Winter Internship Project
Report

Emotions and Threat Detection From Urdu Language

Submitted by

Shirish Shekhar Jha
BS Data Science and Engineering
Indian Institute of Science Education and Research Bhopal

Under the guidance of

Dr. Kirti Kumari
Assistant Professor, Department of CSE, Indian Institute of Information Technology, Ranchi, India

Department of Computer Science and Engineering
INDIAN INSTITUTE OF INFORMATION OF TECHNOLOGY
Ranchi, India

Winter Internship 2022

Abstract

Text Classification is a task that has seen a huge contribution from all around the world in the past few years. With Social Media seeing a massive surge in enrollment of users; as a result, the amount of textual data available has increased significantly. Textual data has played a vital role in many aspects of business and research. Many companies formulate their business strategy about a particular product based on the reviews commented on or made available in the textual format. Analysis of textual data has been evident in the research field as well because it has resulted in the invention of more intelligent software and applications. The multilingual point of view of the analysis of textual data is also significant due to the wide variety of speakers in the world. Mostly, contributions have been made in English Language domain, but analyzing the multi-linguistic characteristics of the text is also essential. In the given project, efforts have been made to perform two tasks. Firstly, The multi-label emotion classification of the Urdu Language, followed by Threat Detection of Urdu Tweets as the second task. The second task involved two sub-tasks; categorizing the tweets as threatening and non-threatening and then classifying them as targeted towards individuals or Groups. Methodologies like Machine Learning, Bag of Words, Tf-Idf Classification, and deep learning were used as experimentation, and their results were analyzed.

Contents

Chapter 1

Work Done

1.1 Introduction

The project describes **Multi-label Emotion Classification of Urdu text and Classifying Urdu Tweets as Threatening and Non-Threatening**, which were tasks as a part of the **EmoThreat@FIRE 2022**.

The experiments were performed to provide detailed information about the methodologies used and the dataset. The pre-processing steps involved in processing the data have also been briefly mentioned and explained. The upcoming report is divided into several major sections of ***Dataset, Exploratory Data Analysis, Machine Learning, Deep Learning, Results, and Conclusion***.

1.2 Dataset

Subtask A

The dataset comprises Urdu Nastalíq tweets elucidating people's emotions as they describe their activities, opinions, and events with the world. This dataset aims to develop a prominent benchmark in Urdu for the multi-label emotion classification task.

The training Dataset had 7800 data points; on the other hand, the test dataset had 1950 data points. Both datasets had no null values or any discrepancies in the type of values stored in the dataset's column. The train and Test Dataset comprised one *sentences* column and seven *emotions* columns. The emotions to be classified as the task were *anger, disgust, fear, sadness,* *surprise, happiness, neutral.* The task was to train a classifier based on the training dataset and classify the test dataset to produce satisfactory results.

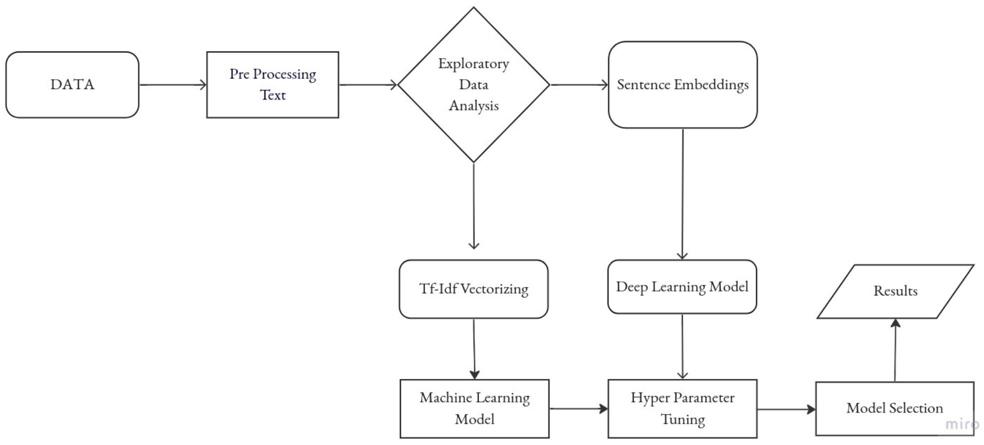
Subtask B

The dataset comprises of threatening tweets in the Urdu language. The detailed processes of dataset collection and annotation are described in this paper. This dataset aims to develop a significant benchmark for threatening language detection in Urdu.

