Chapter 7

Code-Mixed Information Retrieval

After exploring the word-level and sentence-level text processing on code-mixed data, we transition to another research objective (Section 1.8) of this thesis, which is at a higher granularity than a sentence. We try to explore code-mixed information retrieval (CMIR) in this chapter. The flow of this chapter is structured as follows. Section 7.1 provides a detailed problem statement that we are going to address in this chapter. The text collection constructed and used in the experiments is described in Section 7.3. To enhance the performance of CMIR, three experiments are conducted. Section 7.4 demonstrates a language identification-based solution with query expansion using various phonetic algorithms. In Section 7.5, the impact of stop words on retrieval efficiency in CMIR is examined. Each of these sections encompasses corresponding research questions, methodology, experimental setup, results, and discussion.

7.1 Problem Statement

In CMIR, query terms and documents belong to different languages which may be using their native scripts or non-native ones. Here, both query and documents can contain multiple languages and scripts.

Mathematically,

query $q \in \left\langle L^{(i)}, S^{(j)} \right\rangle$, $i \geq 2$ and $j \geq 1$, and

the document pool, $\mathcal{D} = \bigcup D_{L^{(k)},S^{(p)}}$

where,

$$\begin{aligned} L^{(k)} &= \{l_1, l_2, \ldots, l_k\}, \\ S^{(p)} &= \{s_1, s_2, \ldots, s_p\} \quad \text{and} \\ D_{L^{(k)},S^{(p)}} &= \text{documents in any language from } L^{(k)} \text{ written in any script from } S^{(p)}. \end{aligned}$$

Ideally, there may not be any common language between $L^{(i)}$ and $L^{(k)}$.

CMIR is challenging because of the context of the query. A single string could represent different words with different meanings across languages. For example, the word 'book' when written as a transliterated Bengali word means 'chest'. The word 'rate' when written as a transliterated Bengali word means 'at night'. The word 'pray' when written as a transliterated Bengali word means 'almost'. Moreover, the difficulties faced in retrieval are not merely due to the context in which the word is used but also due to the different spellings used. For example, in transliterated Bengali, the 'chest' can also be written as 'buk' or 'boook' and so on. There is neither a well-accepted standard spelling rules for transliteration and even if it exists, neither people necessarily follow them in social media conversation.

Typically, search queries in various search engines or social media platforms are not code-mixed. In other words, users do not use code-mixing while creating queries in traditional search engines [103]. Also, search queries typically contain a terse set of keywords, whereas the queries posed in social media are often long verbose paragraphs that may lead to query dilution and/or query drift. On top of that, the code-mixed query along with the problem of non-standard spellings makes the task of matching queries with documents more difficult and leads to poor retrieval performance.

Broadly, we can therefore frame a research question as to how the users can be

provided with the most relevant responses (comments or replies in case of conversations) or how the responses can be ordered based on their topical relevance to the original query or statement. Can traditional IR be of help here? How can it be applied to deal with the new scenario, where we have multiple languages and multiple scripts with uncontrolled spellings and language constructs? What modifications do we need to incorporate in the traditional IR setting or what pre-processing and post-processing steps are needed?

However, the task does not have a required text collection. Hence we had to start with building a text collection first.

7.2 Research Questions (RQs)

In this chapter, we can formulate following two broad research questions (RQs) related to this task.

RQ-1: What are the challenges in building a code-mixed text collection for information retrieval-related task?

RQ-2: How to tackle CMIR challenges and improve the performance in terms of MAP?

In the subsequent sections, we attempt to address the RQs.

7.3 Building a Text Collection

A code-mixed dataset is not readily available on the internet for a CMIR task. A substantial amount of time is spent on collecting data to build our initial dataset. Social media platforms are considered to be the best sources of data for our work, hence we crawl through Facebook public pages and groups to collect the required data. We limit ourselves to considering the Bengali language for our task. It is the native language of Bangladesh and the Indian province of West Bengal. There is a humongous Bengali-

speaking population across the world, especially in various Indian metropolitan cities like Hyderabad, Bangalore, Delhi, Mumbai, Chennai and so on. While crawling through Facebook, we observe that people post a plethora of queries in the numerous Facebook groups related to the Bengali populace and others respond with their comments. For example, in a public community group of Bengali people living in Mumbai, one enquires about a good Bengali cuisine restaurant in a particular locality. Other members in the forum respond, while some provide factual answers, some joke, bully or provide sarcastic responses. Often the volume of responses are difficult to fathom, if the initial post is of controversial nature.

We customize the traditional IR setting as follows. The content of these original posts can be considered queries and the responses / comments to an original post as the documents from which relevant information needs to be retrieved.

AAs depicted in Figure 7.1, we treat a Facebook post as a query or topic, and the associated comments as documents. Among these documents, some are relevant to the query or post, while others are not. Initially, we collected over 50 topics, but later observed that some of these topics were nearly identical. Since their needs were the similar, we removed such topics and retained only the dissimilar ones to ensure diversity in the topic set.

Finally, we built a dataset comprising of 50 queries and 107900 documents. The statistics of the dataset are given below.

Table 7.1: Collection statistics

Attributes	**Values**
Document format	Text
No. of documents in the corpus	107900
Total no. of terms	1363672
No. of unique terms	84724
No. of topics	50
No. of relevant documents	802
Mean no. of relevant documents per query	16.04

Figure 7.1: Screenshot of a social media post showing a query and the corresponding documents

7.3.1 Data Pre-processing

The following procedure is followed to transform the crawled raw dataset into a standard TREC format. First, all individual comments are converted into UTF-8 format so that they may be read by any text editor. All user names stated in the comments are erased

to protect the privacy of users. Comments having less than four words are ignored. Each file is issued a unique number <DOCNO>. Finally, each file is converted into a standard TREC format by appending an opening <DOC> tag at the start and a closing < /DOC> tag at the end. To encapsulate the comments text, use the <TEXT> tag and make a separate file with *.trec* file extension. Figure 7.2, 7.3 and 7.4 illustrate some examples of query representation of Bengali-English code-mixed data, and Figure 7.5 a document. Here, title <TITLE>, description <DESC> and narrative <NARR> are the same to conform to TREC format. Our experiments only look at the title and description sections of a query. In general, the query length in any traditional IR system is quite short, on a scale of two to three words, while documents are much longer than queries. But in our study, the query length is slightly longer, and the document length is short. Figure 7.6 and Figure 7.7 show the length distributions of the query and documents respectively.

<TOP>
<NUM>18< /NUM>
<TITLE>
kondapur kothaguda side e bhalo unisex saloon ki ache please help as i have just shifted to hyderabad
< /TITLE>
<DESC>
kondapur kothaguda side e bhalo unisex saloon ki ache please help as i have just shifted to hyderabad
< /DESC>
<NARR>
kondapur kothaguda side e bhalo unisex saloon ki ache please help as i have just shifted to hyderabad
< /NARR>
< /TOP>

Figure 7.2: An example of code-mixed query (Topic Number 18)

<TOP>
<NUM>1< /NUM>
<TITLE>
hyderabad to howrah kono train ki diyeche ba debe durgapur jete hobe any idea jodi train chare then timing gulo ektu help korben
< /TITLE>
<DESC>
hyderabad to howrah kono train ki diyeche ba debe durgapur jete hobe any idea jodi train chare then timing gulo ektu help korben
< /DESC>
<NARR>
hyderabad to howrah kono train ki diyeche ba debe durgapur jete hobe any idea jodi train chare then timing gulo ektu help korben
< /NARR>
< /TOP>

Figure 7.3: An example of code-mixed query (Topic Number 1)

7.3.2 Pooling and Judgment

The quality of any IR system is determined by user satisfaction, which can be quantified using a number of precision or recall-based metrics. Both these basic metrics are defined in terms of the number of relevant documents. In other words, we need to have ground-truth information or relevance judgments for the collection in hand. For big datasets, judging the entire collection is prohibitively expensive in terms of manpower and money, if not impossible. From a practical point of view, relevance judgments are therefore usually done using a sampling-based technique known as *pooling*. For each query, we choose top-k documents (k is determined based on practicality, typically any number between 50 to 100) from the ranked result lists of a sufficiently diverse and large number of systems (at least 10) to create a pool. This pool of documents is exhaustively judged for relevance and considered as representative of the entire text collection. Even though the pool may not contain *all* the relevant documents for the query, if the systems are diverse and sufficiently large in numbers, they together can retrieve *almost all* the relevant documents within the top-k documents. There may be some relevant

<TOP>
<NUM>7< /NUM>
<TITLE>
hello all keu ki recently hampi travel korechen if yes what all are open in hampi admist covid and which route did you follow from hyderabad are the road conditions fine
< /TITLE>
<DESC>
hello all keu ki recently hampi travel korechen if yes what all are open in hampi admist covid and which route did you follow from hyderabad are the road conditions fine
< /DESC>
<NARR>
hello all keu ki recently hampi travel korechen if yes what all are open in hampi admist covid and which route did you follow from hyderabad are the road conditions fine
< /NARR>
< /TOP>

Figure 7.4: An example of code-mixed query (Topic Number 7)

<DOC>
<DOCNO>456< /DOCNO>
<TEXT>
salon ache anake oi road e first you have to go cothoguda circle then take google road left side e sob shop peye jaben
< /TEXT>
< /DOC>

Figure 7.5: An example of code-mixed document

documents in the collection outside the pool, but they are not judged and therefore ignored in the evaluation. In this study, a pool based on TREC specifications [128] is created for a collection of 50 topics using fourteen (14) distinct models [1], including BM25, term frequency-inverse document retrieval (TF- IDF), In_expB2, In_expC2, and InL2.

Using the Terrier IR platform, the top 100 ranked documents are pooled for rele-

[1] `http://terrier.org/docs/v4.2/configure_retrieval.html`

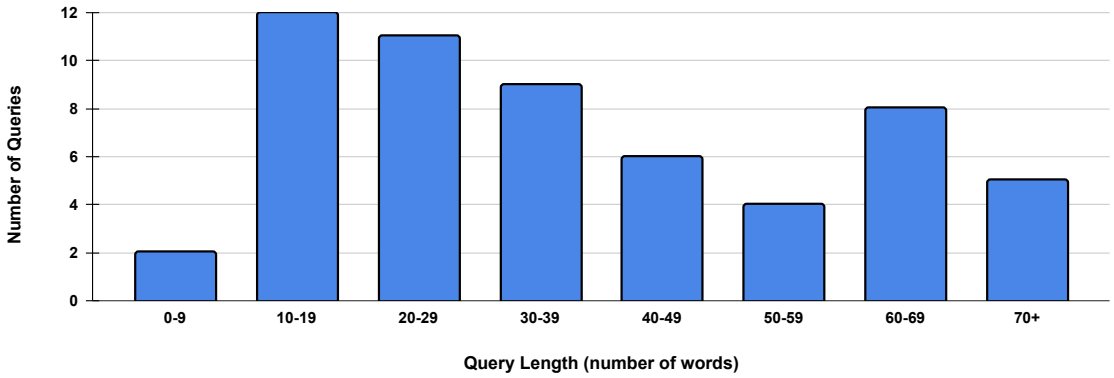

Figure 7.6: Length distribution of queries

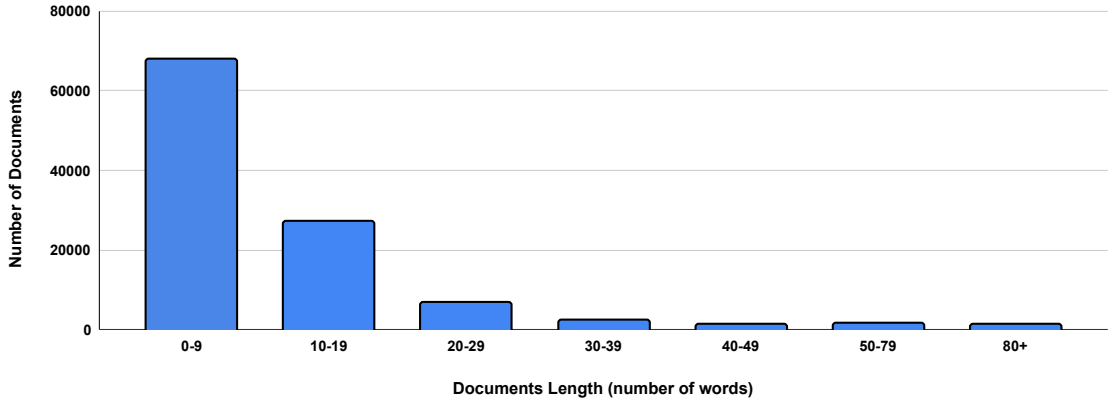

Figure 7.7: Length distribution of documents

vance assessment for each topic. This approach results in the retrieval of 802 relevant documents from a total of 11597 relevance judgments on 50 topics involving 8385 unique documents. The pool is then independently judged for binary relevance by two undergraduate students specializing in computer science and engineering. Both students are bilingual, with good reading, writing, and speaking abilities in Bengali and English. Kohen's Kappa (κ) statistic is used to determine inter-annotator agreement. We obtain a Kappa value of 0.9, indicating acceptable agreement between the annotated pair ($\kappa = 0.6$ is considered as *acceptable agreement* while 0.8 as *good*).

In this chapter, we attempt to explore RQ-2 which mentioned in Section 7.2. We propose three solutions one after another to deal with code-mixed information retrieval.

We are going to discuss the three strategies in the following sections.

7.4 CMIR: A phonetic algorithms-based approaches

We propose a language-identification-based solution with query expansion using different phonetic algorithms to RQ-2. However, related to this broad-level RQ, there are a few sub-questions as follows.

RQ-2.1: Whether language identification is necessary for code-mixed IR? If yes, what is its effect in the retrieval effectiveness?

RQ-2.2: How the uncontrolled spelling variations are handled in the transliterated text, that is very much part of the code-mixed data? Can phonetic encoding help here? If yes, to what extent?

RQ-2.3: Among a number of phonetic encoding techniques, which one gives the best retrieval effectiveness in code-mixed IR?

7.4.1 Experimental Setup

For our experiments, We use the Terrier (Terabyte Retriever) [2] framework developed by the University of Glasgow. Terrier is an open-source modular platform that allows for rapid development of large-scale IR applications. It offers indexing and retrieval capabilities for retrieval models that have already been pre-defined in the Terrier retrieval system. All experiments are carried out on a personal Linux-based machine equipped with 16 GB of RAM and an i5 Processor.

