Chapter 2

Background

In this chapter, we introduce foundational concepts and terminologies aimed for better comprehensibility of the thesis. We cover different types of information retrieval (IR), IR models, IR evaluation metrics including the metrics related to code mixing (CM) pertinent to this study.

2.1 What are Languages and Scripts?

Numerous definitions of languages have been proposed throughout history. Henry Sweet, a renowned English phonetician and language scholar, articulated that "A language is the expression of ideas by means of speech-sounds combined into words. Words are combined into sentences, this combination answering to that of ideas into thoughts." In a simplified explanation, a language can be understood as a structured system of communication characterized by a grammar and a vocabulary. Examples of languages include Hindi, Bengali, English, Telugu, Tamil, Kannada, Sanskrit, German etc.

A script, on the other hand, constitutes a writing system comprising a specific set of symbols. These symbols, known as letters, typically correspond to spoken phonemes. Examples of scripts include Devanagari, Bengali, Roman, Telugu, Tamil, Malayalam etc. It is noteworthy that a language typically precedes the development of a script,

with scripts emerging when a speech community feels the necessity to record their spoken language. Ideally, each language should possess its own unique script capable of accurately representing its distinct sounds. However, this ideal is often hindered by the arbitrary relationship between scripts and languages, which may vary from a one-to-one correspondence to a one-to-many relationship. Figure 2.1 illustrates the complex mapping between languages and scripts.

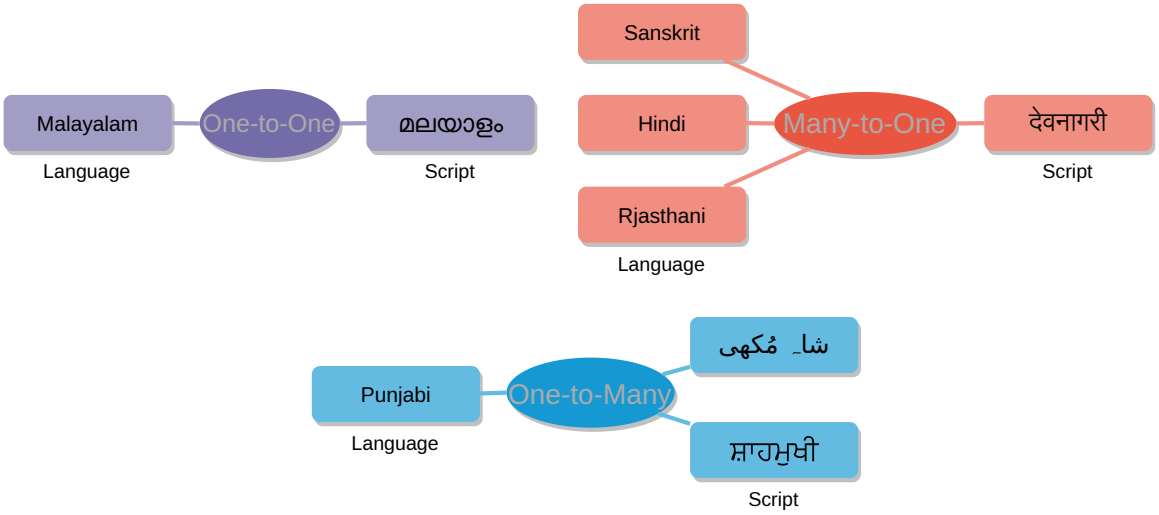

Figure 2.1: Mapping between language and script

With the exponential use of social media text in our communication and increasing prevalence of multilingualism, traditional one-to-one correspondence between language and script is fast becoming obsolete. People increasingly write a language using a non-native script (mostly Roman due to technical convenience) and thus there exists no assurance that comprehension of a language can be facilitated solely through its corresponding script, especially as far as social media text is concerned. The association between language and script is nowadays characterized by arbitrariness, permitting any language to be transcribed into any script, and conversely, enabling the representation of all languages using a single script. This phenomenon has thus spawned the concepts of transliteration and code-mixing.

2.2 Transliteration and Code-Mixing

Transliteration is the process of phonetically converting words of a language into a foreign or non-native script. It has two primary forms: forward transliteration and backward transliteration. In forward transliteration, a language is written in a non-native script. When the text written using non-native script is brought back to the native script, it is called backward transliteration (see Figure 2.2).

Code-Mixing denotes the mixing of lexical elements and grammatical structures from multiple languages within a single sentence. Example 1 illustrates a Code-Mixing sentence, wherein the initial segment is written in Bengali language and script, while the subsequent portion is composed in English language and Roman script. Example 2 similarly showcases a Code-Mixing sentence, where the first segment is in Hindi language (*Kal subha bazaar mai*) but transliterated, followed by English language text presented in Roman script.

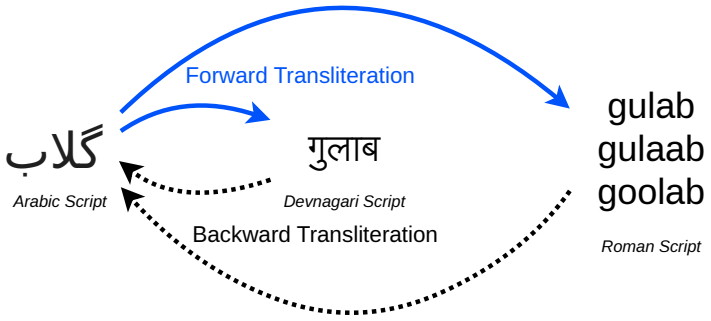

Figure 2.2: Example of a transliteration

Example 1

Example of Code-Mixed sentence

আমি বুন্দি লাড্ডু ভালোবাসি; *could you send me some?*

English Translation: I am fond of boondi-laddoos; could you send me some?