The training and test dataset had 3564 and 935 data points, respectively. Both datasets had no null values or any discrepancies in the type of values stored in the dataset's column. The train and Test Dataset comprised one *Tweet* column and two *target* columns. The targets were *Threat and S/G.* The task was to develop a classification model to classify the tweets into threatening and non-threatening.

1.3 Methodology

Following flowchart describes the steps undertaken to get the results.

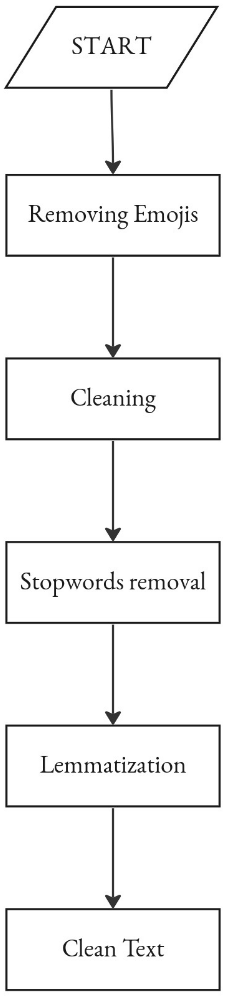


During the experimentation, several machine-learning models were used for both tasks, and Deep Learning was used for Subtask A. Methods like Classification using Bag-of-Words and N-gram classification were also deployed.

Still, the major satisfactory results were obtained using Machine Learning and Deep Learning Models.

1.4 Pre Processing

Text processing is a very essential step while performing NLP tasks. Since the text is collected from social media and other related platforms, it becomes a mandatory step to clean it before producing results. Besides cleaning, several other steps are also involved, like lemmatization and tokenization, which are essential for the NLP tasks. The following flowchart describes the steps involved in Text Pre Processing. (*The steps involved in pre-processing are the same for both tasks*).



1.4.1 Removing Emojis

An emoji is a pictogram, logogram, ideogram, or smiley embedded in the text and used in electronic messages and web pages. The primary function of emoji is to fill in emotional cues otherwise missing from a typed conversation. An emoji is just another character, like letters in the alphabet. The Unicode Standard has assigned numbers to represent emojis. In the Unicode Standard, each emoji is represented as a "code point" (a hexadecimal number) that looks like U+1F063. Our devices worldwide can all agree that U+1F603 is the combination that triggers a grinning face. Since emojis are encoded, it becomes difficult for programming languages to understand their meaning unless provided by the user in advance. Thus, removing emojis from the text was essential to avoid any errors. Emojis express the emotional state and can indicate the intentions expressed with the text; their use is left for future work.
To remove emojis, regex and the emoji library of python were used to substitute Unicode characters with spaces that were further substituted.

1.4.2 Cleaning

After the removal of emojis, text cleaning was done. Text cleaning involved a number of substitutions of characters with spaces followed by space normalization. Using the Urduhack library of python.

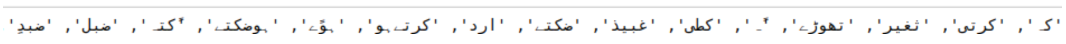
We started with normalizing characters, one of the most important cleaning steps. Every character has a Unicode. You can search for any character Unicode from any language you will find it. No two characters can have identical Unicode. This module works concerning Unicode. The Urdu language has its roots in Arabic, Persian, and Turkish. So we must deal with all those characters and convert them into a normal Urdu character. Thus, normalizing Urdu characters means replacing two identical words with one word, keeping the context of the sentence in mind. Normalizing Urdu text is essential for machine learning because it fixes the problem of correct encodings for the Urdu characters and replaces Arabic characters with correct Urdu characters. It brings all the characters in the specified Unicode range (0600-06FF) for the Urdu language.