Here we use a code-mixed collection of Bengali-English (BN-EN) documents with BN-EN queries written in Roman script. There can be following strategies to solve the problem.

- We can use code-mixed machine translation to generate monolingual documents and queries. Here the complete sentences are translated into English. We apply

[2] `http://terrier.org/`

the traditional IR model on the modified corpus in English domain.

- We can also use a language identification system and find the Bengali phrases used in both queries and documents. A machine translation (MT) system can be applied to translate only the Bengali phrases to English and then a traditional monolingual IR model can be applied on English-only text (original + translated). Since MT is applied partially, the sentences are not likely to be semantically meaningful in English.
- Thirdly, let the documents and query remain in their original formats in Roman script only. The retrieval is to be done using the Roman script. The sentences are semantically as per the original document, whether in Bengali or English or in mixed mode.

The difficulty of the first two approaches is the machine translation of code-mixed text into English (or to any language, *per se*). MT itself is an important research problem and that becomes a crucial component in our retrieval framework. The accuracy of the MT system will greatly influence the performance of our code-mixed retrieval system. Also, for the second case, partial translation of Bengali text (only phrases, but not the complete sentences) will lead to semantic incoherence.

We, therefore, choose the third strategy. All 50 queries used in this study are code-mixed. The majority of the documents in the collection are also code-mixed barring only a few documents which are monolingual (either English or Bengali). Both the documents and queries are written in Roman script. Two major problems in code-mixed text processing are language identification and spelling variation. Figure 7.8 depicts the entire process flow of our proposed methodology incorporating the strategies adopted to handle these two issues. The entire process is divided into several modules, starting with language identification and named entity recognition, followed by the extraction of identical terms with different spellings using several phonetic methods. Following subsections detail the steps.

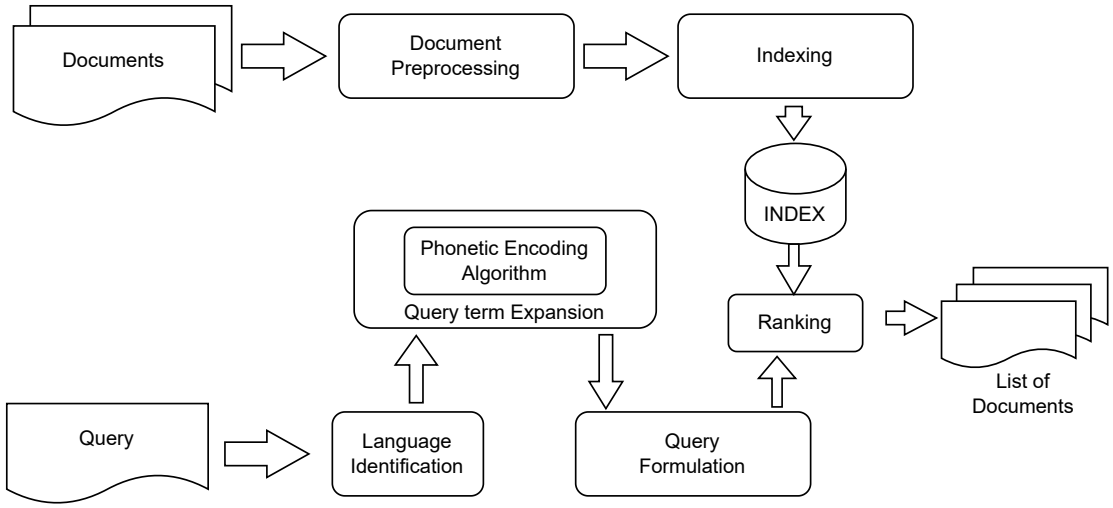

Figure 7.8: Process flow for retrieval

7.4.1.1 Language Identification and Named Entity Recognition

A query may contain terms from more than one language, which must be detected prior to query term expansion. The language that a given query word belongs to can be identified manually or automatically using a classifier. We use both the techniques. First, we use an in-house transformer-based language and named entity identifier as described in Chapter 4, supplemented by human annotation. We train our model using annotated corpora from the `ICON_POS`[3] dataset, and all query terms are classified using the trained model.

Let $q = \{q_1, q_2, \ldots q_n\}$ be a query where $q_1, q_2, \ldots q_n$ are individual terms. Each query word q_i is annotated and associated with a tag either BN or EN or NE. So, following example query demonstrates the annotation.

kondapur/NE kothaguda/NE side/EN e/BN bhalo/BN unisex/EN saloon/EN ki/BN ache/BN please/EN help/EN as/EN i/EN have/EN just/EN shifted/EN to/EN hyderabad/NE

[3] `http://www.amitavadas.com/code-mixing.html`

7.4.1.2 Handling Spelling Variation using Phonetic Encodings

For formal English, spelling variations are rare, though they can be found on social media in informal text. But for non-English languages, the problem is severe. Since there are no well-accepted spelling rules for transliterated text and one writes according to their mental mapping of phonemes to graphemes, spelling variations are obvious and observed very frequently. Transliterated Bengali often generates many different spellings for a given Bengali term. But for any string matching-based retrieval system to succeed, there should be no spelling differences between the queries and the documents. Thus, query expansion is required for both Bengali transliterated words and English terms to take into account their possible spelling variations. From the document collection, we notice spelling differences occur mostly in named entities. We therefore attempt to extract all possible variations of a given query term present in the corpus using phonetic encoding techniques and add them in the query using query expansion.

Homophonic words are those that have similar phonetic sounds but different spellings. We use phonetic matching techniques for each tagged query term to retrieve its homophonic counterparts.

For each query word encoded, matching encodings are searched in the corpus (dictionary of the index) and all such words having the same tag are added to the query to make an expanded query. The expanded query is passed to a retrieval model that yields a ranked list of documents. A general algorithm (Algorithm 7.1) corresponding to the the workflow (Figure 7.8) is given below.

In our experiments, we apply three distinct phonetic encoding techniques (Soundex, Phonix, and Hindex) to the query terms. We test all possible combinations of word tags and phonetic encodings to assess their influence on each word tag (BN, EN, and NE). Table 7.3 outlines each combination, with each experiment identified as a run with a unique number. The experiment in Run 0 is the baseline where neither any

Algorithm 7.1: Find candidate terms for a given query term

Input: Query(Q) = $\{\langle q_1, a_1\rangle, \langle q_2, a_2\rangle, \ldots \langle q_n, a_n\rangle\}$ where $a_i \in \{BN, EN, NE\}$ and n is number of terms present in a Query, Lexicon or collection of all words present in the documents, $L = L_{BN} \cup L_{EN} \cup L_{NE}$

Output: Return a modified query (Q_m)

```
1  function query_expn(Q)
2      Q = {⟨q1, a1⟩, ⟨q2, a2⟩, ..., ⟨qn, an⟩};
3      Q' = {q1, q2, ..., qn};
4      for i = 1 to n do
5          L = list(ai);
6          Choose encoding enc ;        /* enc ∈ {Soundex, Phonix, Hindex} */
7          qc = find_candidate(ai, L, enc);
8          Qm = Q' ∪ qc;
9      return Qm;
10 function find_candidate(q, L, enc)
11     qc = {};
12     Code_q ← enc(q);
13     for j = 1 to length(L) do
14         Code_lj ← enc(lj);
15         if Code_q == Code_lj then
16             qc ← qc ∪ {lj};
17     return qc;
```

language identification is applied nor any phonetic encoding is used on the original queries. Other runs (Run 1 - Run 29) correspond to language identification followed by phonetic encodings employed to generate homophonic terms for the shown word categories or word tag(s).

An example (query number 56) can explain Algorithm 7.1 better (see Table 7.2).

Query Number 56

Original Query: *paka* **tal** *kothay pabo ektu bolben* plz

English - Tell me please where can I get ripe Palmyra fruit

After LID and NE tagging: < paka, BN>< tal, NE >< kothay, BN >

< pabo, BN >< ektu, BN >< bolben, BN >< plz, EN >

Modified Query using Soundex:

paka tal tel tale tele til kothay pabo ektu bolben plz

Modified Query using Hindex:

paka tal taal kothay pabo ektu bolben plz

Table 7.2: An example of query expansion

Five different term-weight based retrieval models, namely BM25, TF-IDF, In_expB2, In_expC2, and InL2, are tried for each combination as indicated above (Table 7.3). We do not apply any stopword list or any stemmer. All the terms with their original phonetics are maintained in both queries and documents.

Table 7.3: All Experiment setup

Experiment	Word Tag			Experiment	Word Tag		
	Bengali (BN)	English (EN)	Name Entity (NE)		Bengali (BN)	English (EN)	Name Entity (NE)
Run 0	-	-	-	Run 15	-	-	Hindex
Run 1	-	-	Phonix	Run 16	-	Hindex	-
Run 2	-	Phonix	-	Run 17	Hindex	-	-
Run 3	Phonix	-	-	Run 18	Hindex	-	Hindex
Run 4	Phonix	-	Phonix	Run 19	Hindex	Hindex	-
Run 5	Phonix	Phonix	-	Run 20	-	Hindex	Hindex
Run 6	-	Phonix	Phonix	Run 21	Hindex	Hindex	Hindex
Run 7	Phonix	Phonix	Phonix	Run 22	Hindex	-	Phonix
Run 8	-	-	Soundex	Run 23	Hindex	-	Soundex
Run 9	-	Soundex	-	Run 24	Hindex	Phonix	-
Run 10	Soundex	-	-	Run 25	Hindex	Soundex	-
Run 11	Soundex	-	Soundex	Run 26	Hindex	Soundex	Phonix
Run 12	Soundex	Soundex	-	Run 27	Hindex	Phonix	Soundex
Run 13	-	Soundex	Soundex	Run 28	Hindex	Phonix	Phonix
Run 14	Soundex	Soundex	Soundex	Run 29	Hindex	Soundex	Soundex

7.4.2 Results

Extensive experiments are conducted to seek answers to the stated RQs, mentioned in Section 7.4. For each query, the models produce a ranked list of documents in decreasing order of probability of relevance as computed by the model. Such ranked lists are evaluated against ground-truths or relevance judgments (a file of query-document pairs called *qrels*) for each query. Retrieval effectiveness is measured by different metrics (defined in Chapter 2).

Evaluation scores on different retrieval models on each word tag are shown in Table 7.4 and 7.5. Table 7.4 shows the evaluation scores on different retrieval models where the word-level language identification is done by the machine, while Table 7.5 shows its counterparts done by humans. We include the scores corresponding to 29 different query combinations (shown in Table 7.3).

Ex.	IR Model	MAP	R-prec	P@10	Ex.	IR Model	MAP	R-prec	P@10
Run 0	BM25	0.1616	0.2000	0.2540	Run 15	BM25	0.1677	0.2043	0.2580
	TF-IDF	0.1549	0.1958	0.2460		TF-IDF	0.1606	0.1974	0.2520
	In_expB2	0.1167	0.1635	0.1980		In_expB2	0.1198	0.1659	0.2020
	In_expC2	0.1046	0.1395	0.1740		In_expC2	0.1068	0.1456	0.1820
	InL2	0.1009	0.1327	0.1720		InL2	0.1035	0.1357	0.1680
Run 1	BM25	0.1675	0.2062	0.2600	Run 16	BM25	0.1680 ↑	0.2162	0.2560
	TF-IDF	0.1613	0.2022	0.2560		TF-IDF	0.1603 ↑	0.1993	0.2400
	In_expB2	0.1201	0.1659	0.2040		In_expB2	0.1161 ↓	0.1464	0.1820
	In_expC2	0.1080	0.1467	0.1820		In_expC2	0.0988 ↓	0.1275	0.1540
	InL2	0.1051	0.1351	0.1660		InL2	0.1027 ↑	0.1279	0.1620
Run 2	BM25	0.1453	0.1809	0.2260	Run 17	BM25	0.1445	0.1789	0.2280
	TF-IDF	0.1413	0.1769	0.2200		TF-IDF	0.1392	0.1755	0.2260
	In_expB2	0.1059	0.1473	0.1720		In_expB2	0.0988	0.1399	0.1660
	In_expC2	0.0940	0.1274	0.1520		In_expC2	0.0858	0.1280	0.1460
	InL2	0.0956	0.1250	0.1480		InL2	0.0932	0.1252	0.1620

Continued on next page

Table 7.4 – continued from previous page

Ex.	IR Model	MAP	R-prec	P@10	Ex.	IR Model	MAP	R-prec	P@10
Run 3	BM25	0.0533	0.0976	0.1080	Run 18	BM25	0.1505	0.1898	0.2400
	TF-IDF	0.0524	0.0945	0.1060		TF-IDF	0.1450	0.1837	0.2340
	In_expB2	0.0243	0.0556	0.0520		In_expB2	0.1042	0.1470	0.1740
	In_expC2	0.0203	0.0453	0.0440		In_expC2	0.0900	0.1352	0.1480
	InL2	0.0369	0.0630	0.0680		InL2	0.0957	0.1330	0.1620
Run 4	BM25	0.0543	0.0983	0.1100	Run 19	BM25	0.1631 ↑	0.2050	0.2560
	TF-IDF	0.0539	0.0965	0.1080		TF-IDF	0.1594 ↑	0.2007	0.2480
	In_expB2	0.0256	0.0545	0.0520		In_expB2	0.1152 ↓	0.1509	0.1760
	In_expC2	0.0211	0.0473	0.0460		In_expC2	0.0983 ↓	0.1299	0.1480
	InL2	0.0384	0.0656	0.0680		InL2	0.1019 ↑	0.1363	0.1580
Run 5	BM25	0.0471	0.0773	0.1020	Run 20	BM25	0.1738 ↑	0.2225	0.2640
	TF-IDF	0.0467	0.0792	0.1020		TF-IDF	0.1651 ↑	0.2094	0.2500
	In_expB2	0.0254	0.0488	0.0520		In_expB2	0.1198 ↑	0.1538	0.1880
	In_expC2	0.0212	0.0421	0.0440		In_expC2	0.1019 ↓	0.1283	0.1560
	InL2	0.0375	0.0605	0.0700		InL2	0.1045 ↑	0.1314	0.1660
Run 6	BM25	0.1492	0.1913	0.2360	Run 21	BM25	0.1670 ↑	0.2102	0.2640
	TF-IDF	0.1446	0.1865	0.2300		TF-IDF	0.1636 ↑	0.2051	0.2520
	In_expB2	0.1106	0.1502	0.1760		In_expB2	0.1200 ↑	0.1547	0.1780
	In_expC2	0.0977	0.1269	0.1540		In_expC2	0.1025 ↓	0.1332	0.1540
	InL2	0.0985	0.1274	0.1520		InL2	0.1048 ↑	0.1403	0.1600
Run 7	BM25	0.0487	0.0813	0.1060	Run 22	BM25	0.1503	0.1908	0.2420
	TF-IDF	0.0484	0.0832	0.1060		TF-IDF	0.1455	0.1873	0.2380
	In_expB2	0.0265	0.0470	0.0520		In_expB2	0.1046	0.1463	0.1720
	In_expC2	0.0221	0.0441	0.0460		In_expC2	0.0911	0.1333	0.1460
	InL2	0.0386	0.0635	0.0700		InL2	0.0973	0.1311	0.1620
Run 8	BM25	0.1683	0.2067	0.2640	Run 23	BM25	0.1507	0.1907	0.2440
	TF-IDF	0.1615	0.2007	0.2580		TF-IDF	0.1450	0.1846	0.2380
	In_expB2	0.1196	0.1670	0.2080		In_expB2	0.1042	0.1457	0.1740
	In_expC2	0.1077	0.1479	0.1840		In_expC2	0.0906	0.1321	0.1460

