Query Optimization

Project submitted in partial fulfillment of the requirements for the degree of

M. Tech

by

Amit Jena
(A115002)

Department of Computer Science & Engineering
International Institute of Informztion Technology-Bhubaneswar
Bhubaneswar-751003
Odisha, India

Query Optimization

Project submitted in partial fulfillment of the requirements for the degree of

M. Tech

by

Amit Jena

Under the esteemed guidance of

Dr. Rakesh Chandra Balabantaray

Department of Computer Science & Engineering
International Institute of Informztion Technology-Bhubaneswar
Bhubaneswar-751003
Odisha, India

dedicated to my parents...

Declaration

I certify that

1. The work contained in this thesis is original and has been done by myself under the general supervision of my supervisor.

2. The work has not been submitted to any other Institute for any degree or diploma.

3. I have followed the guidelines provided by the Institute in writing the thesis.

4. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

5. Whenever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.

6. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Place: IIIT-Bhubaneswar
Date: 22nd May 2017

AMIT JENA
Department of Computer Science and Engineering
IIIT-Bhubaneswar
Bhubaneswar, 751003, Odisha, INDIA

Certificate

*This is to certify that the dissertation entitled, **"Query Optimization"**, submitted by **AMIT JENA** to the International Institute of Information Technology (IIIT) Bhubaneswar for the partial fulfillment of award of the degree **M.Tech. in Computer Science and Engineering**, is a record of bonafide work carried out by him under my supervision and guidance.*

To the best of my (our) knowledge, the results embodied in this dissertation have not been submitted to any other University / Institute for the award of any other Degree or Diploma.

Dr. Rakesh Chandra Balabantaray
Department of Computer Science & Engineering

Examined By

(External Examiner)

Place: IIIT-Bhubaneswar
Date: 22nd May 2017

Acknowledgement

I would like to take this opportunity to thank my guide Dr. Rakesh Chandra Balabantaray for his support and guidance in the thesis work. Also I would like to thank each and every faculty member of IIIT Bhubaneswar for their continued support and blessings.

AMIT JENA

IIIT-Bhubaneswar
22nd May 2017

Abstract

In this study, we are addressing two aspects of the search query. First, we tried to address the problem of Hindi-English code mixed search query. Code-mixing is frequently observed in user-generated content on social media and also in web search engines, especially from multilingual users. We have considered Hindi-English mixed term queries written in common roman script. Second, we also studied the possibility of exploring the query sequence of users to automate the "Query Reformulation" process.

Web search users frequently modify their queries till they get relevant results. Previous research work has mainly concentrated on suggesting queries for users. Some research has also been carried out to identify the concept behind the queries and propose a query reformulation. The query once fired into our system, will undergo language identification to find out the language of each query term, normalize the query terms to remove ambiguities and then the normalized terms are converted to correctly spelled English words. Then the query is analyzed for inherent sequential continuity and final reformulation is done.

To test our system we have to build a Desktop Search Application. On the backend, the data on the computer system in indexed into the Apache Solr using Apache Tika and other dependency parsers. And the user can search his PC using Code-Mixed Hindi-English search query.

Keywords : Query Reformulation, Query Refinement, Query Sequence, Language Identification

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In today's world of rapid growth in technology we are having our personal computers stacked with huge quantities of data. The data being stored may be structured like databases or unstructured like media files. Many a times we fail to find a document even though we know that we have saved it on our PC. To add to the problem, the new file's types are also getting added at a rapid pace. The files being stored in our PC lacks any sort of order or rule. We are free to store any file with any given name at any place in our PC. The files may be self-created, downloaded from internet, fetched from other persons or might have come from any other possible source. The heterogeneity in source of origin of files, makes the metadata available for the files to be very different form each other. We might be having exactly same files with different file names residing in our PC. All these factors make the desktop search a very complex task. The approach need to be different from normal web search to deal with this inherent differences.

In the country like India, where we have more than 22 languages, to make the technology accessible to the masses, language barrier should be removed. Though the user queries in Indian language scripts is beyond the scope of this paper. Here we present a novel approach where the user can enter search queries in Hindi-English code-mixed form but the script will be roman i.e. the user query will basically be a combination of Hindi and English terms written in English. The query then will be processed to get correctly spelled English words. And finally, we will keep track of the user's query sequence to automate the query reformulation task.

At the back end, we indexed the data present on our PC using Apache Tika into Apache Solr. Once indexed into Solr, we can have a highly reliable search platform which will give us relevant documents. The Apache

Tika toolkit detects and extracts metadata and text from over a thousand different file types (such as PPT, XLS, and PDF). All of these file types can be parsed through a single interface, making Tika useful for search engine indexing, content analysis, translation, and much more. Apache Solr is an open source search platform built upon a Java library called Lucene. Solr is a popular search platform for enterprise search because it can be used to index and search documents and email attachments.

Google now processes over 40,000 search queries every second on average and translates to over 3.5 billion searches per day and 1.2 trillion searches per year worldwide. Out of that, approximately 30% are reformulation to the previously entered query. For example, a user may search for “capital of India” and then gives the query “population of the country”. Then the existing search engines treat each query as independent queries. The search result in Google for the second query gives us a general population statistics of different countries of the world. But the user intention was to search for the population of India.

The search engine side of query reformulation is in place for quite some time now. Using our system we can do searching like a man-to-man conversation by asking subsequent questions in a way we could never do with regular search engines. We try to imitate the way humans converse, where there is a link between the present and previous query. Ongoing query refinement techniques are helpful, but text-based conversational search is a step ahead. It’s not about only padding terms to the original query but also about parsing the language. We parse the query so as the search engine understands, really comprehends, what we intend to search.

The techniques that we focus on generate new queries by substituting or adding words or phrases to the original query. Two queries that are different from each other by only one phrase are selected and the corresponding pairs of phrases are recorded as substitution candidates, which are used to generate substitutions for new queries. This study, however, did not evaluate the retrieval effectiveness of the new queries with substitutions.

Our system has three modules :

1. Handling of Hindi-English mixed query written in roman script.
2. Query reformulation using query sequence.
3. Desktop search Application using Apache Tika and Apache Solr

1.1 Query Reformulation

Web search process involves the user entering a query, learning from the results, and reformulating the queries till the information need is satisfied. When a query is entered into the search system, it makes an attempt to return the closest possible results to that query. Many a times it has been observed that the user reformulates the query with a hope to retrieve better results. Generally users tend to ignore the low ranked results to the query. They scroll through the high ranked results and chose to modify the query than going down to the successive results to search for the information (Jansen and Spink, 2006). Research into query reformulation and user behavior have shown that 37% of search queries are reformulations to previous queries (Jansen e al., 2007) and that 52% of users reformulate their queries (Jansen et al., 2005) to get a better result.

Generally, there is an inherent continuity in the user queries. The research work till now has been to identify and understand the query reformulation behavior. The intention behind this was to refine the query auto suggestions. Also, if we are able to predict the query reformulation with high accuracy then we will be able to evaluate the user satisfaction with query results. This will cross-validate our traditional approach to evaluate user satisfaction based on click-through information.