Example 2

Example of Code-Mixed sentence with transliteration

Kal subha bazaar mai I met Rahul.

English Translation: I met Rahul in the market yesterday in the morning.

2.3 Are Code-Switching and Code-Mixing same?

There is a related concept called Code-Switching. However, from a linguistic perspective, it is important to distinguish between Code-Switching and Code-Mixing. Code-switching occurs when a speaker switches between languages at sentence boundaries, whereas code-mixing happens when language alternation occurs within a single phrase. Generally, Code-Switching is regarded as intentional, whereas Code-Mixing tends to occur more inadvertently. Despite this nuanced differences, researchers often employ these terms interchangeably. Consistent with this convention, in this thesis, we refrain from always distinguishing between Code-Switching and Code-Mixing, opting instead to refer to both phenomena collectively as code-mixing.

2.4 Types of Code-Mixing

Drawing upon the mixing of two or more languages, Code-Mixing can be broadly classified into four distinct types.

1. **Inter-Sentential:** In inter-sentential Code-Mixing, language alternation occurs at the boundaries between sentences (more specifically this is code-switching, as defined earlier). This phenomenon is commonly observed among proficient bilingual speakers. For instance, in Example 3, the initial sentence is entirely in English, while the subsequent sentence is written in transliterated Bengali. The transition between languages is evident at the sentence boundaries.

Example 3

Example of inter-sentential CM sentence

Fear cuts deeper than sword. bukta fete jachche.

English Translation: Fear cuts deeper than a sword. It seems my heart will blow up.

2. **Intra-Sentential:** In intra-sentential Code-Mixing, language transitions occur seamlessly within a single sentence, without any discernible interruptions, hesitations, or pauses to signal the shift. Typically, the speaker remains unaware of this linguistic alternation. Example 4 illustrates intra-sentential Code-Mixing, where the initial phrase is in transliterated Bengali, followed by an English phrase within the same sentence. This demonstrates a shift occurring within the confine of a single sentence.

Example 4

Example of intra-sentential CM sentence

Dakho sune 2mar kharap lagte pare but it is true that u are confused.

English Translation: You might feel bad hearing this but it is true that you are confused.

3. **Tag:** This phenomenon entails the incorporation of either a single word or a tag phrase (or both) from one language into another. Example 5 provides an illustration of tag Code-Mixing, wherein the predominant language is transliterated Bengali, with only a few English words interspersed (highlighted in blue).

Example 5

Example of tag CM sentence

Ami majhe majhe fb te on9 hole ei confession page tite aasi.

English Translation: While I get online on facebook I do visit this confession page very often.

4. **Intra-word:** Within intra-word Code-Mixing, the blending of two languages takes place within a single word. Example 6 serves as an illustration of intra-word Code-Mixing, where the word is in English (highlighted in blue) and the suffix is in Bengali.

Example 6

Example of intra-word CM sentence

Tomar osonkkhho admirerder modhhe ami ekjon nogonno manush.

English Translation: Among your numerous admirers I am the negligible one.

2.5 Measuring the amount of CM in corpus

Code-mixed data are predominantly sourced from social media platforms. Consequently, the extent of code-mixing varies across datasets. To address the diverse patterns of code-mixing, several metrics have been proposed to measure the amount of code-mixing within a corpus. Herein, we delve into the discussion of three widely recognized metrics in this domain.

1. **Code-Mixing Index (CMI) [11]:**

   CMI shows the amount of multi-linguality in a given text based on number of words coming from different languages. CMI is expressed in fraction (between 0 and 1) or percentage.

$$CMI = \frac{\sum_{i=1}^{N}(|W_i|) - max_{1\leq j\leq N}\{|W_i|\}}{n-u}$$

$$CMI = \begin{cases} 100 \times \left[1 - \frac{max_{1\leq j\leq N}\{|W_j|\}}{n-u}\right] & : n > u \\ 0 & : n = u \end{cases}$$

   where,

W_i is set of words in language i.

N is the number of languages present in the utterance.

$max\{|W_j|\}$ is the highest number of words present in any language.

n is the total number of tokens in the given utterance.

u is the number of tokens with non-language symbols (not words belonging to any languages).

2. **Multilingual Index (M-Index) [12]:**

   M-Index calculates the distribution of languages used within a text document. It ranges between 0 to 1.

$$\text{M-Index} = \frac{1 - \sum p_j^2}{(N-1)\sum p_j^2}$$

   where $N > 1$ is the total number of languages represented in the corpus. p_j is the probability of a language j present in the corpus. The index is bounded between 0 (monolingual corpus) and 1 (each language in the corpus is represented by an equal number of tokens).