Continued on next page

Table 7.4 – continued from previous page

Ex.	IR Model	MAP	R-prec	P@10	Ex.	IR Model	MAP	R-prec	P@10
	InL2	0.1037	0.1363	0.1700		InL2	0.0956	0.1293	0.1600
Run 9	BM25	0.1568 ↓	0.1981	0.2260	Run 24	BM25	0.1348	0.1772	0.2220
	TF-IDF	0.1515 ↓	0.1931	0.2240		TF-IDF	0.1309	0.1752	0.2120
	In_expB2	0.1146 ↓	0.1519	0.1800		In_expB2	0.1027	0.1414	0.1600
	In_expC2	0.0983 ↓	0.1315	0.1540		In_expC2	0.0893	0.1271	0.1440
	InL2	0.1050 ↑	0.1317	0.1520		InL2	0.0921	0.1240	0.1360
Run 10	BM25	0.0893	0.1234	0.1480	Run 25	BM25	0.1496 ↓	0.1773	0.2140
	TF-IDF	0.853	0.1219	0.1440		TF-IDF	0.1454 ↓	0.1760	0.2080
	In_expB2	0.0351	0.0674	0.0700		In_expB2	0.1125 ↓	0.1400	0.1700
	In_expC2	0.0298	0.0557	0.0620		In_expC2	0.0982 ↓	0.1315	0.1480
	InL2	0.0548	0.0753	0.0880		InL2	0.1029 ↑	0.1305	0.1560
Run 11	BM25	0.0960	0.1266	0.1540	Run 26	BM25	0.1543 ↓	0.1874	0.2220
	TF-IDF	0.0922	0.1263	0.1540		TF-IDF	0.1501 ↓	0.1860	0.2200
	In_expB2	0.0368	0.0704	0.0720		In_expB2	0.1176 ↑	0.1464	0.1760
	In_expC2	0.0312	0.0574	0.0600		In_expC2	0.0983 ↓	0.1299	0.1480
	InL2	0.0555	0.0803	0.0920		InL2	0.1019 ↑	0.1363	0.1580
Run 12	BM25	0.0999	0.1316	0.1560	Run 27	BM25	0.1386	0.1846	0.2280
	TF-IDF	0.0997	0.1252	0.1520		TF-IDF	0.1345	0.1831	0.2200
	In_expB2	0.0483	0.0753	0.1000		In_expB2	0.1074	0.1461	0.1660
	In_expC2	0.0385	0.0665	0.0760		In_expC2	0.0928	0.1302	0.1500
	InL2	0.0677	0.0870	0.0980		InL2	0.0944	0.1273	0.1440
Run 13	BM25	0.1611 ↓	0.2047	0.2360	Run 28	BM25	0.1394	0.1873	0.2300
	TF-IDF	0.1553 ↑	0.2002	0.2300		TF-IDF	0.1354	0.1852	0.2220
	In_expB2	0.1191 ↑	0.1532	0.1860		In_expB2	0.1078	0.1500	0.1660
	In_expC2	0.1028 ↓	0.1345	0.1580		In_expC2	0.0930	0.1315	0.1480
	InL2	0.1079 ↑	0.1319	0.1580		InL2	0.0954	0.1307	0.1440
Run 14	BM25	0.1048	0.1335	0.1620	Run 29	BM25	0.1537 ↓	0.1865	0.2240
	TF-IDF	0.1033	0.1293	0.1600		TF-IDF	0.1492 ↓	0.1851	0.2220
	In_expB2	0.0502	0.0764	0.1020		In_expB2	0.1175 ↑	0.1475	0.1800

Continued on next page

Table 7.4 – continued from previous page

Ex.	IR Model	MAP	R-prec	P@10	Ex.	IR Model	MAP	R-prec	P@10
	In_expC2	0.0401	0.0682	0.0780		In_expC2	0.1020 ↓	0.1336	0.1580
	InL2	0.0697	0.0894	0.1040		InL2	0.1056 ↑	0.1346	0.1620

Table 7.4: The results of retrieval effectiveness measured by MAP, R-prec, and P@10 when LID task is done by Machine. ↑ sign denotes that the model for this setup performs better than the baseline (Run 0). ↓ sign denotes that the model for this setup performs less than the baseline.

Ex.	IR Model	MAP	R-prec	P@10	Ex.	IR Model	MAP	R-prec	P@10
Run 0	BM25	0.1616	0.2000	0.2540	Run 15	BM25	0.1826	0.2255	0.2760
	TF-IDF	0.1549	0.1958	0.2460		TF-IDF	0.1774	0.2195	0.2740
	In_expB2	0.1167	0.1635	0.1980		In_expB2	0.1388	0.1815	0.2180
	In_expC2	0.1046	0.1395	0.1740		In_expC2	0.1233	0.1598	0.2000
	InL2	0.1009	0.1327	0.1720		InL2	0.1175	0.1486	0.1840
Run 1	BM25	0.1794	0.2248	0.2760	Run 16	BM25	0.1682 ↑	0.2139	0.2520
	TF-IDF	0.1737	0.2215	0.2740		TF-IDF	0.1607 ↑	0.2006	0.2400
	In_expB2	0.1398	0.1811	0.2240		In_expB2	0.1143 ↓	0.1445	0.1800
	In_expC2	0.1246	0.1594	0.1960		In_expC2	0.0978 ↓	0.1260	0.1540
	InL2	0.1167	0.1463	0.1800		InL2	0.1027 ↑	0.1265	0.1620
Run 2	BM25	0.1424	0.1750	0.2180	Run 17	BM25	0.1394	0.1780	0.2131
	TF-IDF	0.1381	0.1705	0.2140		TF-IDF	0.1350	0.1702	0.2220
	In_expB2	0.1031	0.1365	0.1640		In_expB2	0.0922	0.1308	0.1540
	In_expC2	0.0908	0.1223	0.1440		In_expC2	0.0810	0.1199	0.1360
	InL2	0.0935	0.1181	0.1420		InL2	0.0883	0.1215	0.1520
Run 3	BM25	0.0530	0.0902	0.1120	Run 18	BM25	0.1589 ↓	0.1967	0.2500
	TF-IDF	0.0522	0.0885	0.1060		TF-IDF	0.1528 ↓	0.1907	0.2440
	In_expB2	0.0224	0.0533	0.0540		In_expB2	0.1132 ↓	0.1490	0.1760
	In_expC2	0.0188	0.0432	0.0460		In_expC2	0.0997 ↓	0.1372	0.1480
	InL2	0.0375	0.0644	0.0640		InL2	0.1018 ↑	0.1410	0.1640
	BM25	0.0597	0.0973	0.1220		BM25	0.1576	0.1983	0.2460
	TF-IDF	0.0594	0.0955	0.1140		TF-IDF	0.1527	0.1938	0.2360

Continued on next page

Table 7.5 – continued from previous page

Ex.	IR Model	MAP	R-prec	P@10	Ex.	IR Model	MAP	R-prec	P@10
Run 4	In_expB2	0.0299	0.0576	0.0620	Run 19	In_expB2	0.1060	0.1406	0.1660
	In_expC2	0.0246	0.0529	0.0560		In_expC2	0.0901	0.1201	0.1420
	InL2	0.0439	0.0740	0.0780		InL2	0.0950	0.1272	0.1500
Run 5	BM25	0.0428	0.0733	0.0940	Run 20	BM25	0.1861	0.2342	0.2740
	TF-IDF	0.0423	0.0728	0.0900		TF-IDF	0.1776	0.2209	0.2660
	In_expB2	0.0235	0.0481	0.0540		In_expB2	0.1349	0.1698	0.2080
	In_expC2	0.0194	0.0421	0.0520		In_expC2	0.1157	0.1446	0.1780
	InL2	0.0371	0.0564	0.0640		InL2	0.1133	0.1380	0.1720
Run 6	BM25	0.1560 ↓	0.2014	0.2440	Run 21	BM25	0.1738	0.2186	0.2700
	TF-IDF	0.1498 ↓	0.1958	0.2420		TF-IDF	0.1694	0.2132	0.2580
	In_expB2	0.1204 ↑	0.1572	0.1940		In_expB2	0.1253	0.1595	0.1820
	In_expC2	0.1074 ↑	0.1424	0.1720		In_expC2	0.1074	0.1368	0.1600
	InL2	0.1035 ↑	0.1350	0.1640		InL2	0.1064	0.1404	0.1620
Run 7	BM25	0.0482	0.0825	0.1080	Run 22	BM25	0.1560	0.1928	0.2520
	TF-IDF	0.0476	0.0836	0.1040		TF-IDF	0.1497	0.1876	0.2440
	In_expB2	0.0297	0.0582	0.0660		In_expB2	0.1123	0.1477	0.1740
	In_expC2	0.0247	0.0542	0.0620		In_expC2	0.1011	0.1365	0.1480
	InL2	0.0414	0.0682	0.0720		InL2	0.1005	0.1359	0.1640
Run 8	BM25	0.1846	0.2278	0.2800	Run 23	BM25	0.1603 ↓	0.1984	0.2560
	TF-IDF	0.1798	0.2245	0.2800		TF-IDF	0.1541 ↓	0.1935	0.2480
	In_expB2	0.1405	0.1855	0.2260		In_expB2	0.1138 ↓	0.1487	0.1780
	In_expC2	0.1261	0.1658	0.2020		In_expC2	0.1012 ↓	0.1359	0.1480
	InL2	0.1194	0.1492	0.1840		InL2	0.1035 ↑	0.1374	0.1620
Run 9	BM25	0.1550 ↓	0.1923	0.2180	Run 24	BM25	0.1296	0.1742	0.2060
	TF-IDF	0.1495 ↓	0.1847	0.2180		TF-IDF	0.1258	0.1690	0.2020
	In_expB2	0.1110 ↓	0.1483	0.1700		In_expB2	0.0932	0.1295	0.1480
	In_expC2	0.0950 ↓	0.1274	0.1460		In_expC2	0.0803	0.1145	0.1300
	InL2	0.1018 ↑	0.1273	0.1440		InL2	0.0859	0.1119	0.1300
	BM25	0.0874	0.1196	0.1400		BM25	0.1438	0.1762	0.2080

Continued on next page

Table 7.5 – continued from previous page

Ex.	IR Model	MAP	R-prec	P@10	Ex.	IR Model	MAP	R-prec	P@10
Run 10	TF-IDF	0.0837	0.1175	0.1440	Run 25	TF-IDF	0.1391	0.1727	0.2040
	In_expB2	0.0324	0.0630	0.0720		In_expB2	0.1039	0.1298	0.1600
	In_expC2	0.0274	0.0536	0.0620		In_expC2	0.0889	0.1172	0.1360
	InL2	0.0549	0.0768	0.0900		InL2	0.0947	0.1196	0.1460
Run 11	BM25	0.1032	0.1331	0.1600	Run 26	BM25	0.1576 ↓	0.1938	0.2320
	TF-IDF	0.0999	0.1313	0.1620		TF-IDF	0.1513 ↓	0.1909	0.2280
	In_expB2	0.0445	0.0732	0.0840		In_expB2	0.1237 ↑	0.1506	0.1840
	In_expC2	0.0377	0.0650	0.0700		In_expC2	0.1068 ↑	0.1396	0.1640
	InL2	0.0651	0.0927	0.1040		InL2	0.1050 ↑	0.1384	0.1660
Run 12	BM25	0.0970	0.1255	0.1420	Run 27	BM25	0.1275	0.1554	0.2040
	TF-IDF	0.0969	0.1206	0.1420		TF-IDF	0.1229	0.1520	0.1980
	In_expB2	0.0451	0.0734	0.0940		In_expB2	0.0886	0.1279	0.1560
	In_expC2	0.0357	0.0617	0.0780		In_expC2	0.0783	0.1125	0.1340
	InL2	0.0660	0.0832	0.0960		InL2	0.0872	0.1182	0.1440
Run 13	BM25	0.1760	0.2156	0.2540	Run 28	BM25	0.1438	0.1991	0.2380
	TF-IDF	0.1705	0.2142	0.2540		TF-IDF	0.1368	0.1904	0.2300
	In_expB2	0.1364	0.1685	0.2000		In_expB2	0.1081	0.1492	0.1680
	In_expC2	0.1191	0.1488	0.1780		In_expC2	0.0940	0.1353	0.1540
	InL2	0.1195	0.1504	0.1760		InL2	0.0948	0.1320	0.1540
Run 14	BM25	0.1111	0.1418	0.1600	Run 29	BM25	0.1615	0.2014	0.2400
	TF-IDF	0.1097	0.1398	0.1640		TF-IDF	0.1564 ↑	0.1984	0.2360
	In_expB2	0.0577	0.0874	0.1120		In_expB2	0.1263 ↑	0.1553	0.1920
	In_expC2	0.0471	0.0727	0.0920		In_expC2	0.1095 ↑	0.1415	0.1680
	InL2	0.0774	0.0978	0.1140		InL2	0.1087 ↑	0.1427	0.1740

Table 7.5: The results of retrieval effectiveness measured by MAP, R-prec, and P@10 when LID task is done by Human. ↑ sign denotes that the model for this setup performs better than the baseline (Run 0). ↓ sign denotes that the model for this setup performs less than the baseline.