1.2 Handling of Mixed Language query terms

Bilingual speakers (users having knowledge of both English and Hindi) widely use code-mixing and code-switching in their regular use of language on social media and search platforms. This tendency is highly observed in search query logs too. Users tend to enter the query in a casual manner. Code-Mixing can be formally defined as an embedding of phrases, words or morphemes of one language into a query of another language. And code-switching refers to the co-occurrence of words belonging to two different grammatical systems (Gumperz., 1982). We are referring to both code-mixing and code-switching by Code-Mixing.

Hindi-English speakers generate a high volume of the code-mixed query. Vyas et al. (2014) found that it is highly complex to analyze such query stems because they do not generally follow any formal grammar, have spelling variations and typos, lack of annotated data, and query text in inherently conversational in nature.

1.3 Data Distribution

language	Sentences
English	141 (16.43%)
Hindi	111 (12.94%)
Code-Mixed	606 (70.63 %)
Total	858

Table 1.1: Data distribution at sentence level.

language	All Sentences	Only CM Sentences
Hindi	6318 (57.05%)	5352 (63.34%)
English	3015 (27.22%)	1886 (22.32%)
Rest	1742 (15.73 %)	1212 (14.34%)
Total	11075	8450

Table 1.2: Data distribution at token level.

Hindi-English bilingual speakers produce huge amounts of Code-Mixed text. The complexity in analyzing the Code-Mixed arises from non-adherence to a formal grammar, spelling variations, lack of annotated data, inherent conversational nature of the text and of course, code-mixing.

Bali et al. (2014) gathered data from Facebook generated by English-Hindi bilingual users which on analysis, showed a significant amount of code- mixing. Barman et al. (2014) investigated language identification at word level on Bengali-Hindi-English CSMT. They annotated a corpus with more than 180,000 tokens and achieved an accuracy of 95.76% using statistical models with monolingual dictionaries.

Solorio and Liu (2008) experimented with POS tagging for English-Spanish Code-Switched discourse by using pre-existing taggers for both languages and achieved an accuracy of 93.48%. However, the data used was manually transcribed and thus lacked the problems added by CSMT. Vyas et al. (2014) formalized the problem, reported challenges in processing Hindi-English CSMT and performed initial experiments on POS tagging. Their POS tagger accuracy fell by 14% to 65% without using gold language labels and normalization. Thus, language identification and normalization are critical for POS tagging (Vyas et al., 2014).

Jamatia et al. (2015) also built a POS tagger for Hindi-English CSMT using Random Forests on 2,583 utterances with gold language labels and achieved an accuracy of 79.8%. In the monolingual social media text context, Gimpel et al. (2011) built a POS tagger for English tweets and achieved an accuracy of 89.95% on 1,827 annotated tweets. Owoputi et al. (2013) further improved this POS tagger, increasing the accuracy to 93%.

1.4 Code-Mixed Dataset Examples

Sl. No.	**Example**	**Type**
1	try fr sm gov job jiske forms niklte h...	Code-Switching from English to Hindi
2	divya bharti temple marriage center ko donate karna	Code-Mixing (English word is used in Hindi utterance.)
3	tum kab ja rahe ho bcoz thr is no tckt avalble	Code-Switching from English to Hindi
4	you to aab gone	Code-Mixing (Hindi words are used in English utterance.)
5	thoda toh overacting banta hai na	Code-Mixing (English word is used in Hindi utterance.)
6	tumne koi movie dekhi	Code-Mixing (English word is used in Hindi utterance.)
7	wht r u doing, mein free hun..	Code-Switching from English to Hindi
8	puri beach pasand hai.!	NO Code-Mix
9	bharat ka pradhn mantri kaun	NO Code-Mix
10	aap kese Delhi jaoge	NO Code-Mix

Table 1.3: Code-Mixed Dataset Example

Chapter 2

Related Work

In our work we are having three modules: Handling of Code-Mixed Hindi-English search query, query sequence analysis, and Desktop Search Application. We will briefly explain the work done in all the three aspects stated above. Before going into the related works specific to our project, we will first see some interesting work done in the field of Query Optimization in general.

2.1 Query Optimization

Understanding query reformulation behavior and being able to accurately identify reformulation queries have several benefits. One of these benefits is learning from user behavior to better suggest automatic query refinements or query alterations. Another benefit is using query reformulation prediction to identify boundaries between search tasks and hence segmenting user activities into topically coherent units. Also, if we are able to accurately identify query reformulations, then we will be in a better position to evaluate the satisfaction of users with query results. For example, search satisfaction is typically evaluated using clickthrough information by assuming that if a user clicks on a result, and possibly dwells for a certain amount of time, then the user is satisfied. Identifying query reformulation can be very useful for finding cases where the users are not satisfied even after a click on a result that may have seemed relevant given its title and summary but then turned out to be not relevant to the user's information need.

Previous work on query reformulation has either focused on automatic query refinement by the search system, e.g. (Jones et al., 2006; Boldi et al., 2008) or on defining taxonomies for query reformulation strategies, e.g. (Lau and

Horvitz, 1999; Anick, 2003). Other work has proposed solutions for the query reformulation prediction problem or for the similar problem of task boundary identification (Radlinski and Joachims, 2005; Jones and Klinkner, 2008). These solutions have adopted the bag-of-words approach for representing queries and mostly used features of word overlap or character and word level edit distances. Take the queries "hotels in New York City" and "weather in New York City" as an example. The two queries are very likely to have been issued by a user who is planning to travel to New York City. The two queries have 5 words each, 4 of them are shared by the two queries. Hence, most of the solutions proposed in previous work for this problem will incorrectly assume that the second query is a reformulation of the first due to the high word overlap ratio and the small edit dis- tance. In this work, we propose a method that goes beyond the bag-of-words method by identifying the concepts underlying these queries. In the previous example, we would like our method to realize that in the first query, the user is searching for "hotels" while for in the second query, she is searching for the "weather" in New York City. Hence, despite similar in terms of shared terms, the two queries have different intents and are not reformulations of one another.

Boldi et al. (2008) introduced the concept of the query-flow graph where every query is represented by a node and edges connect queries if it is likely for users to move from one query to another. Mei et al. (2008) used random walks over a bipartite graph of queries and URLs to find query refinements. Query logs were used to suggest query refinements in (Baeza-Yates et al., 2005). Hierarchical agglomerative clustering was used to group similar queries that can be used as suggestions for one another. Other research has adopted methods based on query expansion (Mitra et al., 1998) or query substitution (Jones et al., 2006). This line of work is different from our work because it focuses on automatically generating query refinements while this work focuses on identifying cases of manual query reformulations.

2.2 Query Reformulation Strategies

Traditionally Query reformulation is done using three global methods :

1. assisting the user to reformulate the query using vocabulary tools
2. using a manual thesaurus
3. by automatic thesaurus generation

2.2.1 Using vocabulary techniques for Query reformulation

While a user is searching, we can take help of various vocabulary tools to guide the user. There can be many support techniques to show the user which search query is giving accurate result and which need some refinement. The user can be provided with the following list of their search :

1. omitted words form the query (stop words)
2. stemmed words
3. number of matches for each term and phrase
4. dynamic conversion of words to phrases

The search system may also suggest alternate queries by taking help from a thesaurus or some existing vocabulary. We can also provide the user with the inverted index list of words, which would help the user in finding appropriate word substitutions from the indexed terms.

