on query intent by applying classification and heuristic techniques on the multilingual query. The experimental results are evaluated to analyze and compare the performance of the Google IR system with that of the IRIS search engine. Experimental result shows that the effective retrieval and performance of the IRIS search engine has improved by 13.2% over the Google IR system.

திட்டபணிசுருக்கம்

தேடல் இயந்திரங்கள் விரைவான வளர்ச்சி அடைந்த இக்காலத்தில், பன்மொழி தேடல் நோக்கம் கண்டறிவது ஒரு முக்கியமான சவாலாக மாறிவிட்டது. ஆங்கிலம் உட்பட 22 உத்தியோகபூர்வ மொழிகளை உபயோகிக்கும் இந்தியா போன்ற ஒரு நாட்டில், சிக்கலான மற்றும் பன்மொழி தேடல் நுணுக்கங்களை புரிந்து கொள்ளும் ஒரு தேடல் இயந்திரம் தேவை. இண்டிக் ஸ்கிரிப்டில் தகவல் மீட்பு (ஐரிஸ்) என்பது , ஒன்று அல்லது அதற்கு மேற்பட்ட மொழிகளில் கேள்விகளை கேட்டு , ஒன்று அல்லது அதற்கு மேற்பட்ட வெவ்வேறு மொழிகளில் ஆவணங்களை மீட்பதை மேற்கொள்கின்றது. அதிகரித்துவரும் பொருளாதரத்தில் , பிற மொழிகளில் உள்ள தகவல்களை கண்டுபிடிக்க ஒரு வலுவான தேவை உள்ளது.

இந்த திட்டத்தில் ஆங்கிலம், தமிழ் மற்றும் இந்தியில் ஒரு பன்மொழி தேடல் இயந்திரத்தை உருவாக்கவும் மதிப்பிடவும் எங்கள் ஆராய்ச்சி வழங்குகிறது. இத்திட்டம் ஒரு கேள்வியின் முன் செயலாக்கம் அணுகுமுறையை இயற்கை மொழி செயலாக்கம் நுட்பங்களாகிய சொற்றொடர் எக்ஸ்பிரஷன் அங்கீகாரம், பெயரிடப்பட்ட நிறுவனத்தின் அங்கீகாரம், கேள்வி ஒலிபெயர்ப்பு மற்றும் கேள்வி மொழிபெயர்ப்பு ஆகியவைகளை பயன்படுத்துகிறது. தேடல் தரத்தை மேம்படுத்த இத்திட்டம் , பன்மொழி கேள்வி வகைப்படுத்தலை மற்றும் பட்டறிவு தொழில்நுட்பங்களை பயன்படுத்துவதன் மூலம் கேள்வி நோக்கத்தின் அடிப்படையில் புதிய தரவரிசை அம்சங்களின் ஒரு தொகுப்பை முன்மொழிகிறது.

சோதனை முடிவுகளை கூகில் ஐஆர் அமைப்பின் செயல்திறனோடு ஐரிஸ் தேடல் இயந்திரத்தின் செயல்திறனுடன் ஒப்பிட்டு மதிப்பீடு செய்யப்படுகின்றன. சோதனையின் கண்டுபிடிப்பில் ஐரிஸ் தேடல் இயந்திரத்தின் மீட்பு மற்றும் செயல்திறன் , இருக்கும் கூகில் ஐஆர் அமைப்பைவிட 13.2% அதிகரித்து இருக்கிறது என்பதை நிரூபிக்கிறது.

ACKNOWLEDGEMENT

We would like to express our sincere thanks and a deep gratitude towards our guide **Dr. Ranjani Parthasarathi**, Professor, Department of Information Science and Technology, Anna University, Chennai for guiding us through every phase of our project. We are thankful to her thoroughness, tolerance ability to share her knowledge with us. We owe her for harnessing our potential and bringing out the best in us. Without her immense support and encouragement, we could never have never made it to this extent.

We are extremely grateful to **Dr. Saswati Mukherjee**, Professor and Head of Department of Information Science and Technology, Anna University, Chennai for extending the facilities of the Department towards our project and for her untiring support.

We would like to express our thanks to the project review committee members **Dr. K. Vani**, Associate Professor, **Dr. S. Bama**, Assistant Professor, **Dr. E. Uma**, Assistant Professor and **Ms. Tina Esther Trueman**, Teaching Fellow for their valuable scrutiny and critical reviews throughout the course of our project.

We also thank our parents, Family and friends for bearing with us during the course of our project and for the opportunities they provided us in undergoing this course in such a prestigious institution.

Arvindkumar K **Soundharya M** **Shahnaz S**

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS AND SYMBOLS

CLIR	Cross Language Information Retrieval
IR	Information Retrieval
MWE	Multi word Expression
NE	Named Entity
NER	Named Entity Recognizer
NL	Natural Language
NLP	Natural Language Processing
SVM	Support Vector Machines
QI	Query Intention
P	Positive
N	Negative
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
$\sum$	Summation
$\cap$	Intersection

CHAPTER 1

INTRODUCTION

1.1 NATURAL LANGUAGE PROCESSING

Natural Language processing (NLP) is a field of computer science and linguistics concerned with the interactions between computers and human languages. Natural language processing is by far successful in meeting the challenges as far as syntax is concerned. But it still has to go a long way in the areas of semantics and pragmatics. The field of NLP has witnessed a phenomenal growth of interest in recent years, both in academic research space and in the industry scene. Important applications of NLP include machine translation, information retrieval, question answering system, named entity recognition, etc.

With the enormous increase in recent years in the number of text databases available on-line, and the consequent need for better techniques to access information of different types and languages, there has been a strong resurgence of interest in the research done in the area of information retrieval. Information retrieval essentially involves returning a set of documents in response to a user query by storing, searching and fetching information. IR relies on NLP methods like stemming, tokenizing, multi-word recognition, stop-word elimination, etc. all with the aim of comparing and presenting the user with documents that best satisfy their information needs.

1.1.1 Need for Multilingual Information Retrieval

The need for IR has transformed into much more specific needs. Massive search engines like Google and Bing have multiple filters that can help in narrowing down the results. However, with the growth of the multi-lingual nature of web content, the multiple languages and content from distant countries create new opportunities for finding previously buried information resources for savvy searchers. To find these resources, even in languages that one cannot read, the search engine must offer a variety of language aids.

Documents and web pages are written in many different languages; some even have content in multiple languages. A variety of characters are used to express the words of these languages: the Latin alphabet, diacritics (i.e., accented characters), ligatures and others. This is particularly relevant in a diverse country like India which recognizes 23 official languages. The Internet users in India are ranked highest in the world, after China.

It is therefore crucial to improve retrieval of multi-lingual information. The complexity of a search engine lies in the nature of the query. If the query contains multiple scripts and languages, the modules of Information retrieval also need to adapt to this complexity in different ways. There are many approaches to multi-lingual information retrieval; Cross Language Information Retrieval (CLIR) using machine translation of the results and Information Retrieval of a multi-lingual query through query translation and transliteration. While machine translation translates the retrieved results and presents it in one language, the latter translates and transliterates the query retrieving a larger pool of multi-lingual results. This project, **Information Retrieval across Indic Scripts (IRIS)** is about implementing a web search engine that can retrieve information belonging to multiple scripts and languages according to the second approach.

1.2 PROBLEM STATEMENT

This project proposes to develop an Information Retrieval System focusing on two Indic scripts (Tamil and Hindi) and to solve the various challenges faced during the retrieval of information across scripts. The main task lies in working with complications of multi-lingual and multi-script queries like spelling variation, term matching and language identification and build a system that can predict the user intention and provide results ranked accordingly.