RQ-2.1: Whether language identification is necessary for code-mixed IR? If yes, what is its effect in the retrieval effectiveness?

No, it is not absolutely necessary. In Run 0, we get a decent performance without using any LID and NE tagging at all. However, we get substantial performance gain in terms of MAP in Run 1, Run 8, Run 13, Run 15, Run 16, Run 20, Run 21 and Run 29 where LID followed by NE tagging is done.

To investigate the effect of the language identification module on CMIR retrieval effectiveness, we use two types of word tagging, one is system generated, and another human-annotated. From Table 7.4 and 7.5, we can see that in both the cases MAP scores improve for majority of the runs.

RQ-2.2: How the uncontrolled spelling variations are handled in the transliterated text, which is very much part of the code-mixed data? Can phonetic encoding help here? If yes, to what extent?

The phonetic encoding approaches are seen to enhance the effectiveness of code-mixed IR in general. However, the performance depends on the type of words to which the phonetic encoding is applied. In our experiments, we employ 29 separate runs that combine phonetic encodings with three different forms of word tagging.

People often spell words in various ways on social media, particularly for named entities (NE). We apply phonetic encoding to NE tags (Run 1, Run 8, and Run 15), where Soundex consistently performs better than other phonetic encoding methods. Figure 7.9 illustrates that Soundex achieves the highest mean average precision (MAP) at 0.1500, followed by Hindex at 0.1479 and Phonix at 0.1468. Given the different MAP scores across the five IR models, we calculate the mean of these scores for comparison.

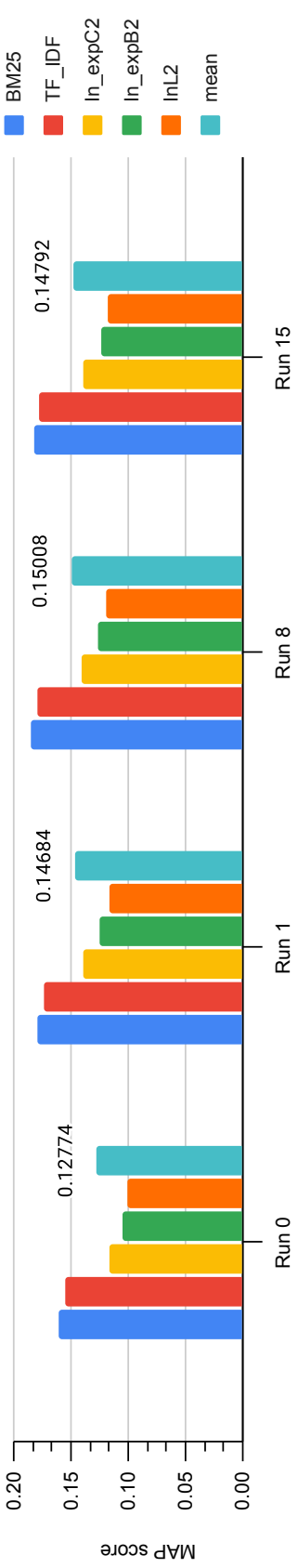

Figure 7.9: MAP on different retrieval model

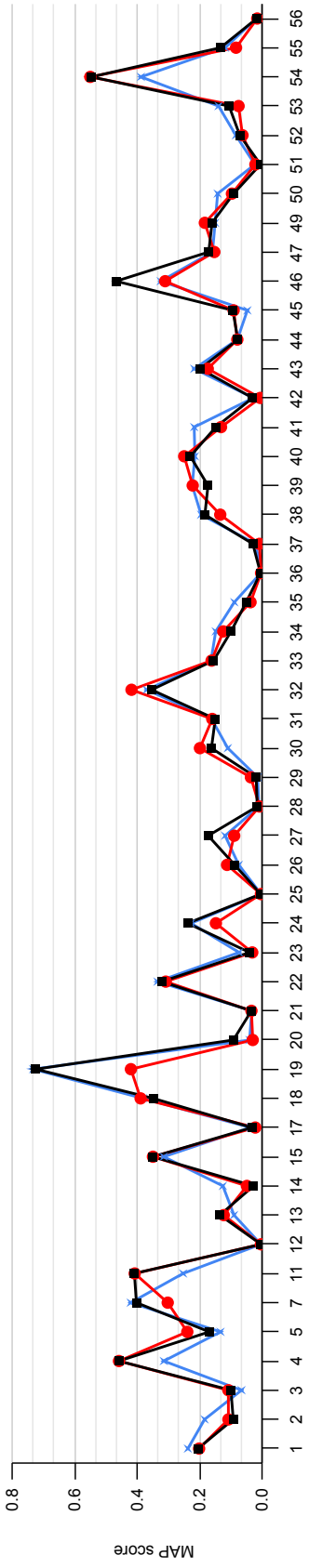

Figure 7.10: Query wise performance changes (Run 0, Run 9 and Run 16) by BM25 model

We also observe variations in the EN tags. When we apply Hindex to EN tags (Run 16), we notice a performance improvement (see Figure 7.10).

The MAP scores of Run 1, Run 8, Run 15, and Run 16 clearly indicate that phonetic encoding effectively handle spelling variations.

RQ-2.3: Among a number of phonetic encoding techniques, which one gives the best performance in code-mixed IR?

Among the three phonetic encoding techniques used, the best performance depends on LID and NE tags, to which the phonetic encoding is applied. Overall, Hindex (Run 20) outperforms the rest of the methods. Soundex provides almost identical performance to Hindex. If we see the MAP score of Run 8 and Run 15 (see Figure 7.9), the score is nearly identical to Soundex. However, when analyzing the average precision of each query, we find that Soundex performs better on some queries while Hindex excels on others. Specifically, using Soundex, we observe a performance improvement in 23 queries, while with Hindex, the number is 14. For 13 queries, the performance remains the same. Soundex outperforms its peers (see Figure 7.11).

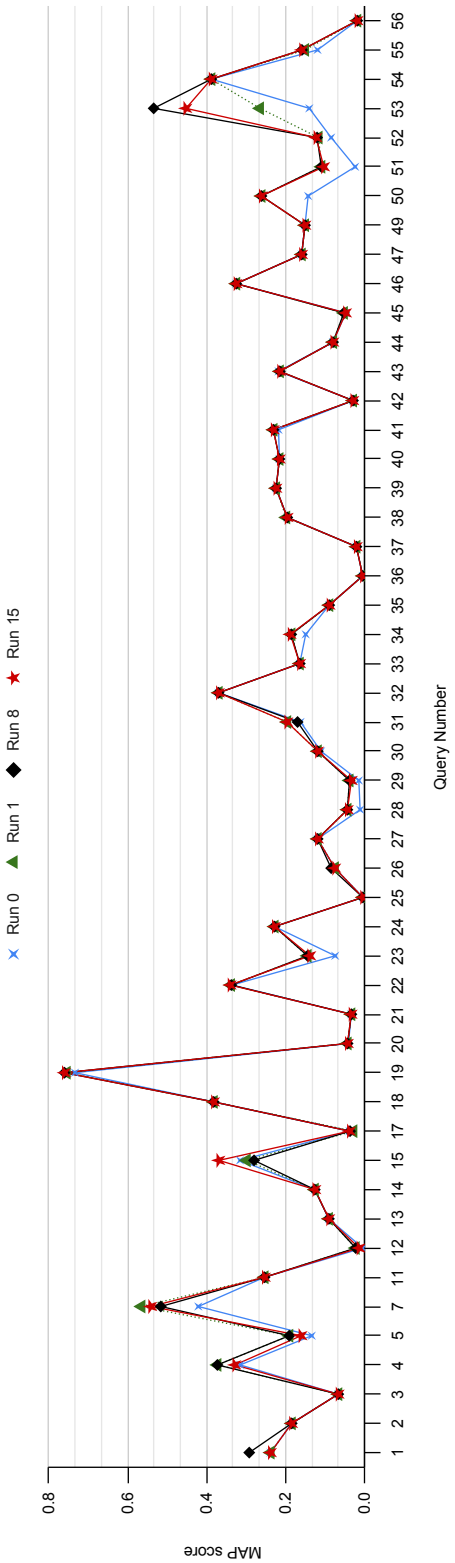

Figure 7.11: Performance changes (Run 0, Run 1, Run 8 and Run 15) across queries by BM25 model

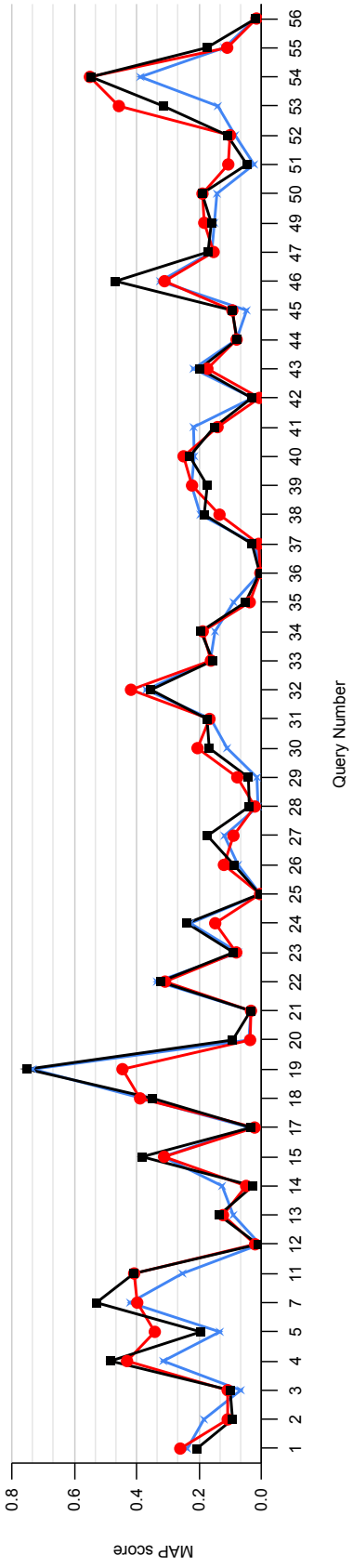

Figure 7.12: Query wise performance changes (Run 0, Run 13 and Run 20) by BM25 model

Soundex outperforms the other methods with a MAP score of 0.1826 (Run 15) when considering only NE tags. For EN tags, the Hindex performs better, achieving a MAP score of 0.1682, and applying the Hindex to both NE and EN tags (Run 20) results in a performance improvement, with a MAP score of 0.1861. However, using Soundex for both EN and NE tags (Run 13) leads to a performance drop (see Figure 7.12).

Table 7.5 indicates a significant decrease in retrieval effectiveness when Phonix is used for BN tags (Run 3, 4, 5, and 7). Both Phonix and Soundex demonstrate poor performance with BN tags (Run 10, 11, 12, and 14).

7.4.3 Discussion

In this section, we attempt to discuss the retrieval effectiveness at the query level, with some insights gained. The performance analysis here are based on Table 7.5. Among the five standard IR models we use, BM25 is seen to consistently perform the best demonstrating the highest MAP in all the runs.

7.4.3.1 Best performance over baseline

Among the runs, Run 20 (Hindex on NE and EN) outperforms others yielding a 15% performance boost in terms of MAP over the baseline (Run 0). When we assess the average precision of each query (refer to Figure 7.13), substantial increase in performance is observed for majority of the queries (35 queries) versus reduction in few (15 queries).