2.2.2 Query expansion using a manual thesaurus

The manual query expansion technique works in close association with the user. The user provides reinforced information for the search query terms or phrases. The user may suggest some additional terms to update the query. Some web based search engines like Google and Bing uses auto suggestion in response to the entered query. The user gets a list of closely related queries for the entered query.

The most widely used query expansion technique uses global analysis of query using the thesaurus. For all the query terms entered for the query, the query should be reformulated by substituting the terms with their synonyms to match all similar results. This can be achieved by using a thesaurus. This idea when combined with the term weighting factor, can boost the search result significantly.

2.2.3 Dynamic generation of thesaurus

The manual formulation and maintenance of thesaurus is highly cost intensive. This can be substituted with dynamic generation of thesaurus by analyzing the existing collection of documents to make it cost effective. There are two main techniques to create a thesaurus dynamically:

- Finding out the co-occurrence of words: Words co-occurring in a document or paragraph are more likely to be semantically similar or synonymous. We just then use simple text analytics approaches to find the similar words.
- The second approach is to use a grammar oriented approach. Here we try to analyse the grammatical relations and how words are grammatically dependant on each other. For example, entities that are woven, stitched, ironed, hooded, collar, are more likely to be dress materials. Simply using word co-occurrence may be misled by the parser errors, but if we take the help of grammatical relations then it is more robust.

There are some other techniques used by researchers for Query reformulation:

2.2.4 Query Reformulation using Anchor text

Query reformulation techniques using the query logs is widely accepted technique to understand user intention behind the query and improving the result retrieval effectiveness. In the paper the author suggested that the readily available anchor text can be an effective alternative to the query log technique. They studied various existing query refinement and reformulation techniques based on stemming, query log based substitution, and query expansion using available TREC collections. The methods demonstrated by highlights that for standard collections, log based query reformulation techniques are more efficient. But research suggest query expansion to be much more effective form of query reformulation than mere word substitution. Their results showed that for the above discussed techniques using anchor text as substitute for query log does not give any significant improvement.

2.2.5 Query reformulation using concept based matching

Here the main idea was to do concept based term matching using a list of words. In their work, they showed that relying on surface level term similarity results into many false positives. There are cases where queries with similar topics but different intent are falsely identified as being reformulated. The researchers proposed a changed representation of web based search queries by identifying the semantics and user intention behind the query. This model showed significant improvement in query reformulation performance by using features of query semantics.

2.3 Code-Mixed Query terms

Liu and Solorio (2008) used existing taggers for both English and Spanish language over code-switched domain for POS tagging of English- Spanish log. They achieved an accuracy of 93.48%. But the data used was manually transcribed hence failed to address the inherent problems of code-mixed query. Vyas et al. (2014) formally studied the problem, reported challenges in processing Hindi-English code-mixed query. He performed initial experiments on POS tagging. Their study showed that identification of code-mixed term's language and normalization are critical for the POS tagging task.

2.4 Desktop Search Engines

The Desktop Search applications currently available, either integrated into operating systems or as standalone applications have very limited capabilities. They search using keywords from metadata or in some cases from the headings and sub-headings of the documents. The desktop is a bit complex to search because different applications format different type of documents differently. Also desktop files can be either structured like in a database or documents with embedded tags or unstructured like media files like images and audio files.

Desktop search engines use a crawler to crawl the documents on the local drive and index them. This step is similar to the web-crawler used in web search engines. The major solutions which were traditionally provided in this segment were by the industry leaders: Google, Microsoft and Yahoo.

The latest desktop search application by Yahoo is X1 Search. Previously it was Yahoo! Desktop Search (YDS), but now it has been discontinued. The users searching for YDS is now being redirected to X1 Search. X1 search promises to work on desktop as well as virtualization and capable of handling documents, audio, video and e-mails. It provides search results on the basis of Boolean, proximity and keyword searching. It covers more than 500 file types in their native format. But it fails to deliver results in the case of code-mixed query. In that case the query result was not at all satisfactory.

Google Desktop was an application with desktop search capabilities. It allowed text based search for documents, audio, video and emails. In September 2011, Google discontinued this search application giving the reason that the storage is now getting shifted from personal computers to cloud. This application used to show search results as top 6 most relevant fetches. Though it performed well with English queries but didn't do well when searched with Code-Mixed query. Google Desktop Search, which was discontinued in 2011, but many of you still use and love even though Google doesn't officially make it available for download, and isn't offering security fixes, patches, or updates for it. There are various installers floating about the web if you're still interested in trying it (or still have it installed), and it really is a great desktop search tool, but getting it to work with current OSes can be challenging. Still, if you have it and it works for you, enjoy it!

Windows Search allows us to perform instant search of our desktop. It allows to search for emails, documents and media files. Using this we can index any program's content then it will give us instant results when searched for. It also provided language packs to support local languages. But it didn't perform well with Code-Mixed query. The latest integrated Microsoft search platform is Cortana. It searches for desktop and web both.

The above applications include no metadata in their searching process, but just regular text-based index. All the above three applications have inbuilt integration with internet. When searched for a user query they also search the web for relevant results.

In the work published by L3S Research Center, University of Hanover they used the activity based metadata of the user. They showed that by exploiting the Web cache content we can achieve a better result. The main characteristic of their work was event triggered metadata generation. They indexed the data on the fly as and when there is any addition of files.

In another work by NTU, Singapore they proposed a visualization and evaluation technique for desktop search. There work did a comparative study using visualization techniques for various desktop search applications. They used various 2D and 3D graphical visualization techniques to show the query result tree view, map view, tile view and any more views to better understand the results.

Chapter 3

Proposed Work

In our work, we developed a desktop search engine which takes code-mixed Hindi-English search query from the user. The query then passes through the language identification module, where each token in the query is assigned a language label: Hindi, English or Rest. Then the query tokens are further processed to get normalized tokens in their respective script i.e. Hindi tokens are now written in Hindi script. These tokens are free from spelling and other possible syntactic errors.

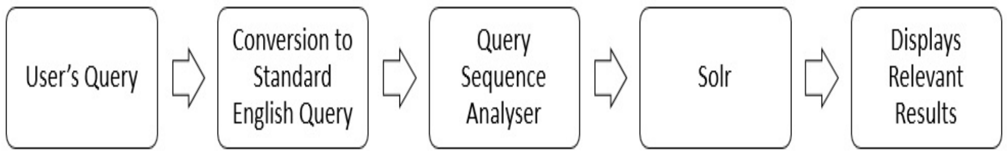
User's Query
Conversion to Standard English Query
Query Sequence Analyser
Solr
Displays Relevant Results

Figure 3.1: Overall Architecture of our System

Then the reformulated query is sent to the Apache Solr. In Solr we have already indexed the documents, files, media files using Apache Tika. We converted the video files to text by first extracting the audio content out of it and then indexed it in Solr. In our application we maintained a search window of size 10. All the user queries were analyzed by comparing with previous 10 queries to find any dependency between them. If found, the query is auto-reformulated before being sent to Solr for searching. But during our experiment we observed that the subsequent queries get reformulated within the window of 4.