1.3 IRIS- AN OVERVIEW

IRIS is a search engine developed to take queries in Tamil Indic Script, Hindi Indic Script and a combination of Hindi, Tamil and English. Search engines like Google can take multi-lingual queries, however the relevance of their results suffer as most of their language analysis methods do not work efficiently on many diverse languages. It involves three phases; the user query is first preprocessed using the IRIS language analyzer. It is then passed to the Google search engine through our intermediary web interface, containing variations that enable the engine to retrieve relevant information. The results are then ranked according to a Query Intention algorithm that analyzes the query using classification and heuristic understanding.

1.4 OUTLINE OF THE REPORT

The report is organized as follows. Chapter 1 gives an introduction to the project. Chapter 2 provides details about the various existing work related to Information Retrieval systems. Chapter 3 discusses about the system architecture and the various modules in the proposed system. Chapter 4 explains the detailed implementations of the corresponding module. Chapter 5 gives the details of the results obtained and the measures to analyse the efficiency of the proposed system. Chapter 6 provides the conclusion of the report.

CHAPTER 2

RELATED WORK

This chapter describes the basics and reviews the work being done in the various techniques associated with Information Retrieval across Indic Scripts.

2.1 QUERY PRE-PROCESSING

2.1.1 Language Identification

King & Abney (2013) tested many techniques for language identification. They used Hidden Markov Model trained with Expectation Maximization and Conditional Random Field Model trained with Generalized Expectation Criteria for identifying the minority languages like Sotho, Uzbek, Oromo, etc. present in a multi-lingual query. They found that Conditional Random Field Model trained with Generalized Expectation Criteria was the most accurate and performed consistently with the multi-lingual testing data.

2.1.2 Multiword Expression and Named Entity Recognizer

Janarthanam & Sethuramalingam (2008) proposed a method for recognizing Named Entities in a Cross-Language Information Retrieval system. Using Compressed Word Format (CWF) with Modified Levenshtein algorithm,

the query words are compressed without losing essential information that is needed for transliteration. It is then checked with the entries in the dataset and mapped to an entry that has least edit distance. This proposal for Named Entity Recognizer is proven to be more accurate without losing precision.
Morwal Jahan & Chopra (2012) proposed a Hidden Markov model based Named Entity Recognizer (NER) system that uses learning by example methodology. It provides easy to use method with minimum efforts for Named Entity Recognition in any natural language. The user has to just annotate his corpus and test the system for any sentence. Steps followed for any language are data preparation, parameter estimation (training) and testing the system.

2.1.3 Stemmer

Gupta (2014) discusses Hindi stemmer for nouns. The proposed stemmer applies a suffix stripping rule based approach for performing stemming. The accuracy achieved for this stemmer is 84%.
Thangarasu & Manavalan (2013) have compared a Light stemmer and a rule based suffix removal stemmer. The experimental results clearly show that their proposed approach, the Light stemmer for Tamil language perform better than suffix removal stemmer and prove effective in Information Retrieval Systems.

2.1.4 Transliteration

Arokia Raj & Maganti (2009) discussed Indic script transliteration as a pre-processing scheme. Font-Encoding Identification and Font-Data Conversion techniques are elaborated to show that a meta standard transliteration scheme makes Indian language web content more accessible in a search engine.

Gupta et al. (2014) presented a principled solution for handling spelling variation and transliteration mining. A multi-script joint model was constructed such that it learns abstract representation of terms across the scripts through deep-learning architecture with terms equivalent close to each other. The deep auto-encoder based approach provided highly discriminative and powerful representation for terms with the abstract space dimensionality as low as 20, compared to Naive, Canonical Correlation Analysis, Latent Semantic Indexing, Editex. An extensive empirical analysis of experiments was presented with a practical and important use-case, ad-hoc retrieval of song lyrics.

2.1.5 Dictionary Analysis

Ballesteros & Croft (1997) provided a comparison between Query Expansion using Automatic Dictionary translations, phrasal translation and via local feedback and local context analysis. The first two are highly effective with monolingual queries. Cross Language Information Retrieval gives higher precision at low recall values.

2.2 CROSS LANGUAGE IR

Arokia Raj & Maganti (2009) proposed a search engine for Indian Languages called inSearch. It had a language focused crawler and an index structure that includes syllables, stem, word, term frequency and language. This structure enabled efficient multi-lingual and cross-lingual search. Ranking of results were based on the phonetic structure of the query. The proposed search engine worked effectively on 10 Indian languages.

2.2.1 Solr/Nutch Based Multi-lingual Search Engine

Atreya & Bhattacharyya & Ramakrishnan (2012) presented a comparison of open source search engine development frameworks in the context of their malleability for constructing multi-lingual search index. They proposed a latest version of Nutch with Hadoop to be used as a crawler and Apache Solr for building the index of crawled documents. Both Nutch and Solr supports customization of modules to make them adaptable to a desired application. However, this paper raises challenges of a multi-lingual search engine; having separate index for every language.

2.3 RANKING BY QUERY INTENTION

Web query classification (QC) is widely studied to understand the user's search intent. User queries are classified into predefined categories like informational, navigational, and transactional queries. This classification can be used to trigger appropriate results corresponding to a query and in turn improve web page ranking.

- Informational searching: The intent of informational searching is to locate content to address an information need of the searcher.
- Navigational searching: The intent of navigational searching is to locate a particular website. The searcher may have a particular Website in mind, or the searcher may just assume a particular website exists.
- Transactional searching: The intent of transactional searching is to locate a Website with the goal to obtain some other product, which may require executing some web service on that website. Examples

include purchase of a product, execution of an online application, or downloading multimedia.

2.3.1 Classification

Chang et al. (2011) proposed a set of new ranking features to combine multi-lingual and multi-regional query intent with document language/region attributes, and applied different approaches in integrating intent information to directly affect ranking. This query intent approach combined clicks for popular queries with language models for smoothing unseen queries.

Tawfik & Kamel (2014) proposed a filter to pick queries that may benefit from a Cross Language Information Retrieval System. This project maintained a large list of intents like books, entertainment, sports etc. i.e., it employed a fine grained intent classification on Arabic queries. The original query is classified for intent followed by the classification of its translated query. Both intents are merged based on a probabilistic model. This intent-based approach showed gains in two different regions with clearly different intent profile.

2.4 LIMITATIONS OF THE EXISTING WORK

Most multi-lingual search engines deal with multiple languages, however these predominantly belong to the Roman script. Information retrieval across Indic scripts brings in new challenges as language analysis must be developed to pre-process multiple scripts. Countries like India with 23 different languages along with English as an official language present a new challenge. A query belonging to a certain language expects results in a different language.

These query indications are not extensively studied with regard to multi-lingual IRs and can pose an important challenge.

2.5 OBJECTIVES OF THE PROPOSED WORK

- To develop an Information Retrieval system across Indic Scripts that can retrieve relevant results for multi-lingual and multiscript queries.
- To utilize and shape standard NLP tools for Tamil and Hindi language analysis.
- Train classifiers and employ heuristics to make sure that users get the best results in the language they want, i.e., improve existing query intention techniques.
- To develop multi-lingual search capabilities on top of an existing search engine to improve accuracy.

CHAPTER 3

SYSTEM DESIGN

This chapter describes the architecture of the proposed IRIS system and the various functional modules of the system.