Later we went on to remove some exceptional cases like removing URLs, removing emails, removing punctuation, replacing currency symbols, and removing any English alphabets. These cases were to be considered because the dataset had been taken from social media websites, and it is common to find these enlisted entities inside the text.

The last step was to remove accents from any accented Unicode characters in the text string, either by transforming them or completely removing them and normalizing whitespace, which replaces one or more spacings with a single space, and one or more linebreaks with a single newline. Also, strip leading/trailing whitespace.

1.4.3 Stop words Removal

The words that occur commonly in a document and have the least significance in the context of semantic values are known as stop words. The general strategy for determining a stop list is to sort the terms by collection frequency (the total number of times each term appears in the document collection) and then to take the most frequent terms, often hand-filtered for their semantic content relative to the domain of the documents being indexed, as a stop list, the members of which are then discarded during indexing. An example of a stop list is



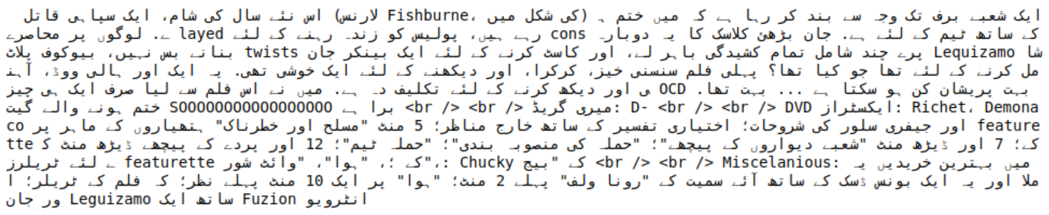
```
'ک', 'کرتی', 'ثغیر', 'تھوڑے', 'ۓ', 'کطی', 'غبیذ', 'ضکتے', 'ارد', 'کرتےہو', 'ہوۓ', 'ہوضکتے', 'ۓکتہ', 'ضبل', 'ضبدِ'
```

Stop words occur multiple times in a document, and the occurrence of stop words has the least semantic value in the document sentences. These words cover a noteworthy bundle of archives that have no semantic significance. So, the stop words ought to be removed for better language description.

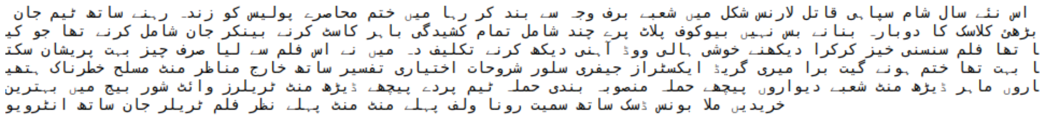
1.4.4 Lemmatization

A text may use words in different forms to meet the context; for example, *he purchased a packet of chips* uses the word *purchase* in its past form to make the user understand that the action of purchasing happened in the past. Additionally, there are families of derivationally related words with similar meanings, such as sarcasm and sarcastic. In many situations, it seems as if it would be helpful to search for one of these words to return documents containing another word in the set. To do so, we do stemming and Lemmatization.
The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time and often includes the removal of derivational affixes. Lemmatization usually refers to