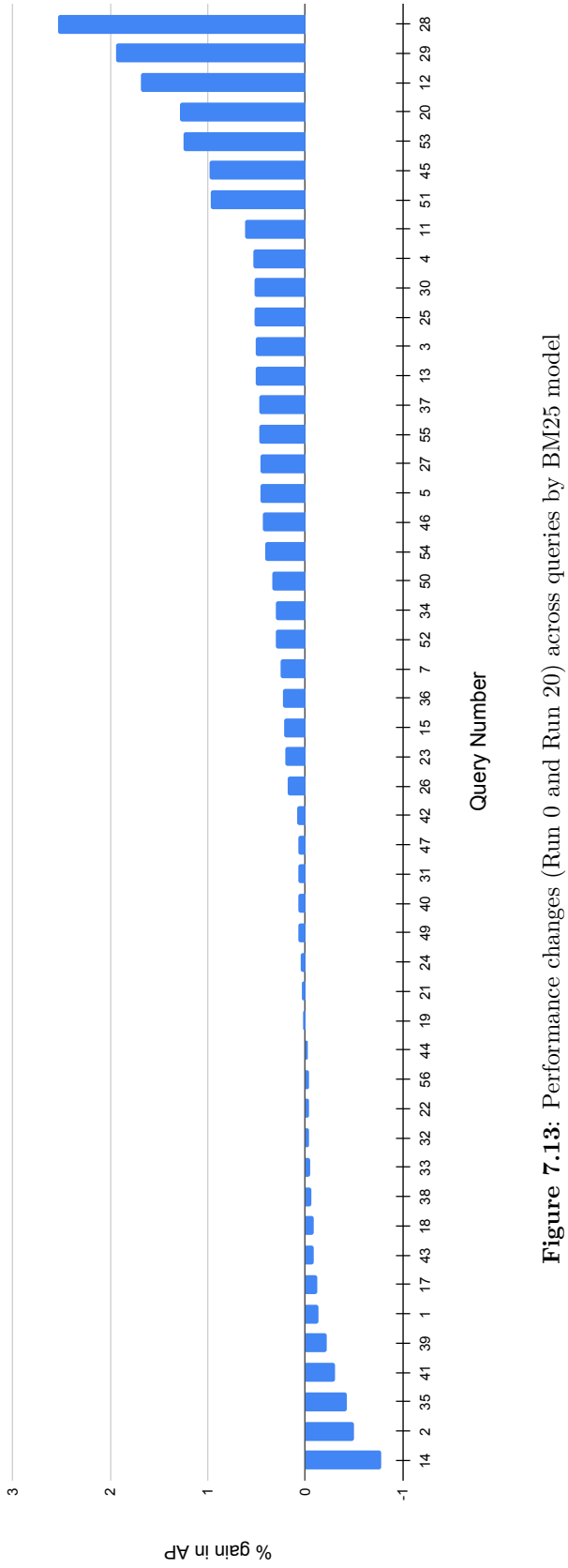

Figure 7.13: Performance changes (Run 0 and Run 20) across queries by BM25 model

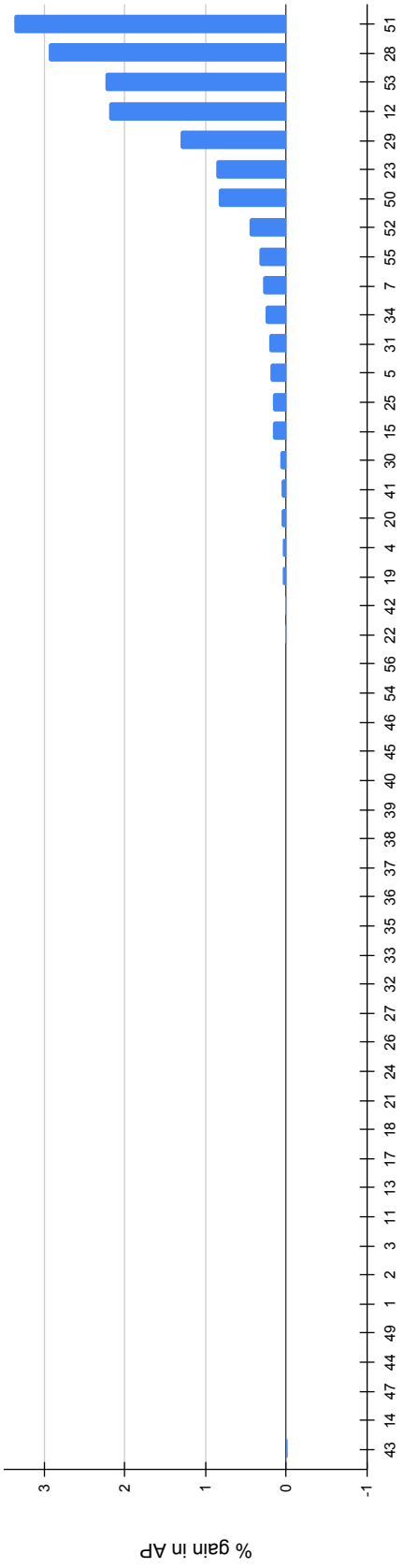

Figure 7.14: Performance changes (Run 0 and Run 15) across queries by BM25 model

7.4.3.2 Performance gain on only NE tags

Named entities are seen to play a crucial role in code-mixed social media information retrieval. Table 7.5 shows that employing Soundex and Hindex on NE words yields the best results. We also observe that in some queries, Hindex outperforms Soundex, while in others, Soundex outperforms Hindex when they are applied on NE tags (Figure 7.11). When we apply Hindex a average precision is observed to have a boost for majority of the queries (22 queries) versus reduction in a small number of queries (6 queries) while for 22 queries, the performance remains the same (Figure 7.14).

For Soundex, per query level performances have the similar trend in terms of gain (24 queries), 13 queries experiencing losses, and 13 queries remaining unchanged over the baseline (Figure 7.15).

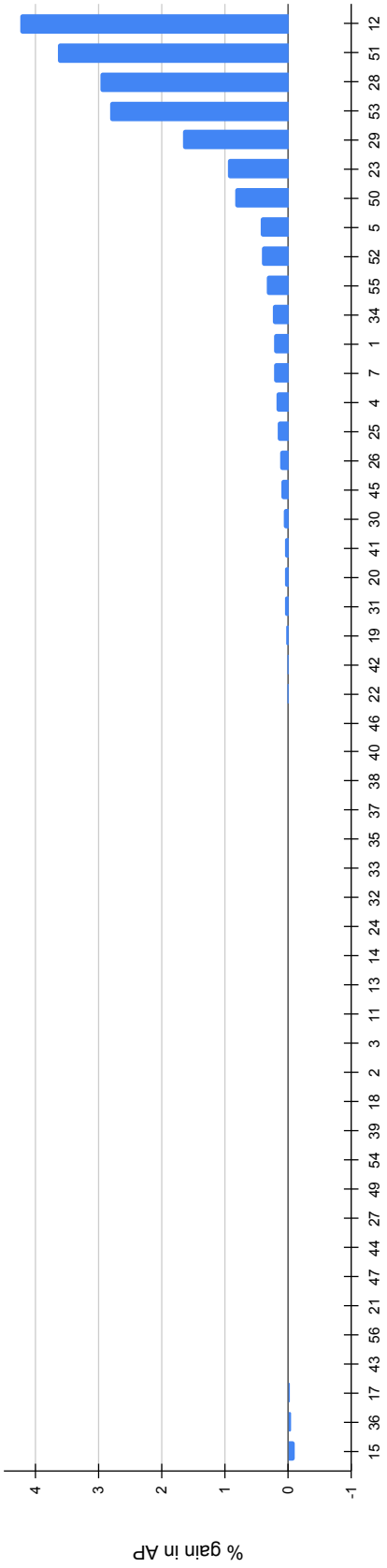

Figure 7.15: Performance changes (Run 0 and Run 8) across queries by BM25 model

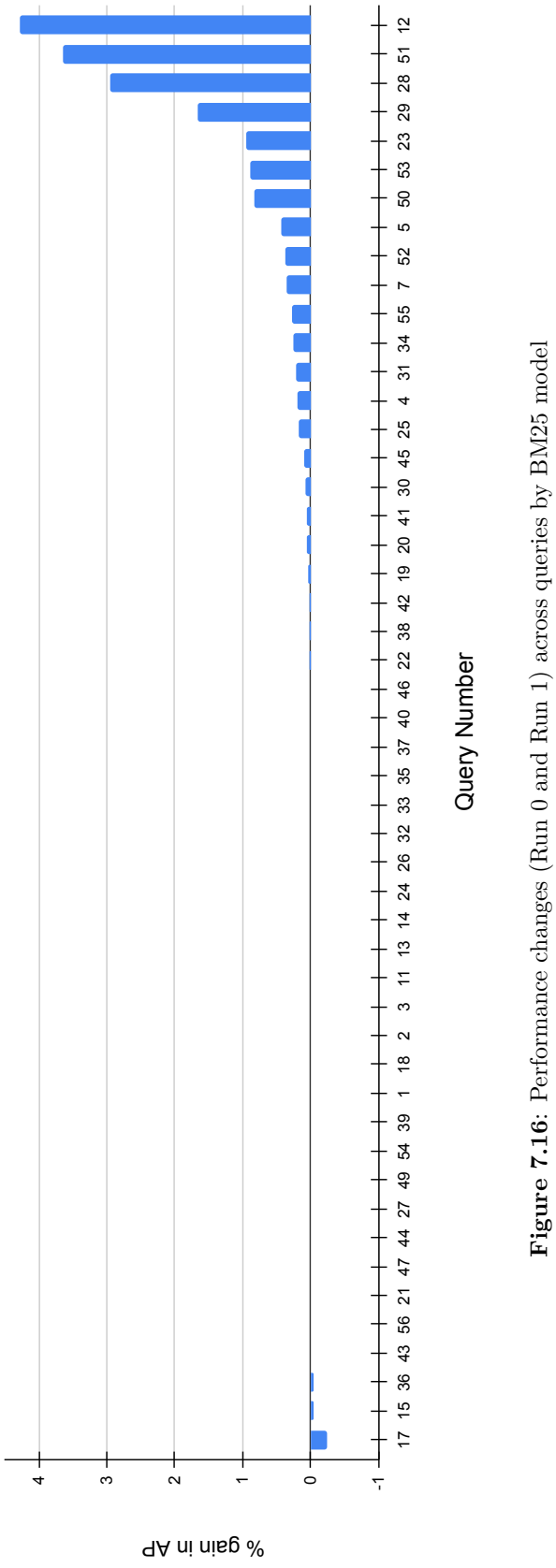

Figure 7.16: Performance changes (Run 0 and Run 1) across queries by BM25 model

Phonix on only NE (Run 1) achieves 11.01% improvement from baseline. It exhibits per-query performance similar to Soundex, with 23 queries showing gains, 14 queries experiencing losses, and 13 queries remaining unchanged (Figure 7.16).

When we use Hindex, we get a smaller number of probable words for a given named entity. On the other hand, Soundex returns a larger proportion of equivalent phonetic terms. Generating less number of phonetically similar words causes potential query mismatches. On the other hand, more phonetically identical terms may cause query drift due to their different meanings. Hence, there needs to be a fine balance between the two.

Let us illustrate the issue with three query examples: 51, 56, and 15. Run 15 (Hindex on NE) delivers the best results for query numbers 56 and 15, while Run 8 (Soundex on NE) produces the best results for query number 51 (Table 7.6). Soundex retrieves more phonetic identical terms than Hindex in query 51 for both named entities (Bangali and Olympics). Presence of their alternatives increases the number of relevant documents retrieved and produces better retrieval performance (Soundex beats Hindex here). On the other hand, in query 56, Soundex produces lot of non-synonymous alternatives (e.g., *tel, tale, tele, til* for the word *tal*, which causes query drift. But, Hindex generates the exact alternatives (Table 7.7). Similar is the case in Query 15 (Table 7.8). Here, 'Air India' is a named entity. Hence, producing alternatives of either 'air' or 'India' here are counter productive.

Query Number 51

bangali *ra eai* **olympics** *e kichu korte parbe* please give your views

English - Can Bengalis do anything in this Olympics please give your views

SOUNDEX

bengali - bangalo bengalis bangli bangla bangal bengala bangalira bangalir bangalis bangalia bansal bangaliy bangalee bengal banglai bangali bengali

olympics - olympiad olympics olympid olympiacos olympic

HINDEX

bangali - bangalir bangli bangali bangalee bangalo bangalis bengali bangal bangalia banali bangaliy

olympics - olympic olympics

Table 7.6: An example of query term expansion (Query Number 51)

Query Number 56

paka **tal** *kothay pabo ektu bolben* plz

English - Tell me please where can I get ripe Palmyra fruit

SOUNDEX

tal - tal tel tale tele til

HINDEX

tal - tal taal

Table 7.7: An example of query term expansion (Query Number 56)

Query Number 15
air india *r flight ta ki bondho kore dilo*
English - Did air india cancel the flight
SOUNDEX
air - are ari ar arr aur air
india - indiay indias indila indian india3 indra indira iindiar india indiar indiaa indie
HINDEX
air - air edr
india - india indiay indiaa indu indo india3 ind indie indi

Table 7.8: An example of query term expansion (Query Number 15)

7.4.3.3 Performance on only EN tags

Substantial performance loss is observed when applying Phonix to EN tags only (Run 2), achieving a MAP score of 0.1424. A similar performance loss is noted when Soundex is applied to only EN tags (Run 9) also, achieving a MAP score of 0.1550. However, the observations for Hindex differ from Phonix and Soundex. We observe a 7.5% performance gain when using Hindex for EN tags (Run 16), resulting in a MAP score of 0.1682. Here, average precision shows improvement for a 50% number of queries (25 queries) while experiencing a reduction of another 50% of queries (25 queries) (Figure 7.17).

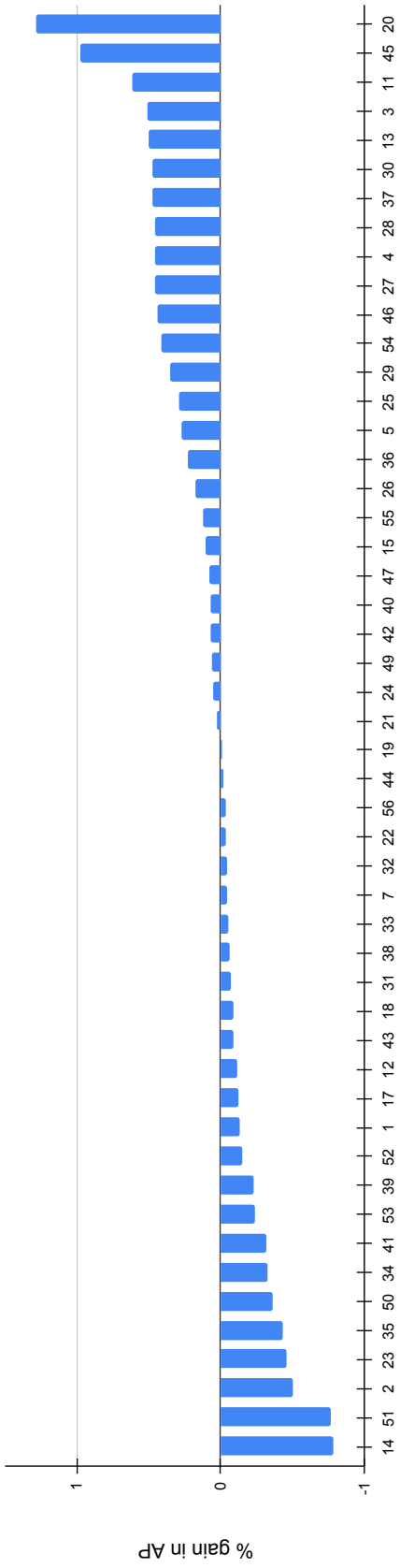

Figure 7.17: Performance changes (Run 0 and Run 16) across queries by BM25 model

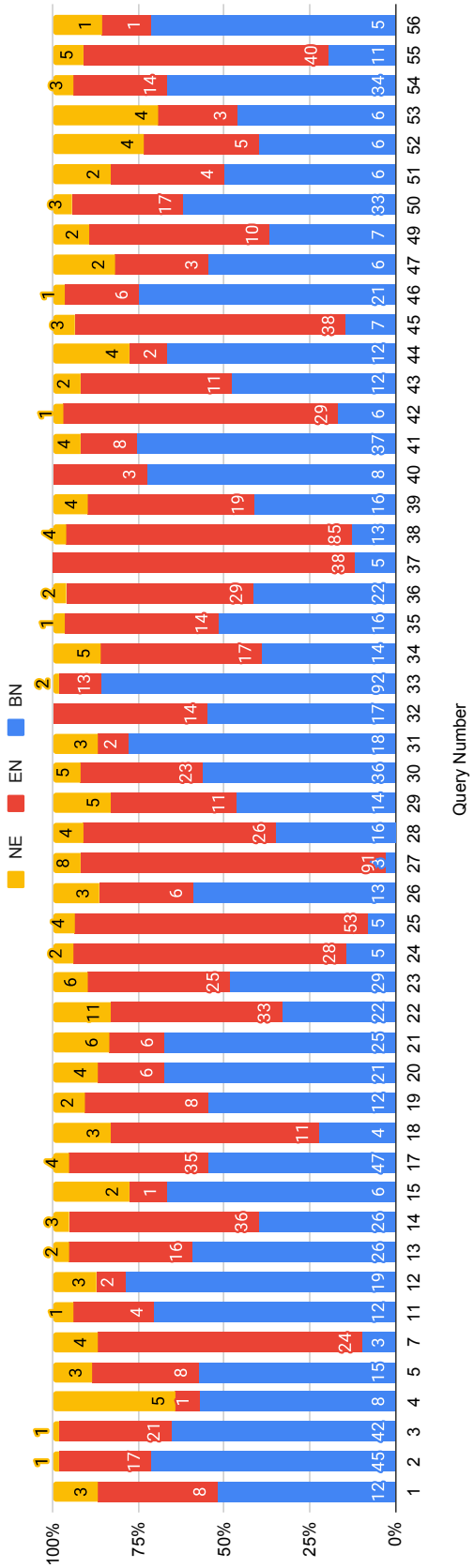

Figure 7.18: Percentage of word tags in each query

7.4.3.4 Effect of only BN tags

We tallied the number of word tags (Bengali, English, and named entity word tags) present in each query to evaluate the effectiveness of those words in retrieval performance. Figure 7.18 depicts the frequency distribution of different word tags in each query. The percentage of BN tags in almost all queries is quite high. So, when we use any phonetic encoding on BN tags, it generates many candidate words which lead to query drifts resulting in performance drop for Run 3, Run 10 and Run 17.