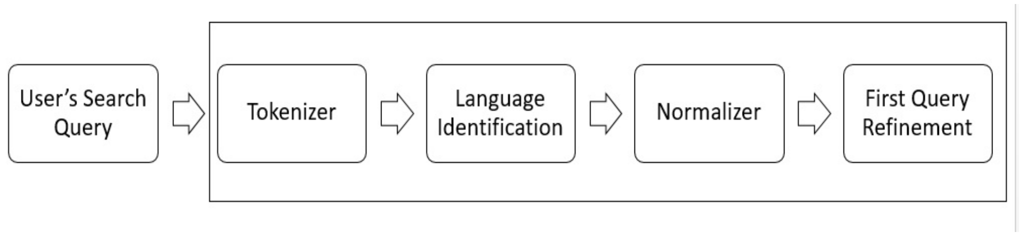

Figure 3.2: Conversion of Hindi-English mixed query to Standard English Query

3.1 Conversion of code-mixed search query to standard words

This stage has the Language Identification and Normalization layers. Our system takes the user entered query as input on which each module runs sequentially. The query may be bilingual code-mixed Hindi-English query or monolingual query, but the script is Roman. We used the NLTK package to tokenize original query into individual tokens. The Language Identification stage processes the tokens and tags then with a label from our predefined language labels. Depending upon the language label tag of the individual tokens, the Normalizer module runs the corresponding Hindi normalizer or the English normalizer.

3.1.1 Language Identification

Each token of the original query is given a tag out of the 'en', 'hi' and 'rest' to identify its language. Words that a Hindi-English bilingual speaker could identify as either Hindi or English were marked as 'hi' or 'en'. The label 'rest' was given to everything else.

The feature set comprised of

1. BNC: We took the normalized frequency of the 10, 00,000 English words in British National Corpus (BNC).
2. LEXNORM: It's a binary feature related to the presence of the word in the Han et al. (2011) dataset.
3. HINDI DICT: It's also a binary feature marking the presence of the word in a Hindi corpus of 30,823 transliterated words, released by Gupta (2012).

4. NGRAM: We have taken the n-grams of tokens up to n=3. We used CRF to perform the Language identification work by formulating the problem as a sequence labeling task.

3.1.2 Normalization

Words in Roman script that were given the tag 'hi' in the language identification stage were labeled with their standard form in the native script of Devanagari. Similarly, English words with language tag 'en' were corrected with their standard spelling. Words with language tag 'rest' were left unaltered.

After the language identification task is over, the noisy non-standard tokens, like Hindi words inconsistently written in many ways using the Roman script, in the entered query were transformed into standard words in the identified language. To address this, a normalization model which does language-specific transformations, resulting into correctly spelled words for a given token is built.

The Hindi normalizer, converted the words identified as Hindi to Devanagari script from roman script. We used GIZA++ and CRF to obtain Hindi words in Devanagari script from roman script. GIZA++ was used to obtain character alignment between non-standard Hindi words written in roman to normalized words format. Then CRF was employed for conversion from Roman script to Devanagari Script using learnt letter transformation. Then we used a standard Hindi dictionary to convert normalised Hindi word from Devanagari script to equivalent English words in roman script.

The English normalizer too worked on the same principle. Using edit distance and comparing with BNC (British National Corpus), we converted the words to Standard English words. These words were free from spelling errors and have no abbreviations.

3.2 Query Reformulation

First, we need to identify if there is a need of query. The reformulation strategy will take into consideration, the inherent continuity in the query sequence, the flow of concept and user intention behind the query. Multiple strategies exist to reformulate a query resulting in to different types of query reformulations:

1. *Generalization :* A generalization reformulation occurs when the second query is intended to seek more general information compared to the first query.
2. *Specification :* A specification reformulation occurs when the second query is intended to seek more specific information compared to the first query.
3. *Same Intent :* A same intent reformulation occurs when the second query is intended to express the same intent as the first query.

3.3 Data Preparation

Code-Mixed text was obtained from social media posts from the data shared for Subtask-1 of FIRE-2014 Shared Task on Transliterated Search. The existing annotation on the FIRE dataset was removed, posts were broken down into sentences and 858 of those sentences were randomly selected for manual annotation.

Table 1.1 and Table 1.2 show the distribution of the dataset at sentence and token level respectively. The language of 63.33% of the tokens in code-mixed sentences is Hindi. Based on the distribution, it is reasonable to assume that Hindi is the matrix language (Azuma, 1993; Myers-Scotton, 1997) in most of the code-mixed sentences.

3.3.1 Dataset Examples

1. hy... try fr sm gov job jiske forms niklte h...

   Gloss: Hey... try for some government job which forms give out...

   Reformulation: Hey... try for some government job which gives out forms...

2. To tum divya bharti mandir marriage kendra ko donate karna

   Gloss: So you divya bharti temple marriage center to donate do

   Reformulation: So you donate to divya bharti temple marriage center

The dataset is comprised of sentences similar to example 1 and 2. Example 1 shows code-switching as the language switches from English to Hindi where as example 2 shows code-mixing as some English words are embedded in a Hindi utterance. Spelling variations (sm-some, gov-government), ambiguous words (To-So in Hindi or To in English) and non-adherence to a formal grammar (out of place ellipsis -..., no or misplaced punctuation) are some of the challenges evident in analyzing the examples above.

3.3.2 Annotation

Annotation was done on the following four layers:

1. **Language Identification:** Every word was given a tag out of three 'en', 'hi' and 'rest' to mark its language. Words that a bilingual speaker could identify as belonging to either Hindi or English were marked as 'hi' or 'en'. The label 'rest' was given to symbols, emoti- cons, punctuation, named entities, acronyms, foreign words and words with sub-lexical code- mixing like chapattis (Gloss: chapatti - bread) which is a Hindi word (chapatti) following English morphology (plural marker -s).

2. **Normalization:** Words with language tag 'hi' in Roman script were labeled with their standard form in the native script of Hindi, Devanagari. Similarly, words with language tag 'en' were labeled with their standard spelling. Words with language tag 'rest' were kept as they are. This acted as testing data for our Normalization module.

This whole dataset was annotated by three Hindi- English bilingual speakers. Another annotator reviewed and cleaned it. To measure interannotator agreement, another annotator read the guidelines and annotated 25 sentences (334 tokens) from scratch. The inter-annotator agreement calculated using Cohen's κ (Cohen, 1960) came out to be 0.97 and 0.89 for language identification and parsing pipeline respectively.

Chapter 4

Conversion to Standard English Query

Our pipeline takes a raw utterance in Roman script as input on which each module runs sequentially. Twokenizer (Owoputi et al., 2013) which performs well on Hindi-English CSMT (Jamatia et al., 2015) was used to tokenize the utterance into words. The Language Identification module assigns each token a language label. Based on the language label assigned, the Normalizer runs the Hindi nor- malizer or the English/Rest normalizer.

Features	Accuracy
BNC	61.26
+LEX_NORM	71.43
+HINDI_DICT	77.50
+NGRAM	93.18

Table 4.1: Feature Ablation for Language Identifier

The functionality and performance of each module is described in greater detail in the following subsections.