3. **Integration Index (I-Index) [13]:**

   I-Index is a metric that describes the probability of switching within a text. I-Index counts the number of language switches in a document. It ranges between 0 to 1. Given a corpus composed of tokens tagged by language l_i, where i ranges from 1 to $v-1$, and j ranges from $i+1$ to v.

$$\text{I-Index} = \frac{1}{v-1} \sum_{1 \le i < j \le v} S(l_i, l_j)$$

where,

$$\begin{aligned}
\text{switch point} S(l_i, l_j) &= 1, \quad \text{if} \quad l_i \neq l_j \\
&= 0, \quad \text{otherwise.} \\
n &= \text{total number of tokens} \\
u &= \text{number of tokens with non-language symbols} \\
&\quad \text{(not words belonging to any languages)} \\
v &= \text{the number of tokens given language tags} \\
&= n - u.
\end{aligned}$$

2.6 Phonetic Matching Algorithms

When a user submits a query based on her information need, any IR based technique matches the query and documents word-by-word. Here, user-generated comments are treated as documents that largely do not adhere to any spelling standards, especially for transliterated text written using Roman script. Therefore, the same word might have several different spellings between a query and a document and/or across documents. For example, *Hyderabad* is a city in India that can be written like *hyderabade*, *hyedrabad*, *hyderabaad*, *hydrabad*, *haydrabad*, *hydrabaad*, *hyderabad*, *hyderaabad*, *hyderbad* ... and so on by users in different native languages using Roman script. If any query contains the word *Hyderabad*, the term matching-based method fails to retrieve all documents containing these phonetically similar terms that mean the city *Hyderabad* only. If we can add these phonetically identical terms to the query, we obtain all documents related to *Hyderabad*. The difficulty now is how to identify the phonetically equivalent terms.

Phonetic algorithm aids us in finding this spelling variations [14]. There are several phonetic encoding techniques that are used to deal with the issue. We briefly discuss here the following techniques that we have used in our work.

2.6.1 Soundex

It is the most used phonetic coding method [15]. It was originally designed to find phonetically identical English surnames. It employs four characters code while encoding, with three numbers after the initial letter. The scheme's concept is mostly centered around consonant sounds. The English alphabets are clubbed into seven groups depending on their sound. As shown in Table 2.1, all the letters belonging to consonant sounds are encoded with a number (1-6). The letters generating vowel-sounds are saved in a separate group (with the '0' tag), that is not considered during the final encoding. For example, ফুল is a Bengali term meaning *flower*. There might be several variants when someone tries to write it using the Roman script, such as *ful*, *fool*, *phul*, *phool* etc. All these words are phonetically identical. However, their Soundex codes are different. F400 (F, 4 for the l, 0 added, 0 added) is the code for *ful* and *fool*. P400 (P, H ignored, 0 added, 0 added) is the code for *Phool* and *phul*. Soundex uses the original letter as the first code that creates two different codes having the same sound. This becomes a problem as it reduces the number of valid alternative candidates for a word in transliterated domain for matching between a query and a document. Another important limitation of Soundex is that longer words also truncate to a four-lettered code. This often produces the same Soundex code identical to completely different-sounding words.

Code:	0	1	2	3	4	5	6
Letters:	a, e, i, o u, y, h, w	b, p f, v	c, g, j, k q, s, x, z	d, t	l	m, n	r

Table 2.1: Soundex phonetic codes

2.6.2 Phonix

Phonix (phonetic indexing), developed originally by Zobel and Dart [16, 17], is a phonetic technique based on Soundex to retrieve similar-sounding terms (mainly names)

that may differ in spelling. As shown in Table 2.2, the codes used for Phonix encoding differ somewhat from those used for Soundex. The approach includes two distinct consonant groups on top of Soundex. Following that, 'phonetic substitution' [16] was incorporated into the encoding operations. Consider the preceding example from the Soundex section, *ful*, *fool*, *phul*, and *phool*. Their Soundex codes did not match. However, in Phonix, all four words have the same code. The code for *ful*, *fool* are F400 (F, 4 for the l, 0 added, 0 added), whereas the code for *phul*, *phool* are also F400 accroding to substitution rule (F substitution for PH, 4 for the l, 0 added, 0 added).

Code:	0	1	2	3	4	5	6	7	8
Letters:	a, e, i, o u, y, h, w	b, p	c, g, j, k	d, t	l	m, n	r	f, v	s, x, z

Table 2.2: Phonix phonetic codes

2.6.3 Hindex

One of the major limitations of the above two algorithms is that they are language-specific, specifically designed for English. Moreover, vowels are completely ignored in both these phonetic schemes. To solve some of these issues, and keeping especially Hindi language in mind, Prabhakar et al. [18] introduced an approximate string matching algorithm called *Hindex* to measure dissimilarity between two terms using the fundamental character-wise mapping concept of Soundex, Phonix, and *Levenstein edit distance*. They customized a set of rules for the phonetic encoding of Hindi terms transliterated in Roman. The rules are derived based on basic ideas of Soundex, Phonix, and phonetic sounds of consonants and vowels. Coding for Hindex is shown in Table 2.3 and 2.4.

2.7 Types of IR

Information Retrieval is the process of searching relevant documents from a large collection of electronically stored unstructured text documents based on a user query that

Table 2.3: Hindex codes

Characters	Code	Characters	Code
b, v, w	0	ch	!
k, q	1	a	”
d, r	2	e	_
j, z	3	i	?
g	4	o	.
l	5	u	#
m	6	t	T
n	7	p	P
f, ph	8	x, ks, ksh	X
s, sh	9	y	Y
h	H	c	C

Table 2.4: Hindex Codes for vowels pair

Vowels		Codes	Vowels		Codes
a	a	”	i	o	$
a	e	_	i	u	$
a	i	_	o	a	.
a	o	@	o	e	,
a	u	@	o	i	,
e	a	_	o	o	#
e	e	?	o	u	@
e	i	_	u	a	.
e	o	$	u	e	/
e	u	$	u	i	/
i	a	-	u	o	.
i	e	-	u	u	#
i	i	?			

fulfills the user's information need [19]. Given a query q and a document pool $\mathcal{D}$, the task of an IR engine is to rank a subset of documents in $\mathcal{D}$ in the decreasing order of their closeness scores (also known as similarity scores or retrieval status values) to q. IR can be categorized based on the languages and scripts utilized. When considering languages alone, IR falls into two primary classifications: monolingual IR and cross-lingual IR.