doing things properly with the use of a vocabulary and morphological analysis of words, usually aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. If confronted with the token saw, stemming might return just s, whereas lemmatization would attempt to return either see or saw depending on whether the use of the token was as a verb or a noun. The two may also differ in that stemming most commonly collapses derivationally related words, whereas lemmatization commonly only collapses the different inflectional forms of a lemma. An additional plug-in component often does linguistic processing for stemming or lemmatization to the indexing process, and several such components exist, both commercial and open-source. Seeing the grammatical advantage of Lemmatization, it was preferred to proceed with Lemmatization. For Urdu text lemmatization, we preferred using Urduhack Lemmatizer.



```
ایک شعبے برف تک وجہ سے بند کر رہا ہے کہ میں ختم ہ (کی شکل میں ،Fishburne لارنس) اس نئے سال کی شام، ایک سپاہی قاتل
کے ساتھ ٹیم کے لئے ہے. جان بڑھئ کلاسک کا یہ دوبارہ cons رہے ہیں، پولیس کو زندہ رہنے کے لئے layed ہے. لوگوں پر محاصرے
شا Lequizamo پرے چند شامل تمام کشیدگی باہر لے، اور کاسٹ کرنے کے لئے ایک بینکر جان twists بنانے بس نہیں، بیوکوف پلاٹ
مل کرنے کے لئے تھا جو کیا تھا؟ پہلی فلم سنسنی خیز، کرکرا، اور دیکھنے کے لئے ایک خوشی تھی. یہ ایک اور ہالی ووڈ، آہنہ
بہت پریشان کن ہو سکتا ہے ... بہت تھا. OCD ی اور دیکھ کرنے کے لئے تکلیف دہ ہے. میں نے اس فلم سے لیا صرف ایک ہی چیز
Richet، Demona :ایکسٹراز DVD <br /> <br /> -D :میری گریڈ <br /> <br /> برا ہے S00000000000000000 ختم ہونے والے گیت
feature اور جیفری سلور کی شروحات؛ اختیاری تفسیر کے ساتھ خارج مناظر؛ 5 منٹ "مسلح اور خطرناک" ہتھیاروں کے ماہر پر co
کے؛ 7 اور ڈیڑھ منٹ "شعبے دیواروں کے پیچھے"؛ "حملے کی منصوبہ بندی"؛ "حملہ ٹیم"؛ 12 اور پردے کے پیچھے ڈیڑھ منٹ کے tte
میں بہترین خریدیں یہ :Miscelanious <br /> <br /> کے "بیج Chucky :."کے ؛، "ہوا"، "وائٹ شور featurette ہے لئے ٹریلرز
ملا اور یہ ایک بونس ڈسک کے ساتھ آئے سمیت کے "رونا ولف" پہلے 2 منٹ؛ "ہوا" پر ایک 10 منٹ پہلے نظر؛ کہ فلم کے ٹریلر؛ ا
انٹرویو Fuzion ساتھ ایک Leguizamo ور جان
```

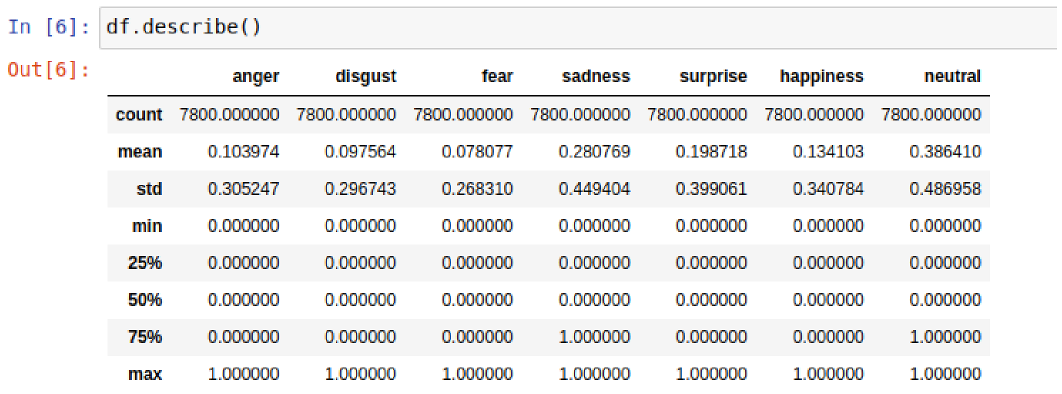


```
اس نئے سال شام سپاہی قاتل لارنس شکل میں شعبے برف وجہ سے بند کر رہا میں ختم محاصرے پولیس کو زندہ رہنے ساتھ ٹیم جان
بڑھئ کلاسک کا دوبارہ بنانے بس نہیں بیوکوف پلاٹ پرے چند شامل تمام کشیدگی باہر کاسٹ کرنے بینکر جان شامل کرنے تھا جو کیا
ا تھا فلم سنسنی خیز کرکرا دیکھنے خوشی ہالی ووڈ آہنی دیکھ کرنے تکلیف دہ میں نے اس فلم سے لیا صرف چیز بہت پریشان سکتا
ا بہت تھا ختم ہونے گیت برا میری گریڈ ایکسٹراز جیفری سلور شروحات اختیاری تفسیر ساتھ خارج مناظر منٹ مسلح خطرناک ہتھیا
اروں ماہر ڈیڑھ منٹ شعبے دیواروں پیچھے حملہ منصوبہ بندی حملہ ٹیم پردے پیچھے ڈیڑھ منٹ ٹریلرز وائٹ شور بیج میں بہترین
خریدیں ملا بونس ڈسک ساتھ سمیت رونا ولف پہلے منٹ منٹ پہلے نظر فلم ٹریلر جان ساتھ انٹرویو
```