7.4.3.5 Performance of phonetic encoding on all (BN, EN and NE) tags

Substantial performance loss is observed when applying Phonix to BN, EN, and NE tags (Run 7), achieving a MAP score of 0.0482. A similar performance loss is noted when Soundex is applied to BN, EN, and NE tags (Run 14), achieving a MAP score of 0.1111. However, the observations for Hindex differ from Phonix and Soundex. We observe a 7.5% performance gain when using Hindex for BN, EN, and NE tags (Run 21), resulting in a MAP score of 0.1738. Here, query level average precision scores show improvement for a majority of queries (33 queries) while experiencing a reduction for small number of queries (17 queries) (Figure 7.19).

7.4.3.6 Performance of phonetic encoding on different set of combination of tags

In most cases, using phonetic encoding on a combination of two or three tags results poor performance. For instance, applying Phonix to BN and EN (Run 5) or BN and NE (Run 4) degrades performance compared to the baseline. Similar trends are observed with Soundex and Hindex. There are only two instances where performance surpasses the baseline. In general, handling spelling variations for BN tags using phonetic encodings does not work. However, it works for EN+NE setting. For example, the best performance (15% improvement) is achieved using Hindex for EN and NE (Run 20), and 8.9% improvement is seen using Soundex for EN and NE (Run 13). Using different encoding schemes for different tag-types within a single run also does not work.

7.4.3.7 Effect of human annotation over machine

We employ language identification and named entity recognition as a module. We use both machine and human annotators for word-level identification and tagging. Furthermore, we discover that only a small percentage of named entities are classified as Bengali. This misclassification leads to significant performance drop across all IR models. Run 20 outperforms all other experiments we conducted. Figure 7.20 contrasts the performance of human-annotated data vs. machine-annotated data. Furthermore, we observe that maximum queries have the same performance for both tag sets. Only a few queries perform poorly when we use machine-generated tags.

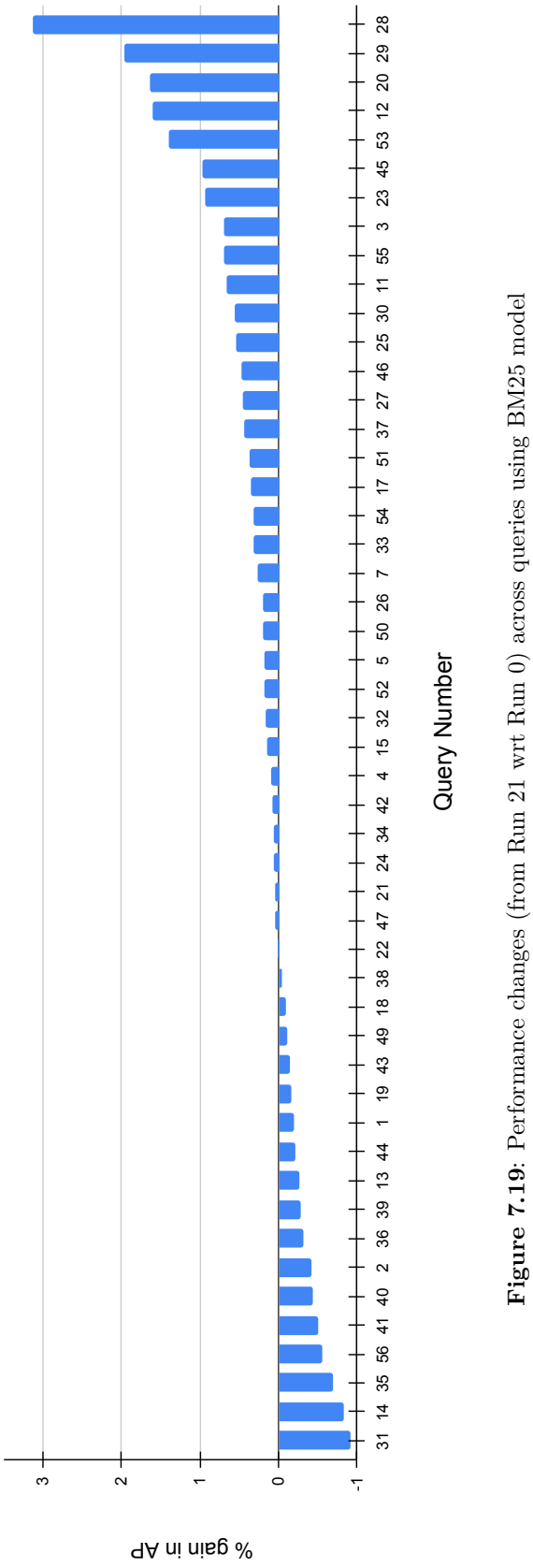

Figure 7.19: Performance changes (from Run 21 wrt Run 0) across queries using BM25 model

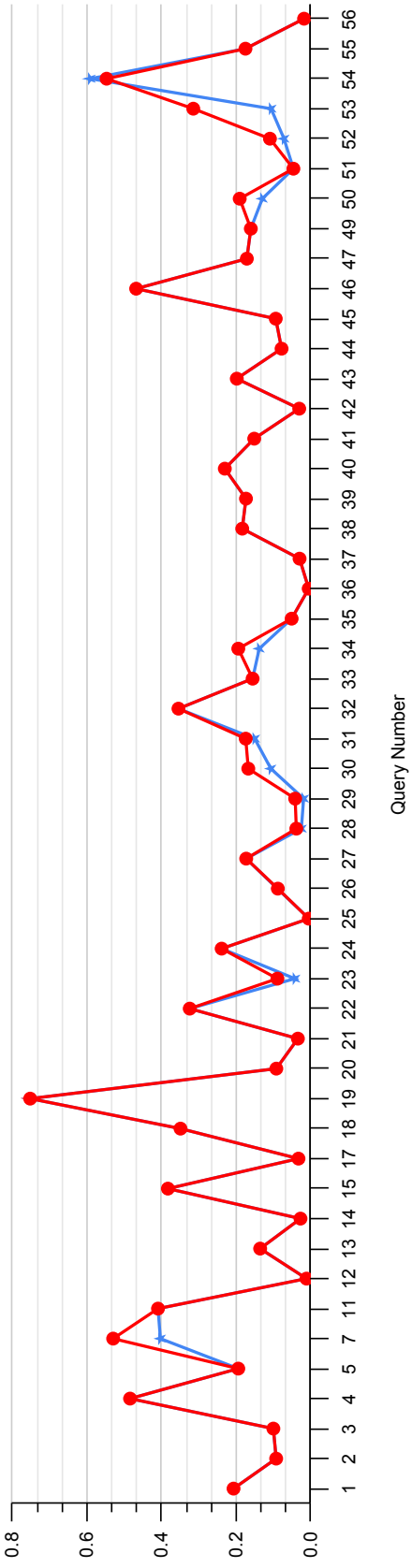

Figure 7.20: Changes in performance on human-annotated and machine-annotated data for the Run 20 experiment across queries by the In BM25 model

Let us also look at the total performance gain (Figure 7.21) for the human-annotated vis-a-vis machine-generated tags. We can see that when we use Soundex on NE tags annotated by humans, we get 14.23 % gain, whereas, on NE tags generated by machine, we get 4.14 % gain. We can see that when we use Phonix on NE tags annotated by humans, we get 11.01 % gain, whereas, on NE tags generated by machine, we get 3.65 % gain. The difference is comparatively higher than Run 21.

Figure 7.21: Performance gain and loss on both annotation data for different experiment (Run 8, Run 20 and Run 21)

In summary, Soundex performs better than other phonetic encodings for only NE tags, while Hindex shows the best performance for EN and NE tags on Bengali-English code-mixed social media data.

7.4.4 Error Analysis

In our experiments, we come across some errors where our models could not perform well.

- Query and document term mismatch is still a problem due to the multilingual environment. For example, if there is any word like 'sweet' in query and a similar term in Bengali, Misti (মিষ্টি) is written in Roman script in documents. The spelling variation technique using phonetic encoding could not solve the multilingual word mismatch between queries and documents.
- In our experiments, the queries are long (compared to traditional monolingual IR settings), and we do not use any stopword and stemming approaches for the code-mix queries or documents. This makes the search space very lousy and leads to query drift

situation.

7.5 CMIR: A Study on Effects of Stop words Removal

Stop words are a set of frequently used words that do not carry major information but are employed to complete sentences according to grammar rules and to provide a holistic sense for an intended message. Their role is only supportive in natural language tasks, but they take good amount of space. Owing to their extremely high frequency, they can potentially reduce the importance of less frequent but highly informative terms and seriously affect the retrieval performance. Identifying stop words and eliminating them thus can improve the indexing process's efficiency by reducing the index size 30 to 50 %, which makes search and retrieval faster by avoiding a lot of unnecessary text processing. Corpus-based identification of stop words is done by looking at the collection frequency of different words. Words that occur frequently in the entire corpus across almost all documents are identified as stop words. This technique is language-independent and does not need knowledge of the language. Non-corpus-based identification techniques, on the other hand, are predetermined by experts or based on a fixed list of stop words that are common in a given language. Examples of stop words identified through a non-corpus based technique include "the," "and," "of," and "to", etc. in English.

Removing stop words for an information retrieval (IR) task is typically done to boost both time and space efficiency as well as retrieval effectiveness of an IR system. This process is employed in almost all languages and is not limited to resourceful languages like English. Fox [129], was the first to develop a list of stop words based on English words used in general literature. In the following three decades, several research works on stop words in various languages have been conducted. Each language has its own morphological variety and linguistic characteristics. Thus, there is no general set of rules which govern the creation of these lists of stop words. Also, every list of stop words improves the system's performance in a different way. Evidence in the literature shows that several stop words lists are produced using corpus-based technique and non-corpus based or domain-specific ones.

Typically, search queries in various search engines or social media platforms are not code-mixed. In other words, users do not use code-mixing while creating queries in traditional search engines. [103]. Also, search queries typically contain a terse set of keywords. In contrast, the queries posed in social media are often code-mixed with long and verbose paragraphs full of stop words that may lead to query dilution and/or query drift and leads to poor retrieval performance.

Other than two significant challenges in code-mixed text processing that were discussed like language identification (LID) and spelling variations, the third one is stop words identification. The challenge in stop words identification here in code mixed text is the presence of of words from more than one language. Hence, depending on the languages present, stop words of multiple languages need to be considered. But sometimes a single word may actually belong to multiple languages present in the data and carry different meaning altogether. For example, the word "to" in English is also used in Bengali "to" (তো) which means "so what", and the word "do" in English is also used in Hindi "do" (दो) which means "two" in number or "give" based on the context of its use. While in one language, a word is a stop word, in another it is not. This overlap of dictionary throws a challenge when identifying stop words. Moreover, the concept of stop words is fluid in social media as social media text is often short and cryptic. Removing stop words in short text can sometimes be detrimental to understanding the context of the text. For example, strings like 'to be or not to be' or 'have and have-nots' contain all stop words that, when removed, make huge loss of information.

Thus, a list of stop words based on a single language is not sufficient, we need a domain-specific, code-mixed list for stop words. However, the questions remain like - can it be applied to deal with new scenarios, where we have multiple languages and multiple scripts with uncontrolled language constructs? What modifications do we need to incorporate in the IR setting, or what pre-processing and post-processing steps are needed to handle these?

We use a wide range of techniques to propose an identification of stop words for code-mixed retrieval. A couple of experiments (Runs) are carried out to investigate the impact of stop words with a set of queries. However, related to broad-level RQ-2 mentioned in Section 7.2, there are a few sub-questions as follows.

- *RQ-2.4: Are there any performance differences between non-corpus based and corpus-based techniques for stop words removal in Code-Mixed IR?*
- *RQ-2.5: Can corpus-based stop words removal improve Code-Mixed IR effectiveness? If yes, to what extent?*
- *RQ-2.6: Which stop words list gives the best result among the different corpus-based stop words?*

7.5.1 Experimental Setup

For our experiments, we use the PyTerrier, a declarative platform for information retrieval experiments in Python. It uses the Java-based Terrier (Terabyte Retriever) [4] IR framework, developed by University of Glasgow. When Terrier indexes a collection, a list of stop words is used. All experiments are carried out on a Linux based laptop equipped with 16 GB of RAM and an Ryzen 7 processor with 16 cores.

1. *RQ-2.4: Are there any performance differences between non-corpus based and corpus-based techniques for stop words removal in Code-Mixed IR?*

   Here, we examine the effects of the non-corpus based and corpus-based techniques for stop words removal on retrieval effectiveness. First, we set a baseline (Run 0), where we do not use any list of stop words during indexing. The non-corpus based list of stop words are obtained from the Terrier and SMART systems. However, corpus-based stop words list are created using a variety of techniques.

2. *RQ-2.5: Can corpus-based stop words removal improve Code-Mixed IR effectiveness? If yes, to what extent?*

   To generate corpus-based stop words list we use different algorithms, which are mentioned below.