- **Mono-lingual IR:** When both the query and the documents belong to a single language, the retrieval task is called monolingual IR. For example, a query q belongs to a language l_i and the documents $D = \left\{d_i^{(1)}, d_i^{(2)}, \ldots, d_i^{(k)}\right\}$ are also in the same language l_i with k being the number of documents in language l_i within the collection $\mathcal{D}$. Figure 2.3 illustrates a scenario of monolingual IR settings, where both the query and documents are in Hindi.

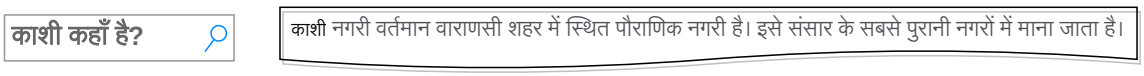

Figure 2.3: Example of Mono-lingual IR

- **Cross-lingual IR:** Cross-lingual IR refers to the scenario where query terms and documents belong to two different languages. Assume that, a query q belongs to a language l_i and documents D_j in language l_j where $i \neq j$. Example 2.4 shows the cross-lingual IR settings, with the query in English and documents in Hindi.

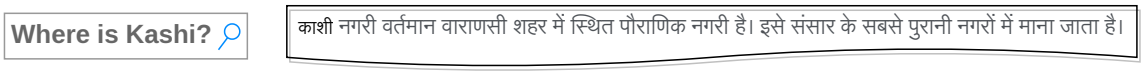

Figure 2.4: Example of Cross-lingual IR

With respect to the scripts used, IR falls into two classifications: mono-script IR and mixed-script IR.

- **Mono-script IR:** When both the query and the documents use a single script, the retrieval task is referred to as mono-script IR. If we also consider the language

of the query and documents, further classifications can be made into mono-lingual mono-script and multi-lingual mono-script scenarios. Figure 2.3 also represents the monolingual mono-script scenario. Figure 2.5 represents the multilingual mono-script, containing terms from both Hindi and English.

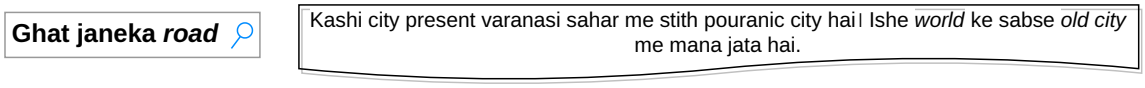

Figure 2.5: Example of multilingual mono script IR

- **Mixed-script IR:** Mixed-script IR refers to the cases where query and documents are in two different scripts. Similarly, within this context, further classifications can be made based on the language of the query and documents, distinguishing between mono-lingual mixed-script and multi-lingual mixed-script scenarios. Figure 2.6 illustrates a scenario within mixed script IR settings, where the query is articulated in the English language and Roman script, while the documents are in Hindi, but use both the Devanagari script and Roman script (Transliterated).

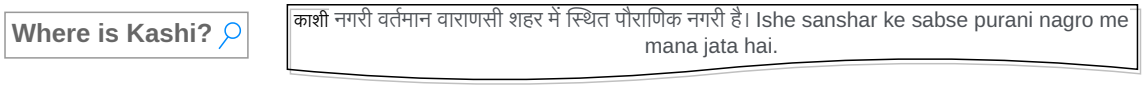

Figure 2.6: Example of Mixed script IR

- **Code Mixed IR:** As described by Chakma and Das [10], when query terms and documents belong to different languages which may be using their native scripts or non-native ones, the retrieval set up is called code-mixed IR (CMIR). 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 $L(k)$.

Fig. 2.7 shows an example of code mixed IR settings, where the query is formulated in the transliterated Hindi language using the Roman script with English language, while the documents are composed in a mixture of two languages, Hindi and English, employing both the Devanagari script and Roman script (Transliterated).

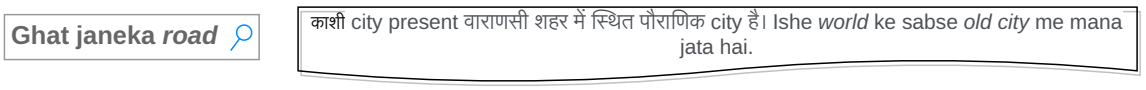

Figure 2.7: Example of Code Mixed IR

2.8 IR Framework

An IR framework serves as an all-in-one platform for indexing, searching, retrieving document collections, and evaluating large-scale retrieval applications. Among the various IR frameworks available, notable ones include Lemur [1], Terrier [2], and Lucene [3].

In our study, we employ the Terrier IR framework, an open-source search engine, for indexing, searching, and retrieval of document collections. Terrier stands out for its high degree of flexibility, efficiency, and effectiveness, making it particularly well-suited for large-scale IR experimentation. Terrier incorporates cutting-edge indexing

[1] `https://www.lemurproject.org/`
[2] `http://terrier.org/`
[3] `https://lucene.apache.org/`

and retrieval functionalities, offering an optimal environment for fast development and assessment. A user here articulates her information need using a query and a retrieval model employed matches query terms with document terms in the collection, thereby retrieving a ranked list of documents from the collection. For a query q, the relevance score of a retrieved document d is computed by

$$score\,(q,d) = \sum_{t \in q} score(t \in d)$$

where $score(t \in d)$ reflects the weight of a query term t in document d generated by the chosen retrieval model.