3.1 SYSTEM ARCHITECTURE

The system architecture of IRIS is depicted in Figure 3.1.

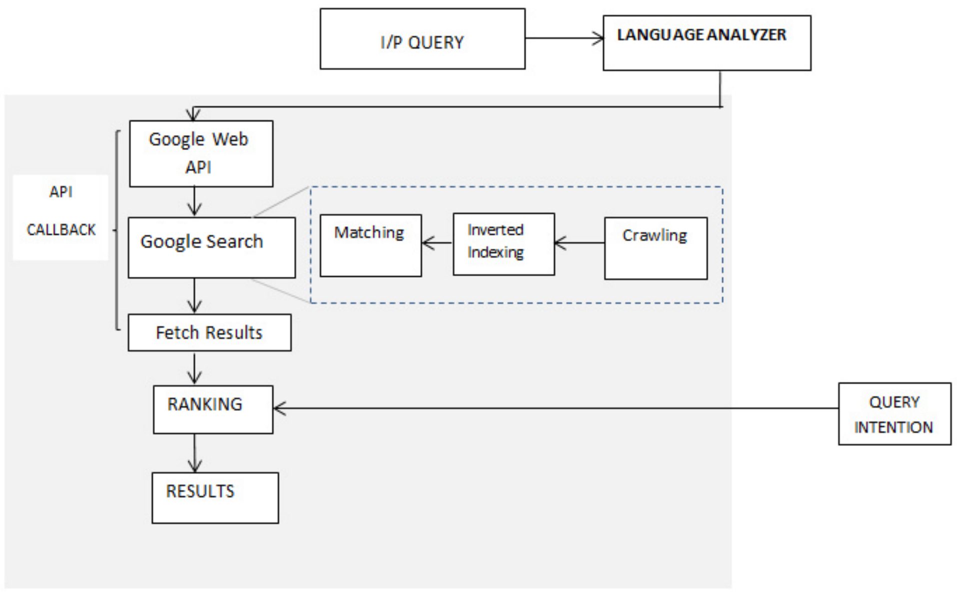

Figure3.1: System Architecture of the IRIS System

IRIS involves three modules. Firstly, the query pre-processing phase where the language analyzer generates three forms of the query; original query, transliterated query and translated query. These queries are passed to the IRIS Search engine which makes a call to the Google API that fetches the search results. The retrieved results are ranked based on Query Intention. Query Intention uses a combination of classifiers and a heuristic algorithm to identify the priority of ranking.

3.2 DETAILED DESIGN

The detailed design of IRIS is depicted in Figure 3.2.

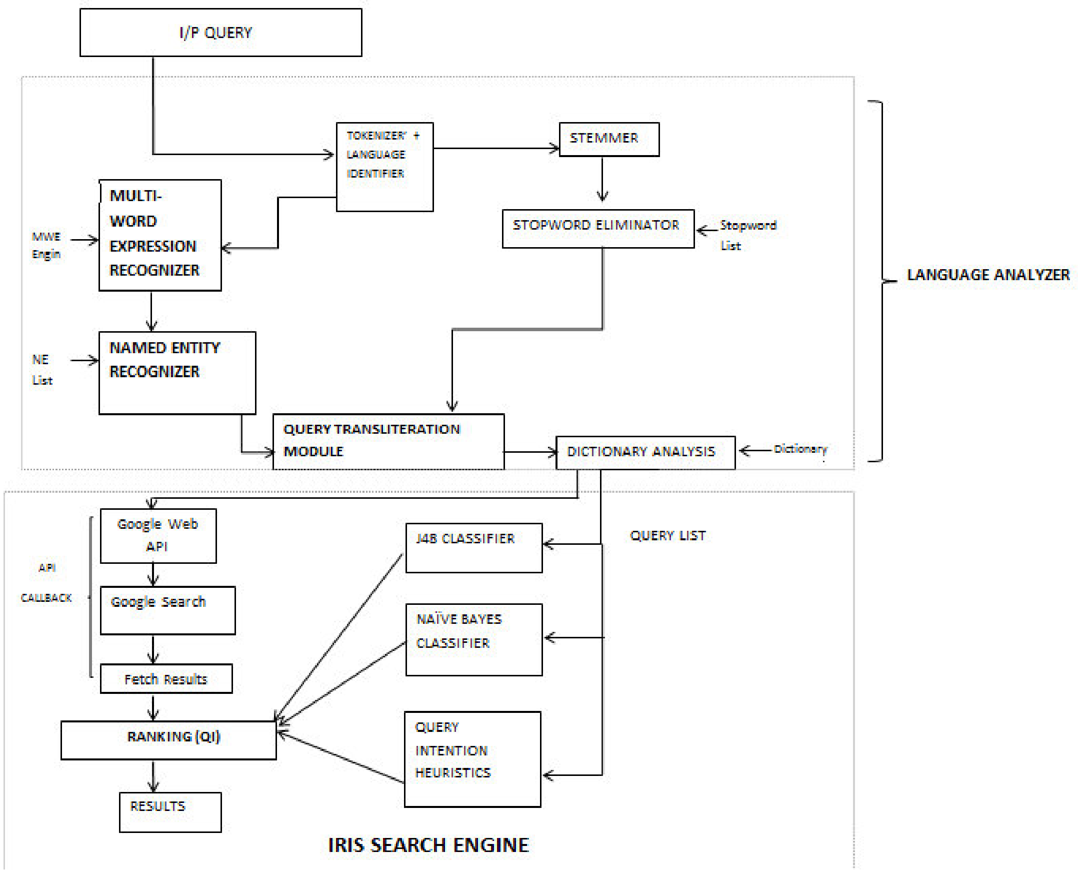

Figure3.2: Detailed Design of the IRIS System

The IRIS search engine involves three major tasks; pre-processing/language analysis followed by information retrieval using the search engine and ranking results based on intent. When the language analyzer receives a query as input, it is first tokenized and the language of input is identified. It is then checked for multi-word expressions and named-entities. Simultaneously this query is stemmed and removed of stop-words. The language analyzer at this point identifies and tags all familiar characteristics and replaces query words with its root words. This pre-processed query is then transliterated (Hindi-English, Tamil-English) to broaden the potential of the results. Dictionary analysis will run the words of the query through a dictionary and generate meanings and translations. This pre-processing phase generates three versions of the query; the original query, the transliterated query and the translated query.

The intermediary user interface receives all three queries, and makes an API call to Google search, which searches through its corpus of documents and returns the individual set of results for each of the queries. The IRIS web interface then displays those results after ranking them according to the findings from the Query Intention algorithm.

The intention of the query and user is determined based on a heuristic approach. Using the query, language of output preference and type of result is predicted and ranked accordingly.

3.2.1 Language Analyzer

The language analyzer module pre-processes the query, which includes passing it through the following components: tokenizer, language identifier, stop word eliminator, stemmer, query transliteration and dictionary analysis.

Tokenizer

The basic operation of a tokenizer is to convert a multi-script query string into a sequence of tokens. Since we are dealing with multi-script languages; Hindi and Tamil are both languages that can be tokenized in a manner similar to English as white-space is used to separate words and both languages, conveniently do not employ hyphens, apostrophes etc. The input to a tokenizer is a string of words, sometimes a sentence. The tokenizer splits the input, separating the words as tokens. The tokenizer mainly helps in removing new-line characters and tabs, identifying and splitting strings that contain numbers attached to the words, processing URLs and splitting hyphenated words.