4.1 language Identification

While language identification at the document level is a well-established task (McNamee, 2005), identifying language in social media posts has certain challenges associated to it. Spelling errors, phonetic typing, use of transliterated alphabets and abbreviations combined with code-mixing make this problem interesting. Similar to (Barman et al., 2014), we performed two experiments treating language iden- tification as a three class ('hi', 'en', 'rest') classification problem. The feature set comprised of - BNC : normalized frequency of the word in British National Corpus (BNC). LEXNORM : binary fea- ture indicating presence of the word in the lexical normalization dataset released by Han et al. (2011). HINDI DICT : binary feature indicating presence of the word in a dictionary of 30,823 transliterated Hindi words as released by Gupta (2012). NGRAM : word n-grams.

Using these features and introducing a context- window of n-words, we trained a linear SVM. In another experiment we modeled language identification as a sequence labeling task, where we employed CRF into usage. The idea behind this was that code-mixed text has some inherent structure which is largely dictated by the matrix language of the text. The latter approach using CRF had a greater accuracy, which validated our hypothesis.

4.2 Normalization

Once the language identification task was complete, there was a need to convert the noisy non-standard tokens (such as Hindi words inconsistently written in many ways using the Roman script) in the text into standard words. To fix this, a normalization module that performs language-specific transformations, yielding the correct spelling for a given word was built. Two language specific normalizers, one for Hindi and other for English/Rest, had two sub-normalizers each, as described below. Both sub-normalizers generated normalized candidates which were then ranked, as explained later in this subsection.

4.2.1 Noisy Channel Framework

A generative model was trained to produce noisy (unnormalized) tokens from a given normalized word. Using the model's confidence score and the probability of the normalized word in the background corpus, n-best normalizations were chosen. First, we obtained character alignments between noisy Hindi words in Roman script (H_r) to normalized Hindi words-format(H_w) using GIZA++ (Och and Ney, 2003) on 30,823 Hindi word pairs of the form ($H_w - H_r$) (Gupta et al., 2012). Next, a CRF classifier was trained over these alignments, enabling it to convert a character sequence from Roman to Devanagari using learnt letter transformations. Using this model, noisy H_r words were created for H_w words obtained from a dictionary of 1,17,789 Hindi words (Biemann etal., 2007). Finally, using the formula below, we computed the most probable H_w for a given H_r.

4.2.2 SILPA Spell Checker

This subnormalizer uses SILPA libindic spell-checker to compute the top 10 normalized words for a given input word.

The English normalizer too worked on the same principle. Using edit distance and comparing with BNC (British National Corpus), we converted the words to Standard English words. These words were free from spelling errors and have no abbreviations.

The candidates obtained from these two systems are ranked on the basis of the observed precision of the systems. The top-k candidates from each system are selected if they have a confidence score greater than an empirically observed λ. A similar approach was used for English text normalization, using the English normalization pairs from (Han et al., 2012) and (Liu et al., 2012) for the noisy channel framework, and Aspell as the spell-checker. Words with language tag 'rest' were left unprocessed. The accuracy for the Hindi Normalizer was 78.25%, and for the English Normalizer was 69.98%. The overall accuracy of this module is 74.48%; P@n (Precision@n) for n=3 is 77.51% and for n=5 is 81.76%.

Working of Silpa Spell Checker

The basic Silpa spell checker carries out the following processes:

- It scans the text and extracts the words contained in it.
- It then compares each word with a known list of correctly spelled words (i.e. a dictionary). This might contain just a list of words, or it might also contain additional information, such as hyphenation points or lexical and grammatical attributes.
- An additional step is a language-dependent algorithm for handling morphology. Even for a lightly inflected language like English, the spell-checker will need to consider different forms of the same word, such as plurals, verbal forms, contractions, and possessives. For many other languages, such as those featuring agglutination and more complex declension and conjugation, this part of the process is more complicated.

Chapter 5

Query Sequence Analysis

First, we need to identify if there is a need of query. The reformulation strategy will take into consideration, the inherent continuity in the query sequence, the flow of concept and user intention behind the query. Multiple strategies exist to reformulate a query resulting in to different types of query reformulations:

1. **Generalization:** A generalization reformulation occurs when the second query is intended to seek more general information compared to the first query.
2. **Specification:** A specification reformulation occurs when the second query is intended to seek more specific information compared to the first query.
3. **Same Intent:** A same intent reformulation occurs when the second query is intended to express the same intent as the first query.

We used the features given below to predict if query reformulation is needed:

1. Length of the queries and difference between them.
2. Count of out-of-vocabulary words in Q1, Q2 and the difference between them.
3. Count of common "exact match" concepts.
4. Count of common "approximate match" concepts.
5. Count of common "semantic match" concepts.
6. Count of concepts in query window and the difference between them.

7. Count of concepts in current query but not in previous query.
8. Count of concepts in previous query but not in current query.
9. Then do the evaluation by replacing the concepts with keywords.

We are maintaining a sliding window size of 10 for searched queries Q_i, $Q_i + 1$,..., $Q_i + 10$, i.e. we will keep track of 9 previous queries while analysing the current query. At present our system is able to handle the following types of queries:

- Pronoun replacement: If the subsequent query have any pronoun then it is substituted with the noun from the previous query.
- Name entity identification: If there is no pronoun in the subsequent query, but there is some named entity in the queries then it tries to combine the queries belonging to the same named entity.
- Continuity of concept: The query is segmented into semantic phrases depending upon the mutual information score. Whenever the point wise mutual information between two consecutive words drop below a predefined threshold, a segment break is inserted.

5.1 Named entity identification

For the search query we identify the named entities present in the query. With each entered query, we search for presence of named entities like company name, hotel, monuments and establish a link between them and the place (city or town), if present in the previous query. For e.g. if Q_1 is "Hotels at New Delhi" and Q_2 is "Hotel Taj", then the reformulated query becomes "Hotels Taj at New Delhi".

5.2 Continuity of concept

To capture concept similarity, we compute the concept similarity by measuring the semantic similarity between two queries in consideration. We used the following formula to compute similarity between two sequence of words, Q = q_1, q_2,... , q_I and S = s_1, s_2,..., s_J.

$$P(S|Q) = \prod_{i=1}^{I} \sum_{j=1}^{J} P(S_i|Q_j) P(Q_j|Q)$$

where $P(S|Q)$ is the unigram probability of word 'S' in the entire Query Q.

5.3 Work Flow of Query Sequence Analyzer

Once the user enters the query into our system, it first goes through *Hindi-English Code-Mixed Analyzer* module. Then the normalized reformulated query is sent to the *Query Sequence Analyzer* module. Since the query has already been reformulated, so the current query is free from spelling errors, and the entire query is in standard English i.e. all the tokens are standard English words. The query is also free from symbols and other unwanted terms. Here the current query is analyzed against the previously entered queries. We are maintaining a sliding window of size ten.

In this module, first it is needed to identify if query reformulation is possible for the current query. If yes, then the current query is searched for possible tokens and/or phrases which can be replaced. We use multiple techniques to do reformulation. One is replace the pronouns in the current query with nouns and named entities from previous query. Other approach is identify the phrases and substitute them with relevant phrases or tokens from previously entered queries. We also take care of continuity of concept, thus ensuring that the query fetches results relevant to what the user *intend to search* rather than based on *only query terms.*

Though we have maintained a sliding window of size ten, but we observed that majority of the queries were reformulated within a window size of three. There may arise situations where there will be multiple candidates for substitution, then the first candidate is chosen as replacement. The named entity identification is done by maintaining a corpus of 10,000 NER entities from Indian context.