1.5 Exploratory Data Analysis

Exploratory Data Analysis, also known as EDA is the most crucial step involved while performing Data Science and related tasks; American mathematician John Tukey developed it in the 1970s. EDA is a technique that provides insights into data and indicates some vital statistics associated with the data. Visualizations often accompany EDA so that the information related to the data is more appealing and can convey messages much more effectively. EDA helps identify patterns in the data, postulate a hypothesis, and help test whether the statistical analyses we are considering for the data are appropriate. The same EDA steps were followed for the Subtasks with a few minute changes each time.

Exploratory data analysis for the Subtask involved studying the Data and finding the balance of different classes present, visualizing some statistics and N-Gram exploration.

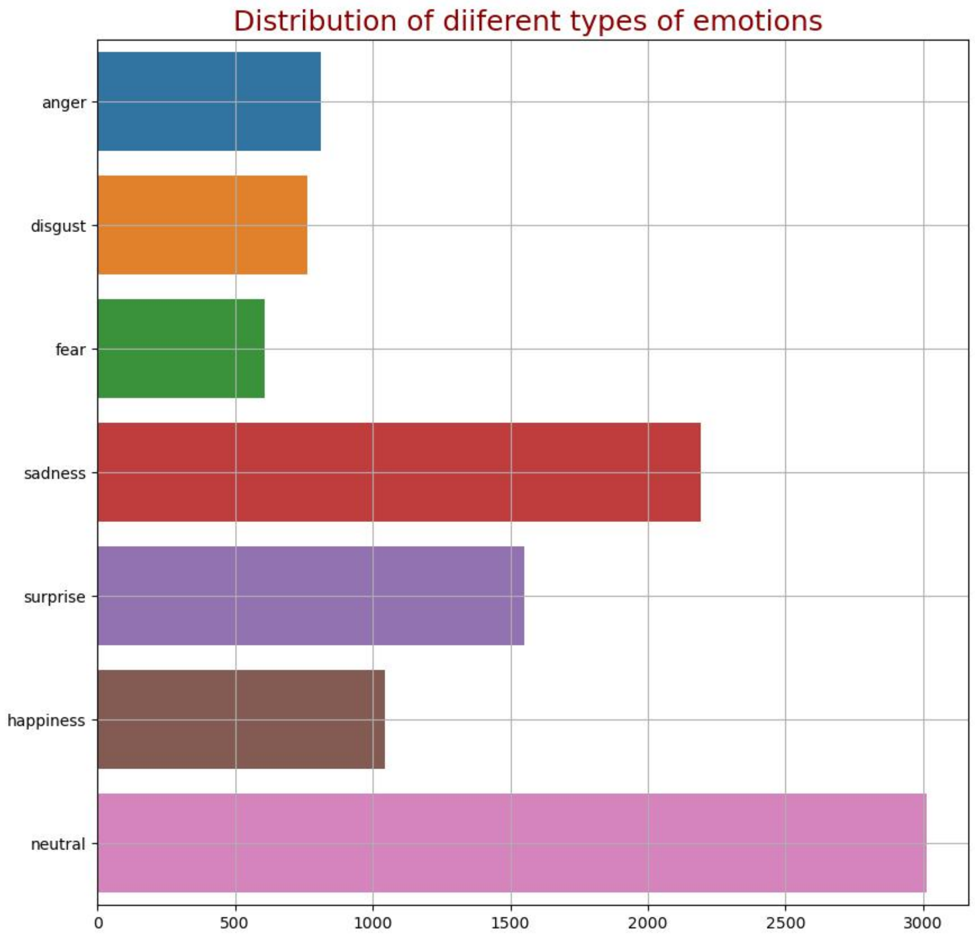


```
In [6]: df.describe()
```

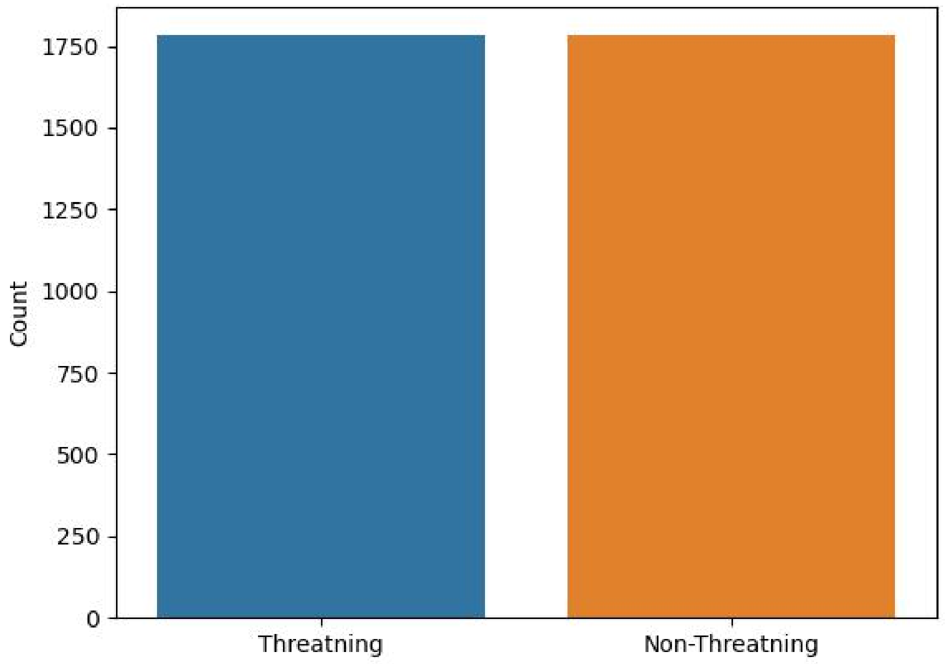
Out[6]:

	anger	disgust	fear	sadness	surprise	happiness	neutral
count	7800.000000	7800.000000	7800.000000	7800.000000	7800.000000	7800.000000	7800.000000
mean	0.103974	0.097564	0.078077	0.280769	0.198718	0.134103	0.386410
std	0.305247	0.296743	0.268310	0.449404	0.399061	0.340784	0.486958
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