Stop-word Eliminator

Some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the query entirely. These words are called stop words. Using a stop list to eliminate stop words significantly reduces the number of postings that a system has to store, however web search engines generally do not use stop lists. This is done in order to support phrasal search. IRIS, however implements a stop word eliminator as phrasal search is taken care in parallel through the multi-word expression recognizer. A list of stop words in Hindi, Tamil and English that generally add no value to the query is maintained and these are eliminated from the query.

Stemmer

Stemming plays an important role in Information Retrieval for improving the performance of a language. The goal of a stemmer is to diminish the derivative forms of a word. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivative affixes. The most common algorithm for stemming English, and one that has repeatedly been shown to be empirically very effective, is Porter's algorithm explained in (Porter et al. 1980). The IRIS stemmer uses the Porter stemmer for English,(related words map to the same stem, even if this stem is not in itself a valid root), a Light stemmer for Hindi query words and the Snowball stemmer (Porter et al. 2001) with a patch for Tamil query words.

Multi-word Expression Recognizer

The multi-word expression recognizer commonly identifies noun phrases and noun compounds. For example, "Prime Minister" is considered a single entity yet, it is a multi-word. A MWE crosses word boundaries and exhibits particular syntactic behaviors. A large fraction of words in English are MWEs (41 % in Wordnet). Other languages like Tamil and Hindi too exhibit this behavior. MWEs can be classified based on syntactic forms and compositionality into institutionalized noun collocations, phrasal verbs, etc. Stanford CoreNLP (Stanford 2001) is used to identify English MWEs, while Hindi and Tamil multi-words are recognized from a compiled list of such words.

Named Entity Recognizer

A Named-Entity Recognizer labels sequences of words in a text which are the names of things, such as person and company names, or location names. The Stanford NER which is a Java implementation of a Named Entity Recognizer is not designed to identify named entities in Indic scripts. To identify Tamil and Hindi Named Entities, each word is first converted into a compressed format using the Compressed Word Format algorithm, and then compared with a list of Indian names which have all been compressed into the same format. The words are compared and the Levenshtein distance between the two is calculated. If the Levenshtein distance is found to be lesser than 1, which essentially means that there is only one letter different between the compressed formats of the two words, the word in question is said to be a named entity.

ALGORITHM 1: COMPRESSED WORD FORMAT

Input: Named entity

Output: Compressed word format of the named entity

```
1: Start with the token list $x$ []
2: for each $n$ of $x$ [] do
3:     Replace consonant digraphs representing one phoneme
4:     Rewrite double consonants
5:     Rewrite clusters based on target language
6: end for
```

[1]

ALGORITHM 2: LEVENSHTEIN DISTANCE

Input: Two word strings

Output: Minimum distance between the two strings

```
   Start
2: Get $string1$,$string2$
   for each $i$ : 1 to $string1.length$ do
4:     for each $j$ : 1 to $string2.length$ do
           $distance[i][j]$ = minimum($distance$[i−1][j] + 1, $distance$[i][j−1] + 1, $distance$[i−1][j−1] + ($string1$[i−1] == $string2$[j−1] ? 0 : 1));
6:     end for
   end for
```

[2]

Query Transliteration

Query Transliteration is a significant part of this subtask. This module involves replacing Tamil scripts in the query with transliterated Tamil text and Hindi scripts with transliterated Hindi text. Four different types of Unicode transliteration have been implemented in this module : Hindi to English, English to Hindi, Tamil to English, English to Tamil. Transliteration of the query is crucial for the IRIS search engine as it increases the relevance by returning transliterated results for multiscript queries.

Dictionary Analysis

This step includes translation of the query, which is the communication of the meaning of a source-language text by means of an equivalent target-language text. This is necessary in order to obtain more relevant results with respect to multilingual queries.

The importance of maintaining a pipelined language analyzer process in the above order is to ensure that information is retrieved in every way. For example, checking for named entities and multi-word expressions after tokenizing and stemming the query will prove useless as the query has already been normalized and reduced.

3.2.2 IRIS Web Search Engine

The IRIS Web Search engine needs to crawl, index, match and rank pages. This can be implemented using Nutch/Solr. However, a call to Google API using a web interface is a more efficient option as it provides the best platform to plug in the Query Intention algorithm and rank the web results.

Google API

A custom IRIS search engine interface is created using PHP and HTML script. This script makes a call to the WebSearch( ) method using the Google API key provided for each of the possible list of queries obtained from the pre-processing stage. At the end of each search query execution, the webSearchComplete( ) method is called which obtains the results and a snippet of its content and prints it in the required format.

3.2.3 Query Intention

This module focuses on two methods to predict user intent; classification and heuristical analysis.

Query Classification

Using a combination of classifiers, the input query is identified and classified as information, transactional or navigational by training a multilingual web query dataset in the same. Naive Bayes Classifier and J48 classifiers are used to classify the category that the input query belongs to based on the training data set.

Multilingual and multiscript search engines have an additional challenge. When a query is multilingual, it is important to identify the preferred language of results. A query like *meaning of vazhkai (meaning of life)* expects results in transliterated Tamil or English. However a query like *vazhkai oda artham thamizh la (meaning of life in tamil)* expects results in Tamil Indic script or Tamil (transliterated). To identify language of output, we isolate characteristics of queries in each language that can serve as identifiers for these types of queries. Language of output is predicted by the classifiers. However, this challenge needs a heuristic approach for additional accuracy.

Heuristical Analysis

In order for the prediction to be meaningful, defining characteristics are isolated and identified in each language. For example, queries that contain a majority of words that belong to transliterated Tamil expect results in transliterated Tamil. Common words like *download, artham, shabdh, venum* have also been determined and included in a list that indicates output language. A series of conditions are tested and the language of output is predicted. Query Intention provides two kinds of predictions; type of query and language of preferred output. These predictions are incorporated in the search engine by ranking the results accordingly.

ALGORITHM 3: QUERY INTENTION

Input: Input query

Output: Language of output

```
Start
Get *query*
Tokenize *query* into *tokens* []
**for** each *n* of *tokens* [] **do**
    Get language
**end for**
Count number of words in each language
**if** maximum words in *lang* **then**
    $x \leftarrow lang$
**end if**
**if** any *EnglishWordList* in *query* **then**
    $x \leftarrow English$
**else if** any *TamilWordList* in *query* **then**
    $x \leftarrow Tamil$
**else if** any *HindiWordList* in *query* **then**
    $x \leftarrow Hindi$
**end if**
```

3.3 CLASS DIAGRAM

The project consists mainly of eight classes; AnalyzerDriver, Tokenizer, Translate, Transliterator, SearchEngine, QueryIntention, Stemmer, CWF, LevDist, which are depicted below in Figure 3.3.

This figure demonstrates the relationships between the classes. The AnalyzerDriver contains the main function and is the central class from which all other functions are executed. The AnalyzerDriver class is associated with the Tokenizer class which contains the tokenize function and is responsible for tokenizing the query first. It then calls the removeStopWords function of the Tokenizer class and the stem function of the Stemmer class, followed by multiword and named entity recognition using the MWE_NER function. This in turn uses the CWF and LevDist classes to determine the presence of phrases and named entities. The query is then transliterated and translated using functions from the Transliterator and Translate classes. The input query is passed to the QueryIntention class to determine the language of output. The query list generated is passed to the SearchEngine class to fetch the Google results which are then ranked accordingly.