Chapter 6

Desktop Search Application

6.1 Integration to Solr

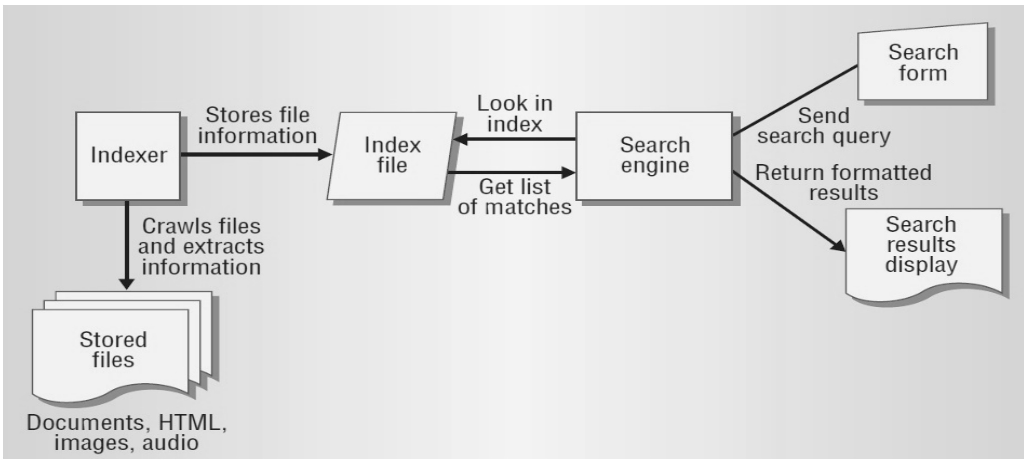

Figure 6.1: Using Apache Tika we indexed the documents and files into Apche Solr. The Media files like video and audio were converted to text before being indexed into Apache Solr.

The desktop search tool catalogs the contents of your computer in a way analogous to how search engines catalog the content of the Internet. The tool creates a repository of words and documents, much like the databases used by Google. With this little repository in place, you can then use your web browser (or some stand-alone application) to perform search-engine-esque searches on everything that has been cataloged. In some incarnations, the results show up at the top of search engine results.

The success or the failure of this idea hinges on 1) users' inability or unwillingness to organize their own data, 2) users keeping their data on a local "desktop" and 3) a homogenized and agreeable notion of what constitutes a "desktop". On first blush, my suspicion is that the desktop search idea will wither as the demographic who would likely be interested in desktop searching is already skilled at data organization. Furthermore, the savvy user probably does not store his or her most important documents on a local hard drive — for security and backup reasons. With the variety of desktop environments available, as well as the wider variety of filesystems available, it is hard to imagine that we could derive a generally agreeable and predictably static conception of "the desktop". Given these simple observations, we suspect that there is something else afoot in the push for desktop searching technologies.

Our application automatically indexes local folders and network paths in real time, so that the integrated desktop search quickly provides reliable and up-to-date search results from your local and network data. This is the central desktop search tool for all data on your PC. Thanks to the real time indexing, moments after a file in the system is altered, Solr Index is updated and they immediately become available to search.

We have integrated all the above modules into our desktop search application. Our application scans the local drive of the PC and using Apache Tika indexes all the text files into Apache Solr. The file formats include: .txt, .doc, .docx, .ppt, .pptx, .pdf. The user has the flexibility of searching the local drives using our desktop search application. The user can enter the search query as Hindi-English Code-Mixed query. The query once entered goes through the first module of Code-Mixed query reformulation. Then the query is sent to Apache Solr. We are exploiting the inherent search features of Solr. The rank algorithms, annotations, faceting of Solr is used. If the user enters subsequent queries then our second module comes into picture. It analyses the query sequence and tries to find out if any query reformulation is possible. If yes, the query is reformulated and then sent to Solr.

6.1.1 Apache Tika - a content analysis toolkit

The Apache Tika toolkit detects and extracts metadata and text from over a thousand different file types (such as PPT, XLS, and PDF). All of these file types can be parsed through a single interface, making Tika useful for search engine indexing, content analysis, translation, and much more. The main use case for which Tika was originally conceived is supporting a search engine to index the full-text contents of various kinds of digital documents. A search engine typically includes a crawler component that repeatedly traverses a set of documents and adds them to a search index. Since the search index normally only understands plain text, a parser is needed to extract the text contents from the documents.

What is Apache Tika?

- Apache Tika is a library that is used for document type detection and content extraction from various file formats.
- Internally, Tika uses various existing document parsers and document type detection techniques to detect and extract data.
- Using Tika, one can develop a universal type detector and content extractor to extract both structured text as well as metadata from different types of documents such as spreadsheets, text documents, images, PDFs and even multimedia input formats to a certain extent.
- Tika provides a single generic API for parsing different file formats. It uses 83 existing specialized parser libraries for each document type.
- All these parser libraries are encapsulated under a single interface called the Parser interface.

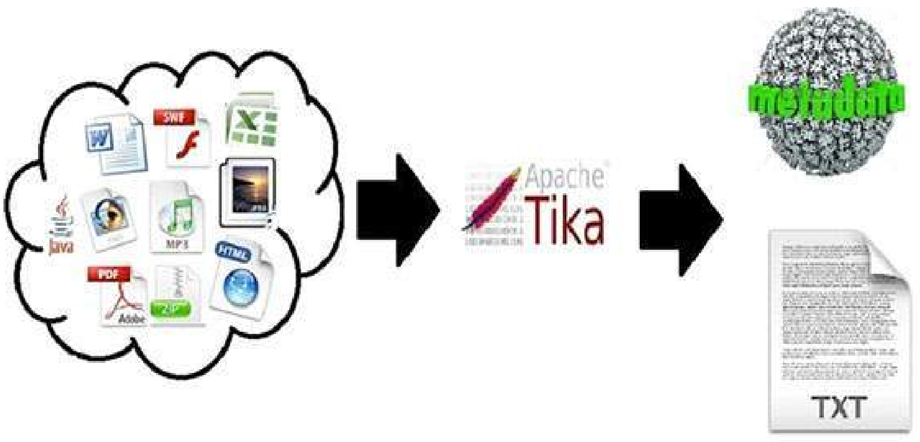

Figure 6.2: Multi-Format Files to Metadata using Tika

Why Tika?

According to filext.com, there are about 15k to 51k content types, and this number is growing day by day. Data is being stored in various formats such as text documents, excel spreadsheet, PDFs, images, and multimedia files, to name a few. Therefore, applications such as search engines and content management systems need additional support for easy extraction of data from these document types. Apache Tika serves this purpose by providing a generic API to detect and extract data from multiple file formats.