The models that are used in the thesis are described below. For all such IR models, we use the following notations.

tf_d : term frequency of a term t in a document d

dl : document length, expressed in number of terms

$avdl$: average document length

d_f : document frequency of term t

N : total number of documents in the collection

k_1 : term-frequency parameter, a constant

b : document length normalization parameter, a constant

F : frequency of t in the collection, *aka* collection frequency (CF)

n_e : expected number of documents containing a given term t with CF F

$= N\left(1 - \left(1 - \frac{d_f}{N}\right)^F\right)$

tfn : normalized term frequency, given by the normalization 2 [20] :

$= tf \cdot log_2 \left(1 + c\frac{avdl}{dl}\right)$

c : a free parameter of the normalisation method

tfn_e : modified version of the normalized term frequency using normalization 2 [21]:

$= tf \cdot log_e \left(1 + c\frac{avdl}{dl}\right)$

- **BM25:** It is one of the best-known term-weighting schemes. BM stands for best match. This method accounts for three components: the term frequency, document frequency, and the length of the document.

$$w\left(t,d\right) = tf_d \left(\frac{log\left(\frac{N-d_f+0.5}{d_f+0.5}\right)}{k_1\left((1-b)+b\frac{dl}{avdl}+tf_d\right)}\right) \tag{2.1}$$

- **TF-IDF:** This weight is a statistical measure for evaluating the importance of a word in a selected document within a corpus. The importance increases proportionally to the number of times a word appears in a document but is offset by the frequency of the word in the corpus.

$$w\left(t,d\right) = \left(\frac{tf_d}{k_1\left((1-b)+b\frac{dl}{avdl}\right)+tf_d}\right) \cdot log\left(\frac{N}{d_f}+1\right) \tag{2.2}$$

- **In_expB2:** The term weight of term t in document d in the inverse expected document frequency model for randomness is obtained by the ratio of two Bernoulli's processes for first normalization and normalization 2 for term-frequency normalization. The logarithms have a base of 2.

$$w\left(t,d\right) = \frac{F+1}{d_f\left(tfn+1\right)}\left(tfn \cdot log_2\frac{N+1}{n_e+0.5}\right) \tag{2.3}$$

- **In_expC2:** The term weight of term t in document d in the inverse expected document frequency model for randomness here is obtained by the ratio of two Bernoulli's processes for first normalization and normalization 2 for term-frequency normalization. With respect to the above, the only difference is in the use of logarithmic base of e.

$$w\left(t,d\right)=\frac{F+1}{d_f\left(tfn_e+1\right)}\left(tfn_e\cdot log_2\frac{N+1}{n_e+0.5}\right) \tag{2.4}$$

- **InL2:** InL2 is an inverse document frequency (IDF) model with Laplace after-effects and normalization 2. This model can be used for tasks that require initial precision.

$$w\left(t,d\right)=\frac{1}{tfn+1}\left(tfn\cdot log_2\frac{N+1}{d_f+0.5}\right) \tag{2.5}$$

2.9 Deep Learning Framework

In this section, we are going to explore some machine learning and deep learning frameworks and embedding strategies that have enhanced natural language processing (NLP) and related to our work. In order to enable intelligent machines to comprehend and process information as close as possible to human cognition, these techniques are essential for modelling sophisticated data structures and deriving meaningful representations.

- **Recurrent Neural Networks (RNNs):** RNNs are artificial neural networks that sequentially process data by preserving internal states in its memory. RNNs can capture temporal dependencies in sequential data, unlike feedforward neural networks, which analyze input data in a single run without considering the sequential nature of the data. This is accomplished by employing recurrent connections, letting information endure over time. Each hidden state is updated according to the input and its previous state in the time domain. RNNs are helpful in many fields, including speech recognition, natural language processing, and time series

prediction.

- **Long Short-Term Memory (LSTM):** A LSTM represents a class of RNN architectures devised to mitigate the vanishing gradient problem and effectively capture prolonged dependencies within sequential data. Distinguished from conventional RNNs, LSTMs exhibit a more intricate design characterized by specialized memory cells and gating mechanisms. These components, called gates such as the input gate, forget gate, and output gate, empower LSTM with the ability to selectively retain or discard information across temporal sequences. Leveraging these memory cells, an LSTM excels in learning and preserving patterns over extended intervals, rendering them well-suited for tasks including speech recognition, language modeling, and time series prediction. Consequently, LSTMs have emerged as a cornerstone in sequence modeling, finding widespread application across diverse domains wherein the capture of temporal dependencies is paramount.
- **Bidirectional Long Short-Term Memory (Bi-LSTMs):** Bi-LSTMs represent a variant of RNN architectures focused to assimilate information from both past and future time steps during the processing of sequential data. In contrast to traditional LSTMs or RNNs, which solely consider past context, Bi-LSTMs leverage two distinct LSTM layers: one processes the input sequence in its original order (forward), while the other traverses it in reverse order (backward). This bidirectional processing mechanism empowers the model to capture dependencies from both temporal directions, thereby adeptly encapsulating long-range dependencies and contextual information within the input sequence. Bi-LSTMs have garnered widespread adoption in diverse natural language processing endeavors, encompassing tasks such as sentiment analysis, named entity recognition, and machine translation, where discerning the contextual information of words or tokens within a sentence is indispensable for precise predictions.