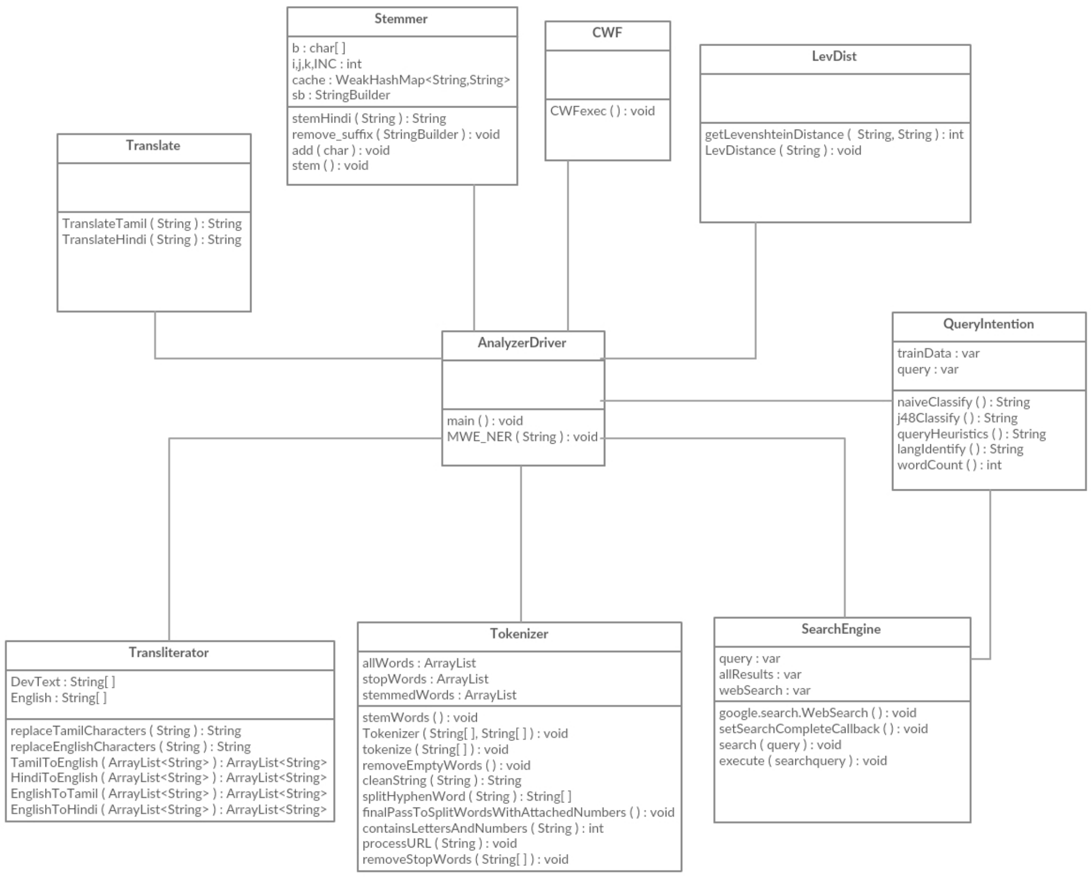
Stemmer
b : char[ ]
i,j,k,INC : int
cache : WeakHashMap<String,String>
sb : StringBuilder
stemHindi (String) : String
remove_suffix (StringBuilder) : void
add (char) : void
stem ( ) : void
CWF
CWFexec ( ) : void
LevDist
getLevenshteinDistance (String, String) : int
LevDistance (String) : void
Translate
TranslateTamil (String) : String
TranslateHindi (String) : String
AnalyzerDriver
main ( ) : void
MWE_NER (String) : void
QueryIntention
trainData : var
query : var
naiveClassify ( ) : String
j48Classify ( ) : String
queryHeuristics ( ) : String
langIdentify ( ) : String
wordCount ( ) : int
Transliterator
DevText : String[ ]
English : String[ ]
replaceTamilCharacters (String) : String
replaceEnglishCharacters (String) : String
TamilToEnglish (ArrayList<String>) : ArrayList<String>
HindiToEnglish (ArrayList<String>) : ArrayList<String>
EnglishToTamil (ArrayList<String>) : ArrayList<String>
EnglishToHindi (ArrayList<String>) : ArrayList<String>
Tokenizer
allWords : ArrayList
stopWords : ArrayList
stemmedWords : ArrayList
stemWords ( ) : void
Tokenizer (String[ ], String[ ]) : void
tokenize (String[ ]) : void
removeEmptyWords ( ) : void
cleanString (String) : String
splitHyphenWord (String) : String[ ]
finalPassToSplitWordsWithAttachedNumbers ( ) : void
containsLettersAndNumbers (String) : int
processURL (String) : void
removeStopWords (String[ ]) : void
SearchEngine
query : var
allResults : var
webSearch : var
google.search.WebSearch ( ) : void
setSearchCompleteCallback ( ) : void
search (query) : void
execute (searchquery) : void

Figure3.3: IRIS: Class Diagram

CHAPTER 4

IMPLEMENTATION

This chapter discusses the various functions and explains the implementation details about the system.

4.1 LANGUAGE ANALYZER

The language analyzer module pre-processes the query, which includes passing it through the following components: tokenizer, language identifier, stop word eliminator, stemmer, query transliteration and dictionary analysis.

4.1.1 Tokenizer

Tokenizing essentially deals with breaking a stream of text up into words, symbols, or other meaningful elements called tokens. The list of tokens then becomes input for further processing.

Input: Query string.

Program Flow:

AnalyzerDriver.java calls the tokenizer( ) function in the Tokenizer class in order to convert the query string into an array list of tokens.

Output: ArrayList with tokenized words

4.1.2 Language Identifier

This module uses the unicode values of the characters in the string in order to determine language of the script.

Input: ArrayList of query tokens

Program Flow:

AnalyzerDriver.java has language identification of tokens incorporated into its main( ) function. This involves checking the unicode values of the first characters of the tokens and comparing it with the unicode value range of the languages in question, namely, Tamil and Hindi.

Output: Separate ArrayLists with the tokens of different scripts.

4.1.3 Stopword Eliminator

This module reads the stop words from a text file that contains a list of stop words in Hindi, Tamil and English. It reads them into an ArrayList and then removes these stop words from the ArrayList that contains the tokenized words of the query.

Input: Files containing Hindi, Tamil and English stop words and ArrayList of query tokens.

Program Flow:

Tokenizer.java contains the function removeStopWords( ) which removes the words in the stop words ArrayList from the query token ArrayList.

Output: ArrayList with query tokens after stop word elimination.

4.1.4 Stemmer

This module implements stemming, which is the process of reducing inflected words to their word stem, base or root form.

Input: ArrayList with query tokens

Program Flow:

The Porter Stemmer algorithm is used for stemming the English tokens. This is done using a Stemmer object from Stemmer.java. The function call is made from the stemWords( ) function in Tokenizer.java. The stemming for the Hindi tokens is done using a rule-based program using the stemHindi( ) function. The stemming for the Tamil words is done using a Snowball stemmer implemented in python through a command line call.

Output: ArrayList of stemmed query tokens

4.1.5 Multiword Expression Recognizer

Multiword expressions are expressions which are made up of at least 2 words and which can be syntactically or semantically distinct in nature. The program for multiword expression recognition uses the Stanford Core NLP tool. Stanford CoreNLP provides a set of natural language analysis tools, which parses through each word of each sentence and labels sequences of words in a text which are the names of things, such company names, or location names. Hindi and Tamil multi-words are recognized from a compiled list of such words.