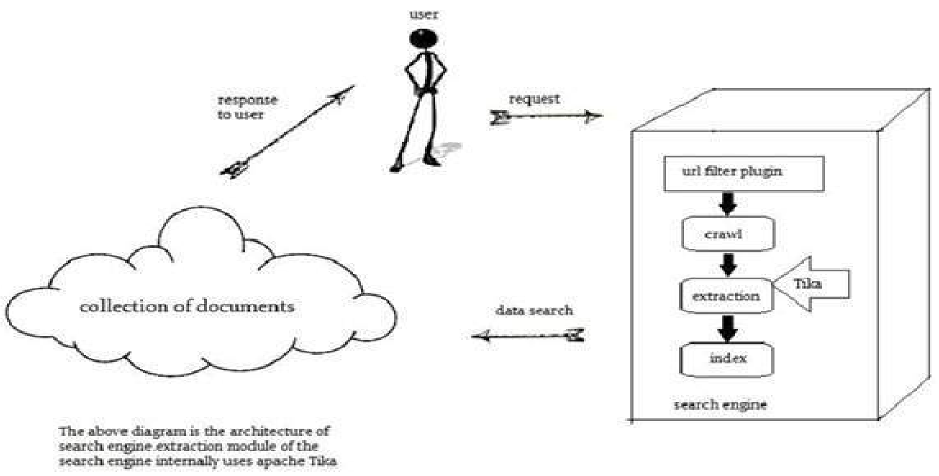
user
response to user
request
url filter plugin
crawl
extraction
Tika
index
search engine
collection of documents
data search
The above diagram is the architecture of search engine extraction module of the search engine internally uses apache Tika

Figure 6.3: Search Engine Extraction module using Tika

6.2 Apache Solr

Solr is highly reliable, scalable and fault tolerant, providing distributed indexing, replication and load-balanced querying, automated failover and recovery, centralized configuration and more. Solr powers the search and navigation features of many of the world's largest internet sites. Solr works with Hypertext Transfer Protocol (HTTP) Extensible Markup Language (XML). It offers application program interfaces (APIs) for Javascript Object Notation (JSON), Python, and Ruby. According to the Apache Lucene Project, Solr offers capabilities that have made it popular with administrators including:

- Indexing in near real time
- Automated index replication

- Server statistics logging
- Automated failover and recovery
- Rich document parsing and indexing
- Multiple search indexes
- User-extensible caching
- Design for high-volume traffic
- Scalability, flexibility and extensibility
- Advanced full-text searching
- Geospatial searching
- Load-balanced querying

Indexing in Apache Solr

In general, indexing is an arrangement of documents or (other entities) systematically. Indexing enables users to locate information in a document. Indexing collects, parses, and stores documents. Indexing is done to increase the speed and performance of a search query while finding a required document. In Apache Solr, we can index (add, delete, modify) various document formats such as xml, csv, pdf, etc. We can add data to Solr index in several ways.

Chapter 7

Results

The Hindi-English mixed query, was fired into our solr search system. The solr system had data indexed in it. We indexed pdfs, text files and doc files. The precision value for the original query and the auto-reformulated query was calculated. We randomly chose 2 users to use our system and fire Hindi-English mix query. They were asked to search for 100 queries. We had to manually ask the users to fire the query, since there is no query log available for Hindi-English mixed query for search engine.

The accuracy of language identification using CRF is 88%. And the accuracy of the Hindi and English normalizer was 76.5% and 81% respectively. The normalization module yields an overall accuracy of 78%.

The reformulated queries were tested by putting them into our solr based local search platform. In this, we have indexed 100 pdf and doc files related to India. The files indexed cover a wide range of details about India starting from history, geography, politics and culture. The precision for each of the code mixed query that was reformulated is given in the Table - 7.2. For precision calculation we have taken top 5 documents fetched for each query. Each document (pdf/doc/txt file) is given a relevancy score as follows:

- Document is relevant : 1
- Document is partial relevant : 0.5
- Document is not relevant : 0

We now present a few sample output from our Desktop Search Application. For each query our system shows maximum five relevant documents. The following table shows the precision value calculated for the files shown against each query.

Sl. No.	**Query**	**Path of Documents**	**P@5**
Q1	mchine learnin aur jankari retrval	/media/amit/Academics/BOOKS /COMPUTERS/AI & ANN Books/book-neuro-intro.pdf	0.8
New Query	machine learning and information retrieval	/media/amit/Academics/BOOKS /COMPUTERS/Information Retrieval/[Bing_Liu]_Web_Data _Mining_Exploring_Hyperlinks, _(BookFi).pdf	0.9
		/media/amit/Project/Machine Learning Books /181115.pdf	0.7
		/media/amit/Project/Machine Learning Books /RW.doc	0.75
		/media/amit/Project/Machine Learning Books/ ciml-v0_9-all.ppt	0.85
Q2	usmein intellgnt systms	/media/amit/Project/Machine Learning Books/ IntroMLBook.pdf	1
New Query	machine intelligent systems	/media/amit/Project/Machine Learning Books/neuronalenetze-en-zeta2-2col-dkrieselcom.docx	0.9
		/media/amit/Project/Machine Learning Books/whole.pdf	0.8
		/media/amit/Project/Machine Learning Books/book.pdf	1
		/media/amit/Project/Machine Learning Books/book_draft.pdf	0.9
Q3	feture neekalna	/media/amit/Project/Machine Learning Books	0.8

Table 7.1: Sample result of Desktop Search Application

7.1 Precision Values

In our Desktop Search Application, we have taken 10 files of different file formats. To generalize our work and test it against larger domain, we have tested the Original queries and reformulated queries by firing them manually to Google. The precision values given in the following table are calculated from the results and links given by Google search Engine for our queries.

Query No.	**Original Query**	**P@3**	**P@5**
Q1	bharat ki rajdhani	0	0
Q2	capital of the country	0.5	0.35
Q3	last viceroy of india	1	1
Q4	ind ka prim minister kaun hai	0	0.2
Q5	unka kitna age	0	0
Q6	gov jobs jiska exam hota hai	0	0.2
Q7	festivals here	0	0
Q8	mickel jackson fav step	0	0
Q9	mumbai to kolkata kese jana hai	0.3	0.28
Q10	what people here eat	0	0
Q11	wannacry kya hai (Before updating the NER list)	0.3	0.2
Q12	wannacry kya hai (After updating the NER list)	0.3	0.2
Q13	Badrnath ke pas landslde	0.8	0.7
Q14	wht hapnd thr	0	0
Q15	wahan ka temp kitna	0	0
Q16	zomato dat thft	1	1
Q17	book fr mchine lernin	0.9	0.8
Q18	places 2 vist at bbsr	0.7	0.6
Q19	wahn famous ky hai	0	0
Q20	odisha assembly ka speaker	1	1

Table 7.2: Precision of Original Query

Query No.	**Reformulated Query**	**P@3**	**P@5**
Q1	bharat's capital	0.6	0.5
Q2	capital of bharat	0.6	0.5
Q3	last viceroy of india	1	1
Q4	india's prime minister who is	1	0.95
Q5	india's prime minister how much age	1	1
Q6	india govt job whose exam have	0.65	0.7
Q7	festivals india	1	1
Q8	mickel jackson favourite step	0	0
Q9	mumbai to kolkata how to go	0.6	0.73
Q10	what people india eat	0.6	0.5
Q11	want to cry what is (Before updating the NER List)	0	0
Q12	wannacry what is (After updating the NER List)	1	0.9
Q13	badrinath near landslide	0.9	1
Q14	what happened badrinath	0	0.2
Q15	badrinath's temperature	0.8	0.7
Q16	zomato data theft	1	1
Q17	book for machine learning	1	0.9
Q18	places to visit at bhubaneswar	1	0.8
Q19	bhubaneswar famous what is	0.8	0.8
Q20	odisha assembly's speaker	1	1

Table 7.3: Precision of Reformulated Query

The relevancy score is awarded on the basis of what the user intend to search but not on what search query he is entering alone.