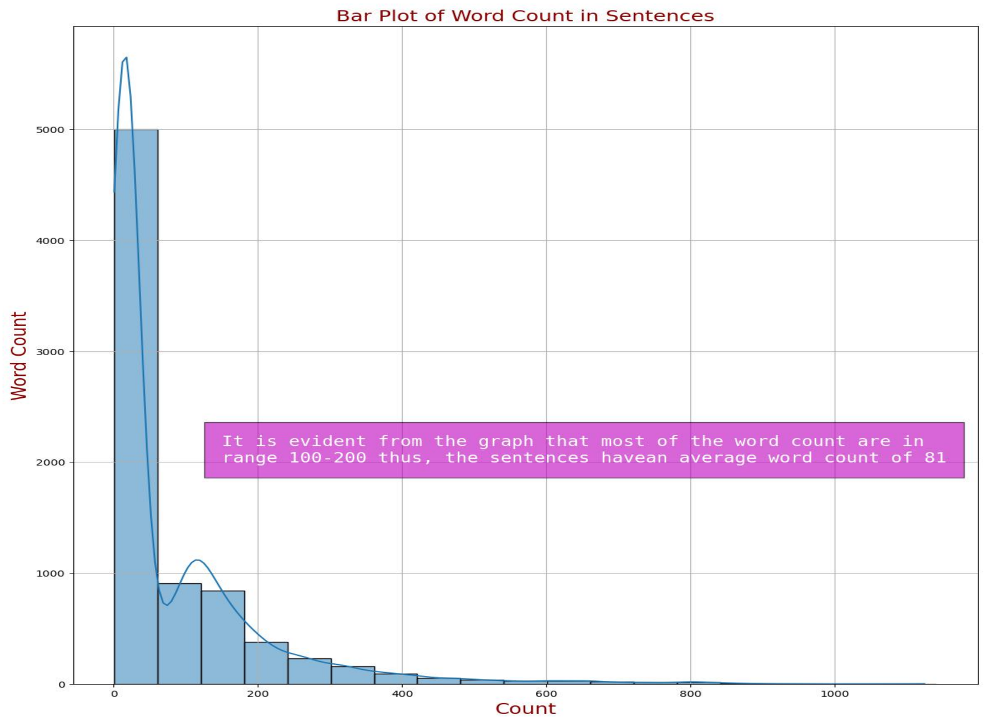
From the above descriptions, we can infer that the labels are [”anger,” ”disgust,” ”fear,” ”sadness,” ”surprise,” ”happiness,” and neutral”]. There are no NA and Null values in the dataset. In statistics, a kth percentile is a score below which a given percentage k of scores in its frequency distribution falls or a score at or below which a given percentage falls. Hence we can hypothesize that the labels are imbalanced as sadness and neutral have a 75 percentile of 1 and might be higher in number. For a data set, the arithmetic mean, also known as ”arithmetic average,” is a measure of the central tendency of a finite set of numbers: specifically, the sum of the values divided by the number of values; since the number of instances of occurrence is the same for every class, and the possible value of each class is either 0 (*meaning this emotion is not present in the sentence* or 1 (*meaning the sentence indicates this emotion*); thus, one with a higher mean indicates the presence of more ones in that particular column that means that class is present slightly more in number creating a class imbalance. Also, since the mean of neutral is more than the sadness, we can infer that the neutral class might have a higher number of ones than the sadness. The given statistic shows that the mean of each class follows the order of $\mu(neutral) > \mu(sadness) > \mu(surprise) > \mu(happiness) > \mu(anger) > \mu(disgust) > \mu(fear)$. To validate this hypothesis, the following plot describes the presence of each class, and it depicts that the dataset had a class imbalance.