Input: Query string

Program Flow:

This is done in the AnalyzerDriver.java file in the MWE_NER( ) method, where the tags of each word are identified using the sent.nerTags( ) function, then grouped together to determine the multiwords, which are then written onto a separate file for phrasal search.

Output: Text file containing all multiword expressions in the query separated by newlines.

4.1.6 Named Entity Recognizer

Named-entity recognition is a subtask of information extraction that seeks to locate and classify elements in text into pre-defined categories such as the names of persons, organizations, etc. The program for named entity recognition also uses the Stanford Core NLP tool, which parses through each word of each sentence and labels sequences of words in a text which are the names of things, such as company names, or location names.

Input: Query string

Program Flow:

This is done in the AnalyzerDriver.java file in the MWE_NER( ) method, where the tags of each word are identified, then grouped together to determine the named entities, which are then written onto a separate file for phrasal search.

The Compressed Word Format algorithm is used on the list of Indian names through the CWF class in CWF.java. The Levenshtein distance calculation is done using the LevDistance( ) method in LevDist.java for all words that were not recognized by Stanford Core NLP.

Output: Text file containing all named entities in the query separated by newlines.

4.1.7 Query Transliteration

This module involves replacing Tamil scripts in the query with transliterated Tamil text and Hindi scripts with transliterated Hindi text.

Input: Processed query string

Program flow:

Four different types of Unicode transliteration have been implemented in this module : Hindi to English, English to Hindi, Tamil to English, English to Tamil. These are implemented through the EnglishToHindi() method, the HindiToEnglish() method, the replaceTamilCharacters() method and the replaceEnglishCharacters() method in Transliterator.java, which map the unicode characters to their respective transliterated characters.

Output: Transliterated query string

4.1.8 Dictionary Analysis

This step includes translation of the query, which is the communication of the meaning of a source-language text by means of an equivalent target-language text.

Input: Processed query string

Program flow:

The TranslateTamil( ) and TranslateHindi( ) methods in the Translate.java file are used for translation of Hindi and Tamil words in the query. This is done using a HashMap generated from exhaustive dictionary text files. A HashMap is used in order to optimize lookup time, and fetch results quickly.

Output: Translated query string

4.2 SEARCH ENGINE

Once the query pre-processing is done, the next step is to make the API call to the Google search engine, retrieve the results, rank them and display them on the IRIS user interface.

4.2.1 User Interface Creation

Process:

The user interface for the search engine is written in PHP and HTML, with a script that makes an API call to Google's Web Search. This is seen in viewresults.php.

4.2.2 API Callback

Input: Query string

Process:

The initial step in this procedure is to install a WAMP server on the system, so as to run a server on the system itself and test the scripts. Once it is up and running, the user interface can be viewed on any web browser. The script makes a call to the WebSearch( ) method using the API key provided for each of the possible list of queries obtained from the preprocessing stage. At the end of each search query execution, the webSearchComplete( ) method is called which obtains the results and a snippet of its content and prints it in the required format.

Output: Unranked results in results objects.

4.3 RANKING USING QUERY INTENTION

4.3.1 Query Classification

Input : Query string

Process:

When the user enters a query, the model classifies it into Informational or Navigational or Transactional. The classification problem is extended to identify the language of output (English, Tamil, Hindi) based on the language of queries existing in the dataset.

The dataset is built by aggregating query logs and by incorporating additional queries in order to have queries in mixed script. Annotation is done manually by analyzing user queries. Weka, a data mining tool is used to perform classification. After loading the training dataset, StringToWordVector filter is used as it converts string attributes into a set of attributes representing word occurrence (depending on the tokenizer) information from the text contained in the strings. This is followed by choosing classifiers Naive Bayes and J48. The next step is to load the test data that has a set of web queries. The final class label is determined by the classifier based on the training data.

Output: Class label [Informational / Transactional / Navigational] and label is Language of output [Tamil, English, Hindi].

4.3.2 Heuristical Analysis

Input: Query string

Process:

To predict user intention, python packages such as Lit(Language Identification and Transliteration tool for indic scripts) and guess_language is imported. The language of input is predicted using the above two packages. Conditions used are a)output language based on maximum number of words in a particular language script b) output language based on maximum number of words in transliterated text (Transliterated tamil/ Transliterated hindi) c) output language containing words that belong to a prediction list. A list containing words that directly indicate output language is maintained as part of the heuristics. Words like *meaning, artham, arth, dictionary, in hindi, in tamil* are stored in this list. When the query words match words in the list, language of output is therefore identified.

Output : Language of output

The next chapter explains the test cases and the results obtained from the system.

CHAPTER 5

RESULTS AND TEST CASES

This chapter includes the results obtained, test cases considered and the output obtained at each stage.

5.1 TEST CASES

Testing is done module wise with the intermediate results shown.

Language Analyzer testing –Tokenizer testing, Stemmer and Stopword Elimination testing, Multiword Expression Recognition testing, Named Entity Recognition testing, Query Processing testing, Dictionary Analysis testing.

Query Intention testing –Testing the accuracy of query intention prediction.

Search Engine testing –Result fetch test, Ranking relevancy test .

5.1.1 Test Case 1

Input: meaning of ज्योति in வாழ்க்கை

Expected output: Meaning of light in life - English results)

Intermediate Outputs:

Tokenizer:{meaning,of,“ज्योति″,"in",“வாழ்க்கை”}

Language Identifier:

Hindi = {“ज्योति”}

Tamil = {“வாழ்க்கை” }

English = {meaning,of,in}

Stopword Eliminator: {meaning,“ज्योति”,}

Stemmer: {mean,“ज्योति”,“வாழ்க்கை”}

Multiword Expression Recog: { }

Named Entity Recog: { }

Query Transliteration: meaning of jyoti in vazhkkai

Dictionary Analysis: meaning of light in life

Query Intention: Language of Output: English

Google API results for query list:

- Meaning of Jyoti - indian baby names
- Jyoti - Name Meaning - What does my name mean
- Light - Definition and Meaning, Bible Dictionary
- Urban Dictionary: light of my life
- ज्योति Meaning in English
- ज्योति - Meaning in English - Shabdkosh

Re-ranked Output:

- Light - Definition and Meaning, Bible Dictionary

- Urban Dictionary: light of my life
- Meaning of Jyoti - indian baby names
- Jyoti - Name Meaning - What does my name mean ...

- Meaning in English-ज्योति - Shabdkosh

- Meaning in English- ज्योति मय करन

Google results for the original query:

- ज्योति - Meaning in English - Shabdkosh

- ज्योति मय करना - Meaning in English

- ज्योति पुंज - Meaning in English - Shabdkosh

- Timeline Photos - Heaven’s Voice Ministry | Facebook
- Baba Murlis - Facebook

5.1.2 Test Case 2

Input: तूफान breaking news

Expected output: Meaning-related English Results.

Intermediate outputs:

Tokenizer: {“तूफान”, breaking,news}

Language Identifier:

Hindi = {“तूफान”}

Tamil = { }

English = {breaking,news}

Stopword Eliminator: { }

Stemmer:{तूफान,break,news}

Multiword Expression Recog: { breaking news }

Named Entity Recog: { }

Query Transliteration: toofan breaking news

Dictionary Analysis: cyclone breaking news

Query Intention: Language of Output: English

Google API results for query list:

- Toofan express की ताज़ा ख़बर, ब्रेकिंग न्यूज़ in

- Toofan News: Find Latest News on Toofan - NDTV.COM
- Toofan - Latest News on Toofan | Read Breaking News on
- Toofan News: Find Latest News on Toofan - NDTV.COM
- Breaking news on cyclones - breakingnews.com
- Odisha cyclone - Latest News on Odisha cyclone | Read ...