   Term Frequency (TF) based: TF is referred as the frequency of a given term in the entire corpus. The number of times a term appears throughout any corpus denoted as TF. Frequency of each term in the corpus is collected and stop words list is generated based on a threshold frequency. All the terms having term frequency greater than a

[4] `http://terrier.org/`

threshold value are included in the stop words list. Different threshold values are tried and an optimum value is chosen that yields the best results in terms of MAP.

Normalised TF based: It is a corpus based methodology where normalised term frequency is calculated instead of raw TF using the given formula:

$$TF_{Norm} = -log\left(\frac{TF}{v}\right) \tag{7.1}$$

where, v is the number of unique words in the corpus.

Document Frequency (DF) based: The DF of a term is the number of documents a term appears in. DF of each term in the corpus is collected, and a stop words list is crated based on a threshold value that includes all terms with DF larger than the threshold. Different threshold values are tried, and the threshold value producing the best MAP is chosen.

Inverse DF (IDF) based: The problem with TF is that the terms with high frequency across are often the least important. On the other hand, rarely occurring terms are more informative and therefore should be given more importance. Also, the documents having such terms should get high importance. IDF is a way of reducing the weight of terms that appear frequently within a corpus (high collection frequency) and increasing the importance of rare terms that occur only in few documents. IDF of a term t is calculated by the formula:

$$IDF(t) = log\left(\frac{N}{n_t}\right) \tag{7.2}$$

where, N is the total number of documents in the corpus and n_t is the number of documents containing term t.

TF-IDF based: TF-IDF is the product of term frequency and inverse document frequency for a term and captures the importance of a given term. We compute TF-IDF of each term and all the terms having TF-IDF value less than a threshold are included in the stop words list.

3. *RQ-2.6: Which stop words list gives the best result among the different corpus-based*

stop words?

In our corpus, the distribution of Bengali and English tokens are maximum. So, we divide the creation of corpus-based stop words list into two ways. One is language-independent, where we consider the whole corpus without regard to any languages and find the threshold value using different methods (TF, NTF, DF, IDF, TF-IDF) that we mention in previous research question (RQ-2.5). A grid search algorithm (Algorithm 7.2) is used.

The algorithm (Algorithm 7.2) is designed to find the optimal threshold values for a set of words that maximize the Mean Average Precision (MAP) score using various techniques (TF, NTF, DF, IDF, TF-IDF). It takes a set of words (words) and a specified technique as inputs, and outputs the optimal threshold values θ_{values}. Initially, it calculates values for the words using the chosen technique and sorts these values. The algorithm then sets the minimum and maximum values for the words as the starting and finishing points. Using a grid search approach, it iteratively explores the search space, calculates MAP scores for each threshold value, and stores these scores. The optimal threshold is determined using the arg max function, which identifies the value that maximizes the MAP score. The search range is refined around this optimal value, and the step size is halved in each iteration. The process continues until the step size falls below a specified threshold (0.2), at which point the optimal threshold value θ_{values} is returned. For three of the techniques (TF, NTF, DF), words having scores higher than the thresholds are considered as stop words. For the other two techniques (IDF and TF-IDF), words having values less than the thresholds are considered stop words. The other is the language-dependent one where we separate tokens based on language tags, and then find out the thresholds separately for each language 7.3 which is similar to the previous algorithm but text into account both the languages Bengali and English. Finally, two stop word lists are combined. We already have words and its corresponding language tags from Section 7.4.1, where we use an in-house transformer-based language identifier as described in Chapter 4, supplemented by human annotation.

In our experiments, we apply two non-corpus based stop words list (Run 1 and Run 2).

Algorithm 7.2: Grid search algorithm to find the best threshold value for words

Input: $words = \{w_1, w_2, \ldots, w_n\}, technique = \{\text{TF}, \text{NTF}, \text{DF}, \text{IDF}, \text{TF-IDF}\}$
Output: θ_{values}

function `grid_search_words`($words, technique$)
  $values = \{\}$;
  $values \leftarrow Calculate(words, technique)$; `// Term score for` $words$ `based on the` $technique$
  $values \leftarrow Sort(values)$; `// numeric sort in increasing order`
  $values_s \leftarrow values[0]$; `// Minimum value for words`
  $values_f \leftarrow values[n-1]$; `// Maximum value for words`
  $step \leftarrow 1$;
  while *true* **do**
    $\theta_{values} \leftarrow values_s$;
    $MAP_{\text{values}} \leftarrow \{\}$;
    while $\theta_{values} < values_f$ **do**
      $MAP_{\theta_{values}} \leftarrow Calculate\_MAP(\theta_{values})$;
      $MAP_{\text{values}} \leftarrow MAP_{\text{values}} \cup \{MAP_{\theta_{values}}\}$;
      $\theta_{values} \leftarrow \theta_{values} + step$;
    $[values_t] \leftarrow \arg\max_{\theta_{values}}(MAP_{\text{values}})$;
    $values_s \leftarrow values_t - 1$;
    $values_f \leftarrow values_t + 1$;
    $step \leftarrow step/2$;
    if $step < 0.2$ **then**
      break;
  return θ_{values};

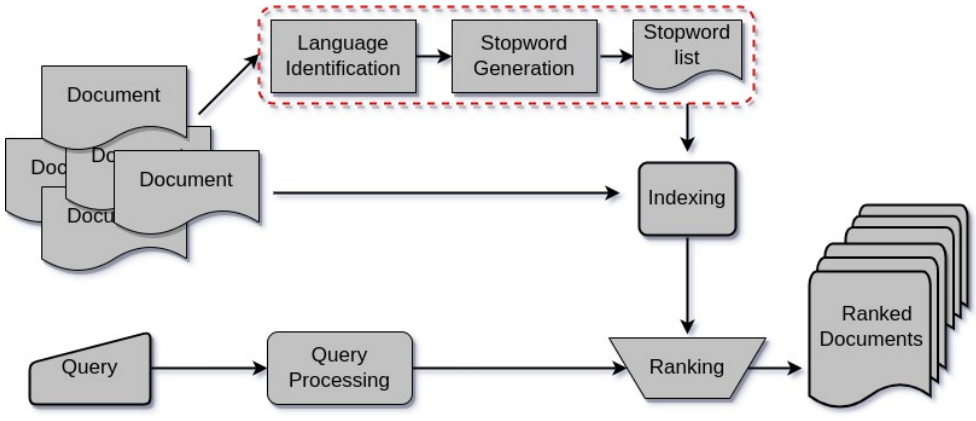

Figure 7.22: Process flow for retrieval

For corpus-based stop words list, we use two separate sets of experiments with language-independent stop words (Run 3 to Run 7) and language-dependent stop words (Run 8 to Run

Algorithm 7.3: Grid search algorithm to find the best threshold value for both languages

Input: $bn = \{bn_1, bn_2, \ldots, bn_n\}, en = \{en_1, en_2, \ldots, en_m\}, technique = \{\text{TF}, \text{NTF}, \text{DF}, \text{IDF}, \text{TF-IDF}\}$
Output: θ_B, θ_E

function `grid_search`$(bn, en, technique)$
  $BN_v = \{\}$; $EN_v = \{\}$;
  $BN_v \leftarrow Calculate(bn, technique)$; `// Bengali term score for` bn `based on the` $technique$
  $EN_v \leftarrow Calculate(en, technique)$; `// English term score for` en `based on the` $technique$
  $BN_v \leftarrow Sort(BN_v)$; `// numeric sort in increasing order`
  $EN_v \leftarrow Sort(EN_v)$; `// numeric sort in increasing order`
  $BN_s \leftarrow BN_v[0]$; `// Minimum value for Bengali`
  $BN_f \leftarrow BN_v[n-1]$; `// Maximum value for Bengali`
  $EN_s \leftarrow EN_v[0]$; `// Minimum value for English`
  $EN_f \leftarrow EN_v[m-1]$; `// Maximum value for English`
  $step \leftarrow 1$;
  while *true* **do**
    $\theta_B \leftarrow BN_s$;
    $\theta_E \leftarrow EN_s$;
    $MAP_{\text{values}} \leftarrow \{\}$;
    while $\theta_E < EN_f$ **do**
      while $\theta_B < BN_f$ **do**
        $MAP_{\theta_B,\theta_E} \leftarrow Calculate\_MAP(\theta_B, \theta_E)$;
        $MAP_{\text{values}} \leftarrow MAP_{\text{values}} \cup \{MAP_{\theta_B,\theta_E}\}$;
        $\theta_B \leftarrow \theta_B + step$;
      $\theta_E \leftarrow \theta_E + step$;
    $[BN_t, EN_t] \leftarrow \arg\max_{\theta_B,\theta_E}(MAP_{\text{values}})$;
    $BN_s \leftarrow BN_t - 1$;
    $BN_f \leftarrow BN_t + 1$;
    $EN_s \leftarrow EN_t - 1$;
    $EN_f \leftarrow EN_t + 1$;
    $step \leftarrow step/2$;
    if $step < 0.2$ **then**
      break;
  return θ_B, θ_E;

12). Four different term-weight based retrieval models, namely BM25, TF-IDF, PL2, and InL2, are tried for each combination as indicated above (Table 7.9). We do not apply any stemmers in the above runs.

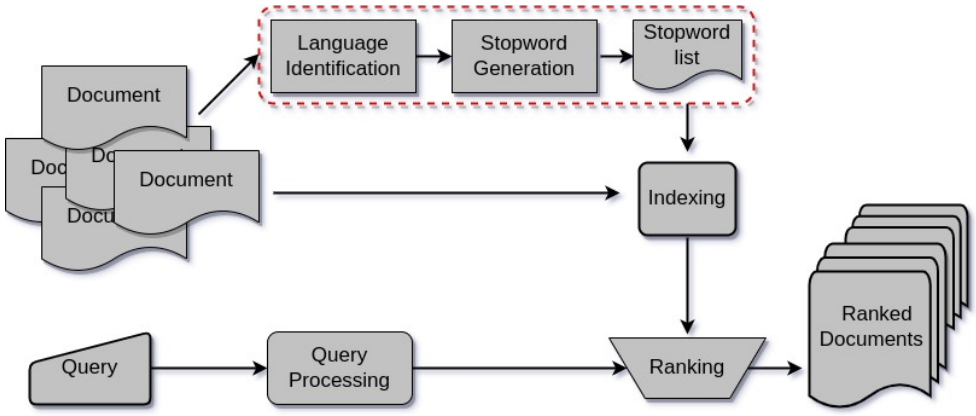

Figure 7.23: Process flow for retrieval

Table 7.9: All experiment setup

# Experiment	Experiment details
Run 0	No stop words and no stemming. Refer as Baseline
Run 1	Use stop words provided by the Terrier system and no stemming
Run 2	Use English stop words list, taken from the SMART system and no stemming
Run 3	Use corpus-based language independent stop words list based on TF and no stemming
Run 4	Use corpus-based language independent stop words list based on NTF and no stemming
Run 5	Use corpus-based language independent stop words list based on DF and no stemming
Run 6	Use corpus-based language independent stop words list based on IDF and no stemming
Run 7	Use corpus-based language independent stop words list based on TF-IDF and no stemming
Run 8	Use corpus-based language-dependent stop words list for both languages (BN and EN) based on TF and no stemming
Run 9	Use corpus-based language-dependent stop words list for both languages (BN and EN) based on NTF and no stemming
Run 10	Use corpus-based language-dependent stop words list for both languages (BN and EN) based on DF and no stemming
Run 11	Use corpus-based language-dependent stop words list for both languages (BN and EN) based on IDF and no stemming
Run 12	Use corpus-based language-dependent stop words list for both languages (BN and EN) based on TF-IDF and no stemming

7.5.2 Results

Extensive experiments are conducted to seek answers to the stated RQs, mentioned in Section 7.5. For each query, the models produce a ranked list of documents in decreasing order of retrieval status values as computed by the model. Such ranked lists are evaluated against ground-truths or relevance judgments (a file of query-document pairs called *qrels*) for each query. Retrieval effectiveness is measured by different metrics (defined in Chapter 2).

Evaluation scores on different retrieval models are shown in Table 7.10. The best performance exhibited by a retrieval model is shown in bold.

Experiment	IR Model	MAP	Recip-rank	nDCG	P@5	P@10
Run 0	TF_IDF	0.1178	0.5107	0.3310	0.272	0.196
	BM25	0.1215	0.5208	0.3384	0.272	0.204
	PL2	0.1119	0.5142	0.3228	0.24	0.18
	InL2	**0.1270**	**0.5438**	**0.3466**	**0.276**	**0.216**
Run 1	TF_IDF	0.1415	0.5800	0.3789	0.296	0.24
	BM25	0.1464	0.5933	0.3877	0.308	0.244
	PL2	0.1370	0.5800	0.3685	0.276	0.224
	InL2	**0.1522**	**0.6112**	**0.3943**	**0.328**	**0.246**
Run 2	TF_IDF	0.1452	0.5840	0.3867	0.296	0.244
	BM25	0.1504	0.5900	0.3958	0.308	**0.25**
	PL2	0.1405	0.5888	0.3789	0.288	0.23
	InL2	**0.1549**	**0.6044**	**0.3995**	**0.328**	0.248
Run 3	TF_IDF	0.1576	0.6440	0.4064	0.328	0.266
	BM25	0.1598	**0.6581**	0.4098	0.332	0.262
	PL2	0.1515	0.6319	0.3958	0.316	0.248
	InL2	**0.1674**	0.6347	**0.4177**	**0.34**	**0.28**
Run 4	TF_IDF	0.1576	0.6440	0.4064	0.328	0.266
	BM25	0.1598	**0.6581**	0.4098	0.332	0.262
	PL2	0.1515	0.6319	0.3958	0.316	0.248
	InL2	**0.1674**	0.6347	**0.4177**	**0.34**	**0.28**
Run 5	TF_IDF	0.1590	0.6505	0.4078	0.324	0.268
	BM25	0.1609	**0.6590**	0.4101	0.324	0.27
	PL2	0.1520	0.6354	0.3970	0.308	0.252
	InL2	**0.1683**	0.6379	**0.4185**	**0.336**	**0.282**
Run 6	TF_IDF	0.1586	0.6409	0.4066	0.32	0.262
	BM25	0.1607	**0.6553**	0.4100	0.332	0.264
	PL2	0.1517	0.6316	0.3962	0.312	0.248
	InL2	**0.1680**	0.6320	**0.4182**	**0.34**	**0.28**

Experiment	IR Model	MAP	Recip-rank	nDCG	P@5	P@10
Run 7	TF_IDF	0.1592	0.6440	0.4077	0.328	0.262
	BM25	0.1612	**0.6578**	0.4107	0.336	0.264
	PL2	0.1522	0.6319	0.3967	0.316	0.248
	InL2	**0.1682**	0.6347	**0.4182**	**0.34**	**0.28**
Run 8	TF_IDF	0.1667	0.6039	0.4241	0.372	0.282
	BM25	0.1690	0.6151	0.4268	0.376	0.286
	PL2	0.1588	0.5861	0.4145	0.36	0.27
	InL2	**0.1783**	**0.6629**	**0.4400**	**0.384**	**0.29**
Run 9	TF_IDF	0.1660	0.6322	0.4230	0.356	0.268
	BM25	0.1678	0.6439	0.4257	0.36	0.27
	PL2	0.1554	0.6032	0.4103	0.344	0.252
	InL2	**0.1746**	**0.6774**	**0.4367**	**0.372**	**0.276**
Run 10	TF_IDF	0.1663	0.6296	0.4295	0.352	0.278
	BM25	0.1693	0.6469	0.4334	0.368	0.274
	PL2	0.1588	0.6140	0.4202	0.344	0.266
	InL2	**0.1765**	**0.6481**	**0.4429**	**0.372**	**0.288**
Run 11	TF_IDF	0.1674	0.6552	0.4184	0.364	0.272
	BM25	0.1705	0.6726	0.4226	0.364	0.276
	PL2	0.1606	0.6490	0.4092	0.356	0.258
	InL2	**0.1777**	**0.6829**	**0.4321**	**0.4**	**0.284**
Run 12	TF_IDF	0.1699	0.5961	0.4285	0.352	0.274
	BM25	0.1721	0.6027	0.4314	0.368	0.278
	PL2	0.1617	0.5697	0.4186	0.34	0.258
	InL2	**0.1800**	**0.6169**	**0.4405**	**0.384**	**0.290**

Table 7.10: The results of retrieval effectiveness measured by MAP, Recip-rank, nDCG, P@5, and P@10 for all the experimental setups.