- **Convolutional Neural Networks (CNNs):** CNNs are a type of deep neural networks specifically engineered to handle organised input that resembles a grid, including pictures and sequences, while accurately identifying temporal and spatial relations. They comprise various layers, encompassing convolutional layers, pooling layers, activation functions, and fully connected layers. Filters traverse over input data in convolutional layers to extract local features and produce feature maps. These feature maps are downsampled by pooling layers, which reduces dimensionality without sacrificing important information. Rectified Linear Unit (ReLU) and other activation functions add non-linearity, which improves the network's capacity to recognise complex patterns. Fully linked layers incorporate acquired characteristics for tasks involving regression or classification. Due to their exceptional performance in tasks like object identification, semantic segmentation, and picture classification, CNNs have significantly advanced the field of computer vision. CNNs are also used for sequential data, like text, where they can extract features from word embeddings and various tasks like text classification, sentiment analysis, and machine translation.
- **Multi-CNN:** Multi-CNN represents a deviation from the conventional CNN architecture, tailored to concurrently process multiple 1-D convolutional layers. Each convolutional layer operates with distinct filter sizes, such as 2, 3, and 4. Concatenation is used to combine output feature maps from all parallel convolutions. This concatenation operation combines the information extracted by each convolutional layer, yielding a cohesive feature representation that encapsulates diverse aspects of the input data across varying filter sizes. Subsequently, this unified feature representation undergoes processing through dense (fully connected) layers.
- **Conditional Random Field (CRF)**: CRF [22] utilizes undirected graphical models to calculate the conditional probability of values of output nodes when

the values of the input nodes are known. Conditional probability is defined for a state sequence $S = \{s_1, s_2, \ldots, s_n\}$ given a corresponding observation sequence $O = \{o_1, o_2, \ldots, o_n\}$, is defined as:

$$P(S \mid O) = \frac{\exp\left(\sum_i \sum_k \lambda_k f_k\left(s_{i-1}, s_i, O\right)\right)}{\sum_S \exp\left(\sum_i \sum_k \lambda_k f_k\left(s_{i-1}, s_i, O\right)\right)} \tag{2.6}$$

here, $f_k\left(s_{i-1}, s_i, O\right)$ is real-valued and represents the feature function whose weights are denoted by λ_k, and the aim of during training exercise is to effectively learn these weights.

In light of the sequence tagging task, the sentences represent the state sequence and the corresponding tags for that sentence represent the observation sequence.

- **Word2vec:** Word2vec is a method to create a word representation, popularly called an *embedding*, that is used for representing input text. The embedding so generated is used in other downstream deep learning components efficiently and has been around since 2013 [23]. It allows users to compute continuous vector representations of words from large datasets in less time and improves the semantic regularities of learned words using the word offset technique.
- **Character-Level Recurrent Neural Networks (CharRNNs):** CharRNNs represent a variation of RNN architectures that operate at the character level, analyzing an input text character-by-character. In contrast to traditional word-level RNNs, which process words or tokens, CharRNNs directly learn text representations from individual characters, enabling them to capture complex patterns and dependencies within the text. This capability allows CharRNNs to effectively handle languages characterized by complex morphology, unseen words, or noisy data. CharRNNs have been applied to various tasks including text generation, spelling correction, and morphological analysis. By learning word representations from corresponding character representations, these models adapt their representations to the specific task at hand. Additionally, the utilization of character

representations facilitates the graceful handling of out-of-vocabulary words. Each character is initialized with a random representation vector within the character look-up table. In sentence processing, the characters are organized into sequences and presented in both forward and reverse order to the forward and backward passes of the Bi-LSTM.

- **FastText:** FastText [24] is an open-source library that combines concepts from other successful models such as bag-of-words, n-gram with the help of a new extension called subword information. FastText is an extension of the word2vec model. Each word is represented by an n-gram of letters. This permits the embeddings to recognize suffixes and prefixes, as well as to capture the meaning of shorter words. FastText can outperform other deep learning-based methods in terms of training times by significantly decreasing it while increasing the accuracy.
- **Global vectors for word representation (GloVe):** GloVe [25] is a word embedding approach that captures both global and local data of a corpus. It is an unsupervised learning approach that aggregates global word-word co-occurrence matrices from a corpus to generate word embeddings. In vector space, the resultant embeddings reveal fascinating linear substructures of the word. It is based on word-context matrices and matrix factorization algorithms.
- **Transformer:** In 2017, Vaswani et al. [26] introduced a novel architecture known as the Transformer. As indicated by the name, machine translation provided the inspiration for this architecture. From there, the Transformer model represents an innovative NLP architecture that is well-known for its exceptional performance in various tasks. Structurally, the Transformer architecture adheres to an encoder-decoder framework. The task of the encoder is to map an input sequence to a series of continuous representations, which are subsequently forwarded to the decoder. The decoder leverages both the encoder output and the decoder output from the preceding step to generate an output sequence. Both encoder and

decoder comprise multiple layers of self-attention mechanisms and feedforward neural networks. The transformer can dynamically assign different input tokens to different levels of importance through self-attention. Given the absence of recurrent networks capable of retaining sequential input order, positional encoding emerges as a requisite technique to impart relative position information to each word in the sequence. This enhancement bolsters the model's capacity to parallel processing of sequential data and to capture long-range dependencies within the input data.