Re-ranked Output:

- Breaking news on cyclones - breakingnews.com
- Odisha cyclone - Latest News on Odisha cyclone | Read ...

- Toofan express की ताज़ा ख़बर, ब्रेकिंग न्यूज़ in

- Toofan News: Find Latest News on Toofan - NDTV.COM
- Toofan - Latest News on Toofan | Read Breaking News on
- Toofan News: Find Latest News on Toofan - NDTV.COM

Google results for the original query:

- Toofan express की ताज़ा ख़बर, ब्रेकिंग न्यूज़
- Toofan News: Find Latest News on Toofan - NDTV.COM
- Toofan express news in Hindi, Toofan express की ताज़ा ...
- Toofan - Latest News on Toofan | Read Breaking News on ...
- VTNews India : Vaicharik Toofan News, Hindi News, Latest

5.1.3 Test Case 3

Input: வசந்த முல்லை lyrics

Expected output: Transliterated Tamil Results

Intermediate outputs:

Tokenizer:{"வசந்த", "முல்லை", lyrics}

Language Identifier:

Hindi = { }

Tamil = {வசந்த", "முல்லை" }

English = {lyrics}

Stopword Eliminator: { }

Stemmer: {"வசந்த", "முல்லை", lyrics}

Multiword Expression Recog: { }

Named Entity Recog: { }

Query Transliteration: vasantha mullai lyrics

Dictionary Analysis: spring landscape lyrics

Query Intention: Language of Output: Transliterated Tamil/ Tamil Script

Google API results for query list:

- வசந்த முல்லை போலே வந்து ஞ்
- வசந்த முல்லை போலே ... – YouTube
- வசந்த முல்லை போலே – Thamizhisai
- Pokkiri - Vasantha Mullai Lyrics - LyricsSpot
- Tamil Songs Lyrics: Vasantha Mullai Polae Vanthu ...
- Tamil Songs Lyrics: Vasantha Mullai - Pokkiri TamilMovie
- Danil Novikov - Spring Landscape lyrics | Musixmatch
- The Saddest Landscape - Spring lyrics | Musixmatch
- Paris in the Spring-Dreaming of an Ethereal Landscape

Re-ranked Output:

- Pokkiri - Vasantha Mullai Lyrics - LyricsSpot
- Tamil Songs Lyrics: Vasantha Mullai Polae Vanthu
- Tamil Songs Lyrics: Vasantha Mullai - Pokkiri Tamil..
- வசந்த முல்லை போலே வந்து
- வசந்த முல்லை போலே – YouTube
- வசந்த முல்லை போலே – Thamizhisai

- Danil Novikov - Spring Landscape lyrics | Musixmatch
- The Saddest Landscape - Spring lyrics | Musixmatch
- Paris in the Spring - Dreaming of an..

Google results for the original query:

- வசந்த முல்லை போலே வந்து ஞ்
- வசந்த முல்லை போலே ... - YouTube
- வசந்த முல்லை போலே – Thamizhisai

5.1.4 Test Case 4

Input: what is கடமை உணர்ச்சி

Expected output: Meaning-related English Results

Intermediate outputs:

Tokenizer: {what, is,“கடமை”, “உணர்ச்சி"}

Language Identifier:

Hindi= { }

Tamil={“கடமை”,“உணர்ச்சி”}

English = {what, is}

Stopword Eliminator: {what,“கடமை”, “உணர்ச்சி”}

Stemmer: {what,“கடமை”, “உணர்ச்சி”}

Multiword Expression Recog: { }

Named Entity Recog: { }

Query Transliteration: what kadamai unarchhi

Dictionary Analysis: what duty sense

Query Intention: Language of Output: English

Google API results for query list:

- இது கடமை உணர்ச்சி - 2Tamil.com
- விதைகள்: கடமை உணர்ச்சி
- sense of duty - Dictionary Definition : Vocabulary.com
- Duty - Wikipedia, the free encyclopedia
- Un Kadamai Unarchikku Oru Alave Illayada Facebook
- KaatuPoochi on Twitter: "Unga kadamai unarchi.."

Re-ranked Output:

- sense of duty - Dictionary Definition : Vocabulary.com
- Duty - Wikipedia, the free encyclopedia
- இது கடமை உணர்ச்சி
- விதைகள்: கடமை உணர்ச்சி
- Un Kadamai Unarchikku Oru Alave Illayada Facebook
- KaatuPoochi on Twitter: "Unga kadamai unarchi.."

Google results for the original query:

- இது கடமை உணர்ச்சி – 2Tamil.com
- விதைகள்: கடமை உணர்ச்சி
- என்ன கடமை உணர்ச்சி ... – GenFB
- Puradsifm - என்ன ஒரு கடமை உணர்ச்சி ...
- ஜெயராம் on Twitter: "@idiyaappam உங்க ...

5.2 EVALUATION

For evaluation of the IRIS project, 20 multilingual/multiscript queries that challenge existing search engines are collected from students through a survey. The results retrieved are crosschecked with the students expectations and the performance of the IRIS search engine is compared against that of an existing Information Retrieval system like Google using the precision evalutation metric.

The Query Intention classifiers are evaluated using their True Positive Rate (TPR), False Positive Rate (FPR) along with recall, precision and F-measure.

The evaluation criteria for the Query Intention classifier use the following values; True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN), Positive (P), Negative (N) :

- The true positive rate (TPR) is given by

$$\mathrm{TPR} = \mathrm{TP}/P = \mathrm{TP}/(\mathrm{TP}+\mathrm{FN}) \tag{5.1}$$

- The false positive rate (FPR) is given by

$$\mathrm{FPR} = \mathrm{FP}/N = \mathrm{FP}/(\mathrm{FP}+\mathrm{TN}) \tag{5.2}$$

- Precision or positive predictive value (PPV)

$$\mathrm{PPV} = \mathrm{TP}/(\mathrm{TP}+\mathrm{FP}) \tag{5.3}$$

- F1 score is the harmonic mean of precision and sensitivity

$$\mathrm{F1} = 2\mathrm{TP}/(2\mathrm{TP}+\mathrm{FP}+\mathrm{FN}) \tag{5.4}$$

- Recall/ Sensitivity or true positive rate (TPR) is given by

$$\mathrm{TPR} = \mathrm{TP}/P = \mathrm{TP}/(\mathrm{TP}+\mathrm{FN}) \tag{5.5}$$

Table5.1: Detailed Accuracy of Query Intention Classifier(Naive Bayes)

Class	TP Rate	FP Rate	Precision	Recall	F-Measure
English	0.891	0.323	0.907	0.891	0.899
Hindi	0.696	0.054	0.604	0.696	0.646
Tamil	0.66	0.041	0.673	0.66	0.667
Average	0.749	0.139	0.728	0.749	0.737

Table5.2: Detailed Accuracy of Query Intention Classifier(J48)

Class	TP Rate	FP Rate	Precision	Recall	F-Measure
English	0.976	0.604	0.851	0.976	0.91
Hindi	0.326	0.01	0.78	0.326	0.462
Tamil	0.46	0.01	0.852	0.46	0.597
Average	0.587	0.202	0.593	0.449	0.383

Table 5.1 and 5.2 give the detailed accuracy of J48 and Naive Bayes classification of *language of output* based on recall, precision, TP Rate, FP Rate and F-Measure. The Naive Bayes Classifier classifies queries into Hindi with an accuracy of 0.64, Tamil with an accuracy of 0.66 while J48 classifies Hindi with an accuracy of 0.46 and Tamil with an accuracy of 0.59. J48 Naive Bayes classify English with an impressive accuracy in language prediction.