Chapter 8

Conclusion

Our effort was to address the following issues pertaining to search system:

1. The popular search engines, perform terribly when Hindi-English code mixed queries are searched.
2. They give a list of recommended queries for our query, but do not auto refine and reformulate them.
3. Each query is treated as individual entity and the inherent connection between them is ignored.
4. Thus, each time the user has to reformulate themselves and enter a new query to get relevant results.
5. Multi-lingual users and social media addicted users, can search using mixed query of Hindi and English.
6. Higher user satisfaction.

Our desktop search application performed well with Code-Mixed Hindi-English query. It was also able to handle the unstructured files on our computers in an efficient way. The queries were automatically reformulated upon discovery of continuity of concept in the subsequent queries. The use of Apache Solr and Apache Tika allowed us to index various text rich file types. Using our application, users were able to search their computer system for existence of relevant files. The search was done not only on the basis of file names but using the file meta-data and their contents too. It also gave the users the ease of searching by allowing to enter search queries as Hindi-English code-mixed query. The users demonstrated higher level of satisfaction due to the increased ease of use.

We intend to make the application as a generic one. The user will be able to provide own training files to train the system. In this way, multilingual users will be able to train the system to work with other code-mixed query languages. But care should be taken that the script for the input query has to be roman.

In the future, we intend to work to include Odia language into our study. We also intend to create a query log of code mixed multilingual search queries. We would like to improve upon the accuracy of language identification and normalization module. Our extend aim is to create a personalized search product that can search our computer system intelligently understanding or intention. This can also be integrated with websites for better intra-site search experience for multilingual users. We are aiming to further extend our work and build a shallow parser for Hindi-Odia-English, which will benefit the local users and help in NLP research.

Dissemination of Work

1. Amit Jena and Rakesh Chandra Balabantaray. "Query Optimization using Query Sequence for Hindi-English Code-Mixed Query." 2nd International Conference on Sustainable Computing Techniques in Engineering, Science and Management (SCESM 2017).
   Published in: International Journal of Control Theory and Applications.

2. Amit Jena and Rakesh Chandra Balabantaray. "Semantic Desktop Search Application for Hindi-English Code-Mixed user Query with Query Sequence analysis."
   Presented in: the 2nd IEEE International Conference On Recent Trends In Electronics, Information & Communication Tehnology (IRTEICT 2017).

Bibliography

[1] Brown, P.F., Pietra, V.J.D., Pietra, S.A.D. and Mercer, R.L., 1993. The mathematics of statistical machine translation: Parameter estimation. Computational linguistics, 19(2), pp.263-311.

[2] Chirita, P.A., Gavriloaie, R., Ghita, S., Nejdl, W. and Paiu, R., 2005, May. Activity based metadata for semantic desktop search. In European Semantic Web Conference (pp. 439-454). Springer Berlin Heidelberg.

[3] Foo, S. and Hendry, D., 2007, December. Desktop search engine visualisation and evaluation. In International Conference on Asian Digital Libraries (pp. 372-382). Springer Berlin Heidelberg.

[4] Schumacher, K., Sintek, M. and Sauermann, L., 2008, June. Combining fact and document retrieval with spreading activation for semantic desktop search. In European Semantic Web Conference (pp. 569-583). Springer Berlin Heidelberg.

[5] Wang, H.F., Lee, K.F. and Yang, Q., Microsoft Corporation, 2004. Search engine with natural language-based robust parsing for user query and relevance feedback learning. U.S. Patent 6,766,320.

[6] Kirsch, S.T., Chang, W.I. and Miller, E.R., Infoseek Corporation, 1999. Real-time document collection search engine with phrase indexing. U.S. Patent 5,920,854.

[7] White, R.W., Chu, W., Hassan, A., He, X., Song, Y. and Wang, H., 2013, May. Enhancing personalized search by mining and modeling task behavior. In Proceedings of the 22nd international conference on World Wide Web (pp. 1411-1420). ACM.

[8] Sharma, A., Gupta, S., Motlani, R., Bansal, P., Srivastava, M., Mamidi, R. and Sharma, D.M., 2016. Shallow Parsing Pipeline

for Hindi-English Code-Mixed Social Media Text. arXiv preprint arXiv:1604.03136.

[9] Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J.R., Bethard, S. and McClosky, D., 2014, June. The stanford corenlp natural language processing toolkit. In ACL (System Demonstrations) (pp. 55-60).

[10] Huang, J. and Efthimiadis, E.N., 2009, November. Analyzing and evaluating query reformulation strategies in web search logs. In Proceedings of the 18th ACM conference on Information and knowledge management (pp. 77-86). ACM.

[11] Dang, V. and Croft, B.W., 2010, February. Query reformulation using anchor text. In Proceedings of the third ACM international conference on Web search and data mining (pp. 41-50). ACM.

[12] Cucerzan, S. and Brill, E., 2004, July. Spelling Correction as an Iterative Process that Exploits the Collective Knowledge of Web Users. In EMNLP (Vol. 4, pp. 293-300).

[13] Hassan, A., 2013. Identifying Web Search Query Reformulation using Concept based Matching. In EMNLP (Vol. 13, pp. 1000-1010).

[14] Boldi, P., Bonchi, F., Castillo, C., Donato, D., Gionis, A. and Vigna, S., 2008, October. The query-flow graph: model and applications. In Proceedings of the 17th ACM conference on Information and knowledge management (pp. 609-618). ACM.

[15] Gao, J., He, X., Xie, S. and Ali, A., 2012, July. Learning lexicon models from search logs for query expansion. In Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (pp. 666-676). Association for Computational Linguistics.

[16] White, R.W., Chu, W., Hassan, A., He, X., Song, Y. and Wang, H., 2013, May. Enhancing personalized search by mining and modeling task behavior. In Proceedings of the 22nd international conference on World Wide Web (pp. 1411-1420). ACM.

[17] Anick, P., 2003. Learning noun phrase query segmentation. In Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval (pp. 88-95).

[18] Bergsma, S. and Wang, Q.I., 2007, June. Learning Noun Phrase Query Segmentation. In EMNLP-CoNLL (Vol. 7, pp. 819-826).

[19] Mitra, M., Singhal, A. and Buckley, C., 1998, August. Improving automatic query expansion. In Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval (pp. 206-214). ACM.

[20] The British National Corpus, version 3 (BNC XML Edition). 2007. Distributed by Oxford University Computing Services on behalf of the BNC Consortium. URL: http://www.natcorp.ox.ac.uk/

[21] Chirita, P.A., Costache, S., Nejdl, W. and Paiu, R., 2006, June. Beagle++: Semantically enhanced searching and ranking on the desktop. In European Semantic Web Conference (pp. 348-362). Springer Berlin Heidelberg.

[22] Jena, Amit, Chandra Balabantaray, Rakesh, 2017, January. Query Optimization using Query Sequence for Hindi-English Code-Mixed Query. In 2nd International Conference on Sustainable Computing Techniques in Engineering, Science and Management (SCESM 2017).

[23] Mogotsi, I.C., 2010. Christopher d. manning, prabhakar raghavan: Introduction to Information Retrieval.