- **Bidirectional Encoder Representations from Transformers (BERT):** BERT uses a transformer, an attention mechanism that learns contextual relationships between words (or sub-words) in a text. In its most basic version, the transformer consists of two distinct mechanisms: an encoder that reads the text input and a decoder that provides a prediction for the task. The models are pre-trained on large text corpora such as Wikipedia and produce state-of-the-art results with necessary fine-tuning on several downstream tasks. The contextual language representation model BERT [27] has been used for the downstream task of code-mixed language identification. We have used two variations of BERT from the Hugging Face library[4]: `BERT-BASE` and `BERT-LARGE`. There are a number of transformer blocks in both BERT model sizes (also known as encoder blocks). The base version has 12 encoder blocks, whereas the large version contains 24 encoder blocks. Both the base and large versions include larger hidden units (feedforward networks), viz. 768 and 1024, respectively - as well as more attention heads, viz. 12 for the base version and 16 for the large version.

  The variants for the base and large versions are listed below.

  - **bert-base-cased**: The `bert-base-cased` is pre-trained on English text and has a total of 110M parameters.

[4] `https://huggingface.co/transformers/pretrained_models.html`

- **bert-large-cased**: The `bert-large-cased` is pre-trained on English text and has a total of 335M parameters. The term `cased` indicates that the model is trained on cased English text.
- **bert-base-uncased**: The `bert-base-uncased` is pre-trained on English text and has a total of 109M parameters. The term `uncased` indicates that the model is trained on lower-cased English text.
- **bert-base-multilingual-cased**: The `bert-base-multilingual-cased` [5] (also known as Multilingual BERT or mBERT) is pre-trained on cased text in the top 104 languages with the largest wikipedia and has a total of 179M parameters with 12 transformers blocks, 768 hidden layers and 12 attention heads.

BERT as a model accepts `[CLS]` as a first special input token, followed by a sequence of words as input. The input is subsequently sent to the next layer. `[CLS]` here stands for Classification. Each layer applies self-attention and communicates the outcome to the next encoder using a feedforward network.

2.10 Evaluation Metrics

Evaluation is an essential task for assessing the performance and effectiveness of a system. Quantitative evaluation is done using one or more metrics defined a priori and system performances are compared based on the values of the metrics so defined. In classification tasks, evaluation metrics such as accuracy, precision, recall, and F_1-score are used to provide quantitative measures of how well a model categorizes input data into predefined classes or categories. These metrics reflect the model's ability to classify instances correctly. Similarly, in information retrieval tasks, evaluation metrics like precision, recall, and mean average precision (MAP) are some of the metrics used for quantifying the relevance and effectiveness of search results retrieved by a system. By

[5] `https://github.com/google-research/bert/blob/master/multilingual.md`

analyzing these metrics, we can examine the system's ability to retrieve and rank relevant documents appropriately. In this thesis, we can categorize all tasks into two types: classification-related tasks and IR-related tasks. We describe both in the following.

2.10.1 Classification Tasks

Evaluation of classification tasks is done using a confusion matrix. A confusion matrix serves as an important setup particularly in evaluation of binary classification tasks [dividing a set of samples into 2 classes: zeros (0) and ones (1)]. It organizes classification outcomes into four distinct categories: *True Positive (TP)*, representing instances where both the actual and predicted values are 1; *True Negative (TN)*, denoting cases where both the actual and predicted values are 0; *False Positive (FP)*, indicating instances where the actual value is 0 but the predicted value is 1; and *False Negative (FN)*, representing scenarios where the actual value is 1 but the predicted value is 0. In binary classification, exemplified in Figure 2.8, the event targeted for spam identification (1 means Spam) is labeled as Positive, while the absence of the event (0 means Not Spam) is denoted as Negative. Subsequently, these values are computed and populated within the confusion matrix.

We evaluate and compare performance of different classification techniques in terms of precision, recall, F_1-score, weighted average F_1-score and accuracy, the definitions of which are given below.

- **Precision:** It is the fraction of correct prediction of the targeted class out of total number of such predictions. Thus it is expressed as the ratio of TP to the sum of TP and FP.

$$\text{Precision}(P) = \frac{TP}{TP + FP}$$

- **Recall:** It is the fraction of correct prediction of the targeted class from the actual number of such instances in the data. It is thus expressed as the ratio of