The evaluation criteria for search performance are as follows:

- **Precision** is the fraction of the documents retrieved that are relevant to the user's information need.

$$precision = (relevant \cap retrieved)/retrieved \tag{5.6}$$

- Average precision computes the finite sum of p(r) values over every position in the ranked sequence of documents:

$$\mathrm{AveP} = \frac{\sum_{k=1}^{n}(P(k) \times \mathrm{rel}(k))}{\text{number of relevant documents}} \tag{5.7}$$

  where

$$\mathrm{rel}(k) \tag{5.8}$$

  is an indicator function equaling 1 if the item at rank k is a relevant document, zero otherwise.

- Mean average precision for a set of queries is the mean of the average precision scores for each query.

$$\mathrm{MAP} = \frac{\sum_{q=1}^{Q} \mathrm{AveP(q)}}{Q} \tag{5.9}$$

  where Q is the number of queries.

Table5.3: Precision Comparison between Existing IR and IRIS

Query	Precision in Google IR	Precision in IRIS
Q-1	0.3912	0.4145
Q-2	0.2172	3071
Q-3	0.4054	0.4706
Q-4	0.4333	0.5487
Q-5	0.4906	0.5059
Q-6	0.7812	0.7968
Q-7	0.513	0.6957
Q-8	0.4738	0.5338
Q-9	0.4023	0.5116
Q-10	0.5116	0.5999
Q-11	0.6913	0.7934
Q-12	0.7032	0.7952
Q-13	0.2235	0.3375
Q-14	0.2753	0.3071
Q-15	0.5006	0.6167
Q-16	0.4913	0.5333
Q-17	0.6158	0.6189
Q-18	0.5632	0.6739
Q-19	0.3827	0.4844
Q-20	0.2101	0.2599
Average	0.4638	0.5402

Table5.4: Calculated Average Precision in Google IR and IRIS

Query	AP in Google IR	AP in IRIS
Q-1	0.5024	0.6201
Q-2	0.6221	0.7671
Q-3	0.6112	0.8162
Q-4	0.5932	0.6667
Q-5	0.685	0.7517
Q-6	0.6186	0.6938
Q-7	0.6234	0.6721
Q-8	0.5997	0.6303
Q-9	0.5811	0.6224
Q-10	0.6531	0.6563

In our evaluation of the IRIS search engine, we compared the attributes of the Google IR system with that of IRIS and we have gotten improved results in terms of effective retrieval. The calculated precision values obtained from 20 Queries on Google IR and the IRIS system are shown in Table 5.3. Table 5.4 contains calculated average precision values obtained from 10 queries applied on Google IR and IRIS systems.

Table5.5: Mean Average Precision in Google IR and IRIS

Queries	MAP in Google IR	MAP in IRIS
10	0.60898	0.68967

Mean average precision values for the systems are calculated from the average precision values, as tabulated in table 5.5. These experimental results show that the effective retrieval of the IRIS search engine has improved by 13.2% over the existing IR system.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this project, a wrapper for multilingual and multiscript search has been developed which can be used with existing search engines. The problems in existing search engines that cannot efficiently cater to multilingual queries has been discussed.

A multilingual, multiscript search engine with language specific query pre-processing and ranking based on query intention has been presented. The IRIS language analyzer module contains NLP methods that pre-process Tamil and Hindi queries providing variations of the query for fetching better search results. Query Intention classification and heuristics have been explored to predict the user's intent behind a query in terms of language of output. Our experimental results have shown that the effective retrieval and performance of the IRIS system has improved by 13.2% over the existing IR system.

It is our understanding that ranking based on query intent could significantly outperform other ranking mechanisms if we apply existing ranking features along with our proposed heuristics and classification methods. Our results have proved that the IRIS search engine is a better option for multilingual queries as the results are more relevant and focused. There is also strong evidence of more heuristic characteristics having a strong correlation with region and language intents. These issues can be investigated in the future. An expansion of the IRIS search engine to more languages can also be done.

REFERENCES

1. King & Abney 2013, ’Labeling the Languages of Words in Mixed-Language Documents using Weakly Supervised Methods’, Proceedings of NAACL-HLT, Atlanta, pp. 1110 - 1119.
2. Janarthanam, Sethuramalingam & Nallasamy 2008, ’Named Entity Transliteration for Cross-Language Information Retrieval using Compressed Word Format Mapping algorithm’, Conference: Proceeding of the 2nd ACM workshop on Improving Non English Web Searching, Napa Valley, vol. 56, no.7, pp. 33-38.
3. Morwal Jahan & Chopra 2012, ’Named Entity Recognition using Hidden Markov Model (HMM)’, International Journal on Natural Language Computing (IJNLC) vol. 1, no.23, pp. 46-53.
4. C.J. Van Rijsbergen, S.E. Robertson, M.F. Porter 1980, ’New models in probabilistic information retrieval’, British Library R D Report, Vol 14 no. 3 pp 130-137.
5. Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J Bethard David McClosky 2001, ’The Stanford CoreNLP Natural Language Processing Toolkit’, pp. 22-31.
6. Vishal Gupta 2014, ’Hindi Rule Based Stemmer for Nouns’, International Journal of Advanced Research in Computer Science and Software Engineering, 2014, vol. 4, no. 1, pp. 63-67.
7. C.J. Van Rijsbergen, S.E. Robertson, M.F. Porter 2001, ’Snowball: A language for stemming algorithms’, Available in `http://snowball.tartarus.org/algorithms/porter/stemmer.html` [July 2001].
8. M.Thangarasu, Dr.R.Manavalan 2013, ’Stemming Algorithms for Indian Languages’, International Journal of Computer Trends and Technology (IJCTT), Vol. 4, no. 8, pp. 113-124.
9. Anand Arokia Raj Harikrishna Maganti 2014 ’Transliteration based Search Engine for Multilingual Information Access’, Proceedings of CLIAWS3, Third International Cross Lingual Information Access Workshop, Boulder, Colorado, pp. 12-20.
10. Parth Gupta, Kalika Bali, Rafael E. Banchs, Monojit Choudhury Paolo Rosso 2014, ’Query Expansion for Multi-script Information Retrieval: FIRE Shared Task on Transliterated Search’, Proceedings of the 37th International ACM SIGIR Conference on Research Development in Information Retrieval, Sydney, pp. 677-686.

11. Lisa Ballesteros, W. Bruce Croft 1997, 'Phrasal Translation and Query Expansion Techniques for Cross-Language Information Retrieval', Proceedings of the 20th annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Philadelphia, pp. 84-91.

12. Atreya, Arjun Chaudhari, Swapnil Bhattacharyya, Pushpak and Ramakrishnan, Ganesh 2012, 'Building Multilingual Search Index using open source framework', Proceedings of the 3rd Workshop on South and Southeast Asian Natural Language Processing (SANLP), Mumbai, pp. 201-210.

13. Yi Chang, Ruiqiang Zhang, Srihari Reddy, Yan Liu 2011, 'Detecting Multilingual and Multi-Regional Query Intent in Web Search', Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, San Fransisco, vol. 56, no.8, pp. 1134-1139.

14. Ahmed Y. Tawfik and Ahmed S. Kamel 2014, 'Query Language and Query Intent in Cross-Lingual Web Search', Proceedings of SIRIP, pp. 44-49.