- *RQ-2.4: Are there any performance differences between non-corpus based and corpus-based stop words removal in Code-Mixed IR?*

  Yes, there is a performance difference between non-corpus based and corpus-based stop

words removal. We get substantial performance gain in terms of MAP after corpus-based stop words removal over non-corpus based stopword removal (Run 3 to Run 12).

- *RQ-2.5: Can corpus-based stop words removal improve Code-Mixed IR retrieval effectiveness? If yes, to what extent?*

  Yes, all runs using corpus-based stop words removal techniques improve retrieval effectiveness over the baseline (Run 0). In corpus-based techniques, we use language-independent and language-dependent stop words list. While using language-independent stop words list based on DF scores (Run 5), we have the best MAP score of 0.1683. In Run 5, we observe a 32.4% performance gain over the baseline (Run 0).

- *RQ-2.6: Which stop words list gives the best result among the different corpus-based stop words?*

  While considering language-independent strategies, among the five stop words removal techniques, DF-based one demonstrates the best performance across the IR models used.

  In the language-dependent strategies, language tags are used to separate the tokens in two language groups and then TF, normalized TF, DF, IDF, and TF-IDF scores are applied separately (Algorithm 7.3). Figure 7.24 shows how we got the threshold TF-IDF value for Bengali and English languages (Run 12) that outperforms all runs (best MAP score 0.1800).

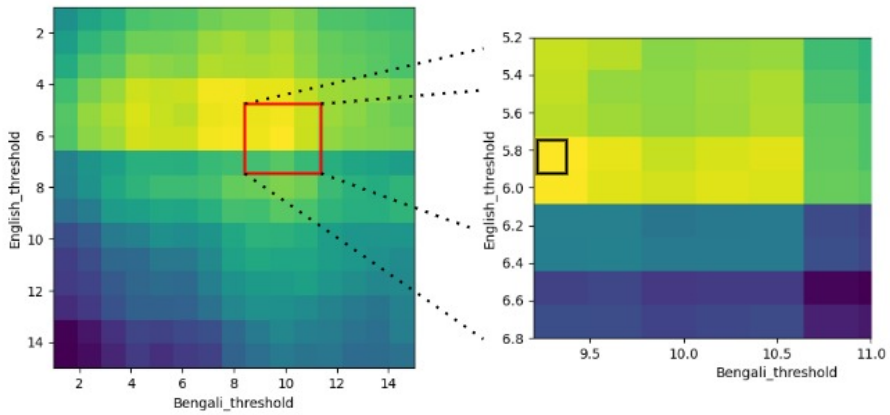

Figure 7.24: Find the best threshold value for corpus-based language-dependent stop words list

7.5.3 Discussion

In this section, we attempt to discuss the retrieval effectiveness at the query level, with some insights gained. The performance analysis here are based on Table 7.10.

7.5.3.1 Best technique to generate stop words

Among the all techniques to generate corpus-based stop words, TF-IDF based technique (Run 12) outperforms others yielding a 41.79% performance boost in terms of MAP over the baseline (Run 0). The performance difference between TF (Run 8), Normalised TF (Run 9), DF (Run 10), IDF (Run 11), and TF-IDF (Run 12) are very less (Figure 7.25).

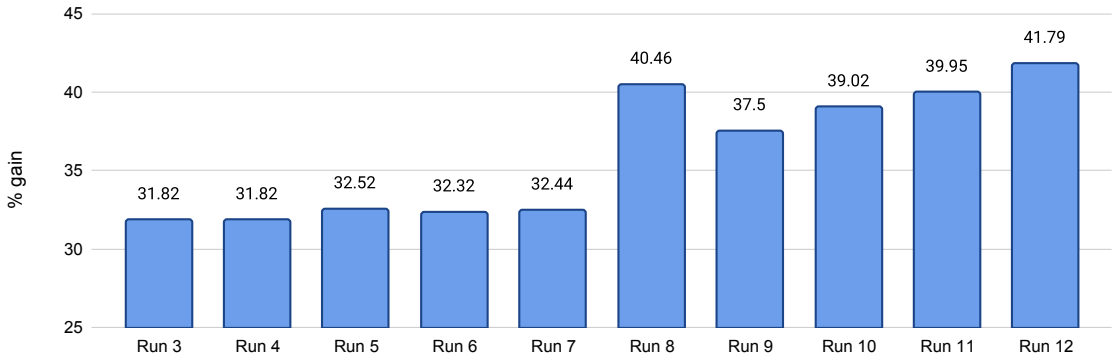

Figure 7.25: Performance gain from Run 3 to Run 12 wrt Run 0

7.5.3.2 Effect of corpus-based stop words over baseline

Our dataset contains Bengali and English code-mixed data, where Bengali words written in its transliterated form. As per our knowledge, there is no stop words available for BN-EN code-mixed data. So we choose not to use any stop words during indexing and check the retrieval effectiveness called baseline (Run 0). In comparison to the baseline, we observe a 41.79% performance gain when using corpus-based stop words (Run 12), resulting in a MAP score of 0.1800. Here, query level average precision scores show improvement for a majority of queries (35 queries) while experiencing a reduction for small number of queries (13 queries) and 2 queries remaining unchanged (Figure 7.26).

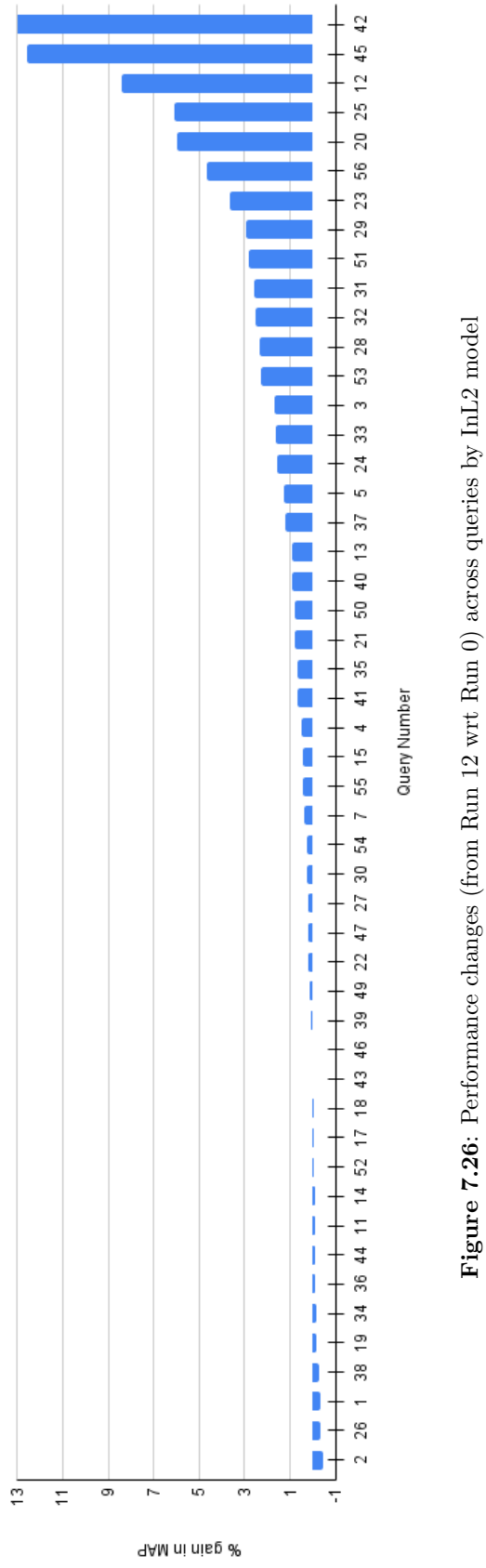

Figure 7.26: Performance changes (from Run 12 wrt Run 0) across queries by InL2 model

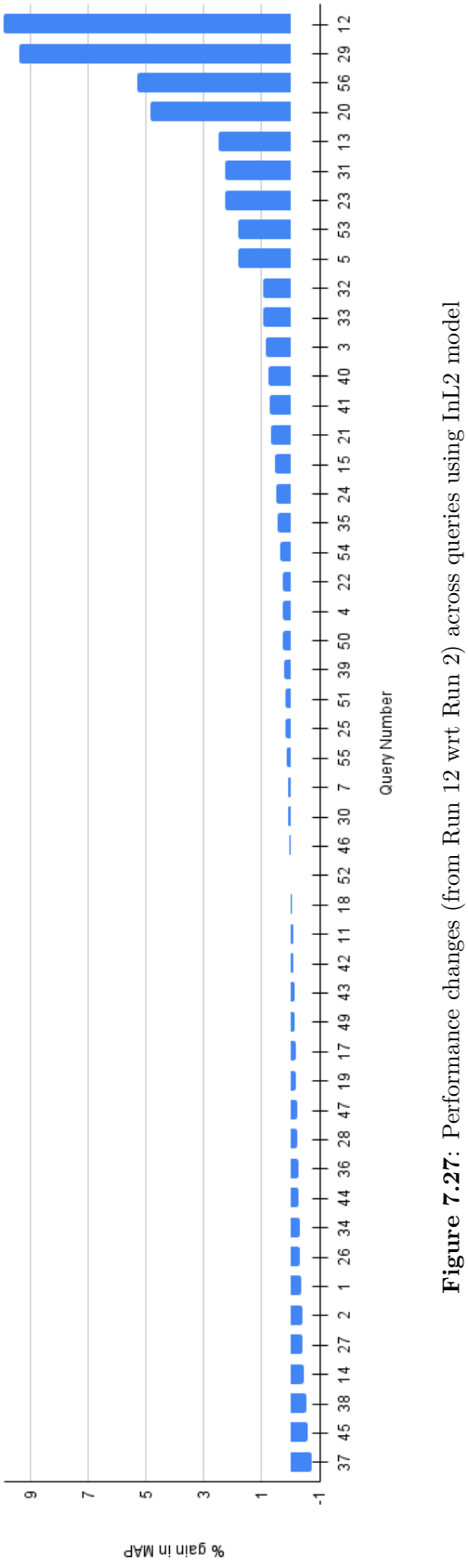

Figure 7.27: Performance changes (from Run 12 wrt Run 2) across queries using InL2 model

7.5.3.3 Effect of corpus-based stop words over non-corpus based

We use two non-corpus based stop words list from Terrier and SMART. Stop words from SMART improve the performance over Terrier stop words list, resulting in a MAP score of 0.1549. In comparison to non-corpus based stop words list, we observe a 16.25% performance gain when using corpus-based stop words (Run 12), resulting in a MAP score of 0.1800. Here, query level average precision scores show improvement for a majority of queries (30 queries) while experiencing a reduction for small number of queries (20 queries) (Figure 7.27)

Overall using both corpus-based and non-corpus based stop words list improves MAP score over the baseline.

7.6 Summary of this Chapter

Code-mixed language, primarily a spoken language phenomenon, has increasingly become a textual phenomenon due to the proliferation of the internet and social media. This shift presents significant challenges for the text processing research community, particularly for information retrieval (IR) specialists focused on searching code-mixed text using code-mixed queries. In this chapter, we investigate various contemporary phonetic algorithms for addressing spelling variations and identify constraints where these algorithms fall short. Due to the lack of a suitable dataset, our study focuses exclusively on Bengali-English code-mixed data that we created with 50 queries and 107900 documents. We conduct a series of experiments using five different IR models on three distinct phonetic encoding schemes. Our experiments on code-mixed Bengali-English data show a 15% performance gain over the baseline. Soundex outperforms other phonetic encoding for NE tags, while Hindex demonstrates the best performance for EN and NE tags on Bengali-English code-mixed social media data. We do not apply any stop words list or any stemmer for this set of experiments.

However, the next set of experiments removing stop words proves to be a beneficial preprocessing step in code-mixed IR. Empirical evaluations report a significant improvement in mean average precision (MAP) when stop words are removed compared to cases where they are not. In a code-mixed setting, two approaches have been explored for generating

stop word lists based on language information. Empirical results indicate that corpus-based stop word removal significantly improves MAP values compared to non-corpus-based stop word removal by 16.25%. Document frequency (DF)-based stop word generation is the most effective method when language information for words in a corpus is unavailable. However, when such information is available, term frequency-inverse document frequency (TF-IDF)-based stop word generation proves to be the most effective. Following stop word removal, there is a 41.7% improvement in performance compared to the baseline.