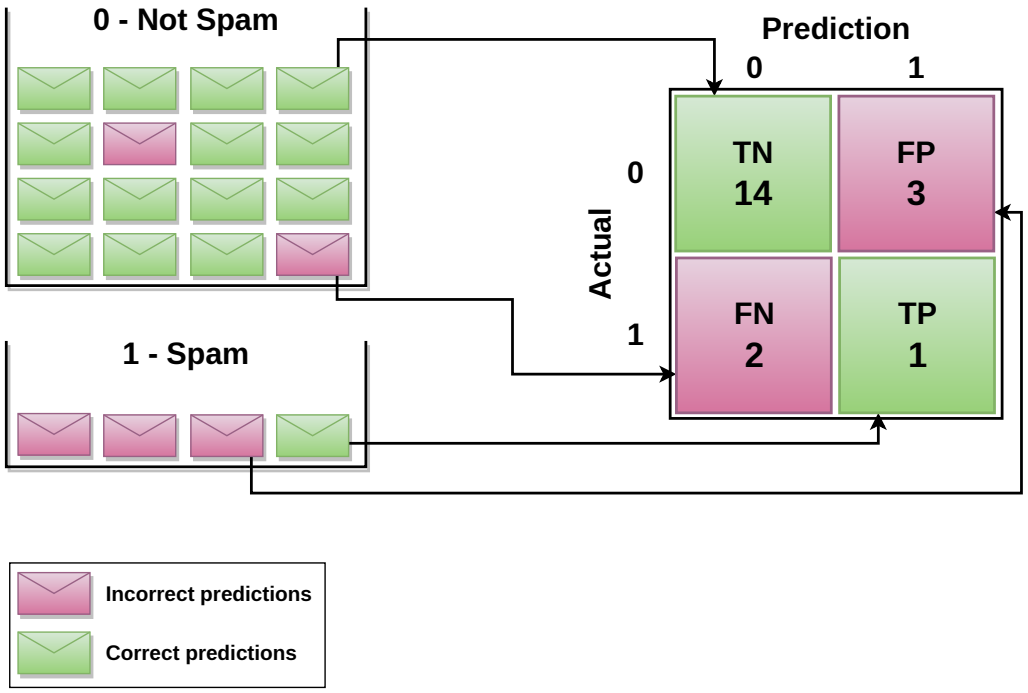

Figure 2.8: Populating the classification outcome in the Confusion Matrix

TP to the sum of TP and FN.

$$\text{Recall}(R) = \frac{TP}{TP + FN}$$

- **Accuracy:** It is the ratio of number of correct predictions to the total number of test instances i.e.,

$$Accuracy = \frac{\#\text{ correct predictions}}{\#\text{ test-instances}} = \frac{TP + TN}{TP + TN + FP + FN}$$

- F_1**-score:** It is used to have an overall idea of performance comprising both precision and recall. It is expressed as the harmonic mean of precision and recall and used to have a composite idea of precision and recall.

$$F_1\text{-score} = \frac{2 * R * P}{R + P}$$

When there are more than two classes, this idea needs to be extended further. Eval-

uation of a multi-class classification task can be done by considering the task as a collection of several 2-way classification tasks. Evaluation metrics is thus computed by considering one class vs others for each of the classes. Two popular versions of F_1-metrics are macro-average F_1 and micro-average F_1 which are defined below.

- **Macro-average F_1:** It is the simple average of class-wise F_1-scores where n is the number of classes.

$$\text{Macro-average } F_1 = \frac{\sum_{i=1}^{n} F_1 - score_i}{n}$$

- **Micro-average F_1:** It is calculated as the balanced harmonic mean of the overall multi-class precision (P_{multi}) and multi-class recall (R_{multi}) which are defined as below.

$$\begin{aligned} P_{multi} &= \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n}(TP_i + FP_i)} \\ R_{multi} &= \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n}(TP_i + FN_i)} \\ \text{Micro-average } F_1 &= \frac{2 * P_{multi} * R_{multi}}{P_{multi} + R_{multi}} \end{aligned}$$

- **Weighted average F_1:** It is the weighted version of the average F_1-scores where each class is weighted by the number of samples from that class (also called support of that class).

$$\text{Weighted average } F_1 = \frac{\sum_{i=1}^{n} support_i * F_{1i}}{\sum_{i=1}^{n} support_i}$$

2.10.2 IR Tasks

We evaluate the performance of different retrieval models by the following evaluation measures.

- **Precision:** It is the fraction of retrieved relevant documents among the total retrieved documents.

$$\text{Precision} = \frac{\text{number of documents retrieved that are relevant}}{\text{Total number of documents that are retrieved}}$$

- **Recall:** It is the fraction of relevant documents retrieved from a set of relevant documents:

$$\text{Recall} = \frac{\text{number of relevant documents retrieved}}{\text{Total number of relevant documents in the collection}}$$

- **Precision@k:** For a given query, there may be hundreds of relevant pages, and relatively few users need to read all of them. Recall is thus often not regarded as an important metric, particularly for IR tasks similar to web search.

$$Precision@k = \frac{\text{Number of relevant documents retrieved in top k documents}}{k}$$

  Precision@k or P@k is considered as a useful metric since it needs to consider only the top-k documents (for example, P@10 needs reading only top-10 retrieved documents). The advantage of this measure is that it does not require any prior information regarding the total number of relevant documents in the collection.
- **R-Precision:** R-precision is the precision at R, where R is the number of relevant documents for a query Q. In other words, if there are r relevant documents among the top-R retrieved documents, then R-precision $= \frac{r}{R}$
- **Average Precision (AP):** Average precision is the mean of the precision values calculated after retrieval of every relevant document at rank R_k. If the total

number of relevant documents for a given query is 'n', then

$$AP = \frac{1}{n} \sum_{k=1}^{n} precision\,(R_k)$$

If for the given query, a system can not retrieve all n relevant documents, the precision values corresponding to the non-retrieved documents are considered zero (as if they are retrieved at rank infinity).

- **Mean Average Precision (MAP):** As AP corresponds to a single query, the performance of a system can not be reliably judged based on a single AP score. System retrieval performance should be considered for a number of queries so that a its overall performance can be gauged. MAP is used to provide a single value for the performance of a system over a set of queries. It is calculated as the mean of the AP scores for a number of queries.

$$MAP = \frac{1}{|Q|} \sum_{t=1}^{|Q|} AP\,(t)$$

where Q is a set of queries and t corresponds to the topic or query id.

In recent years, the most used evaluation metric for IR tasks has been MAP. TREC_EVAL software is used to compute the MAP value. Simple arithmetic mean as a performance metric indicates that we give equal weight to all the queries in the topic set. MAP scores over a set of at least 50 queries is often used for IR evaluation.