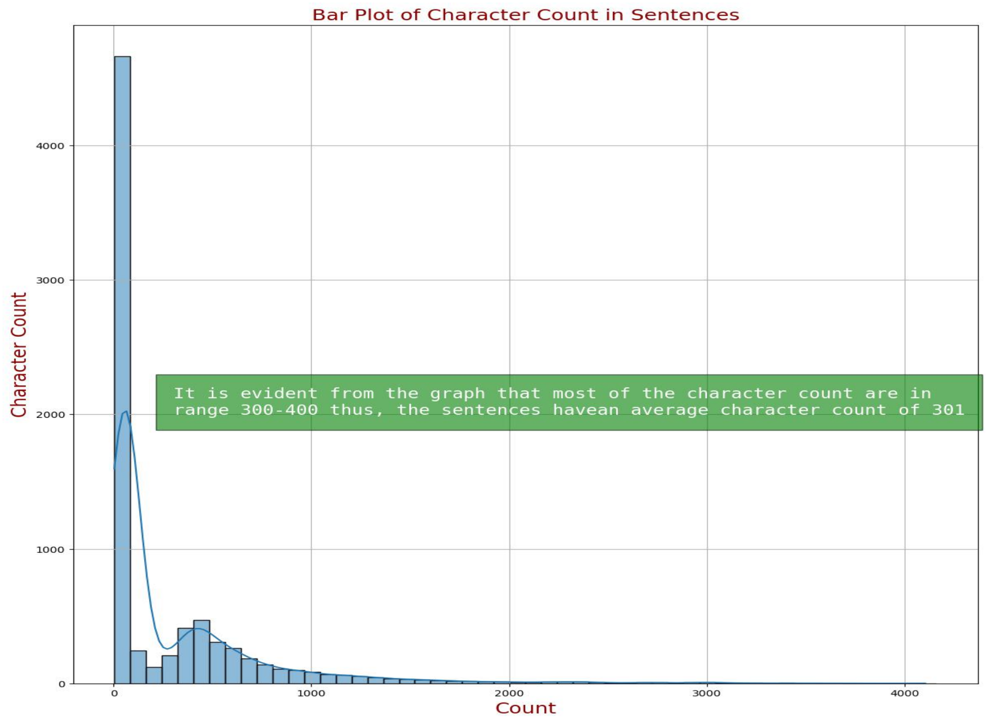


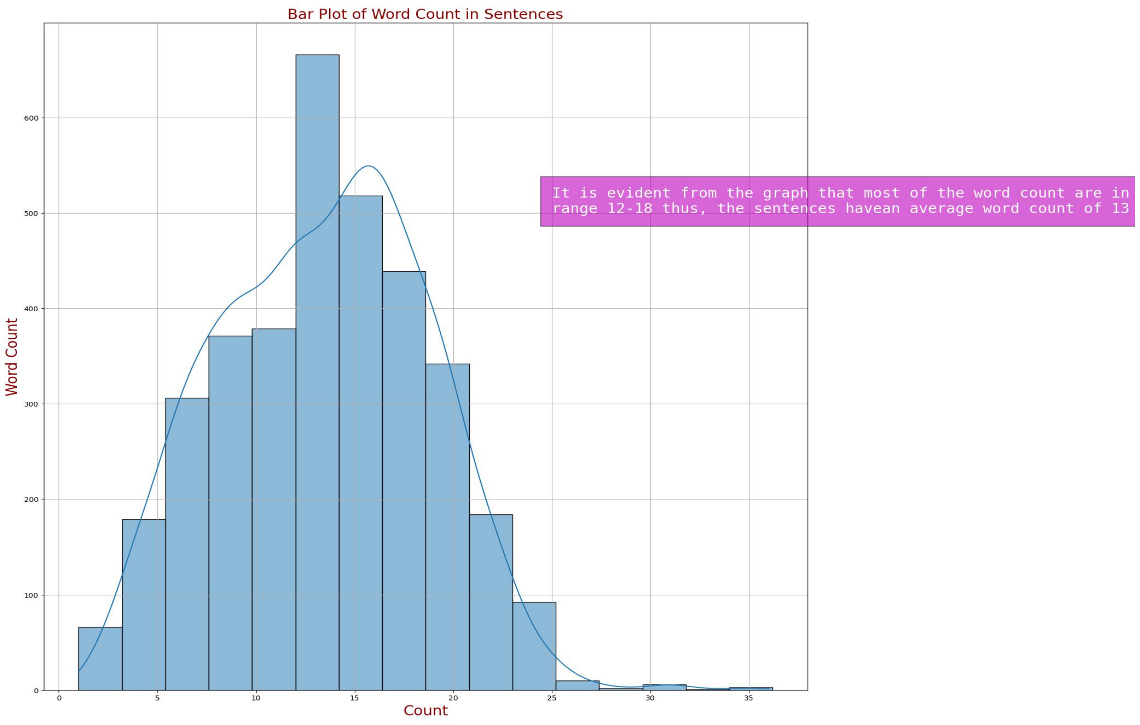
The dataset already had a class balance between threatening and non-threatening Tweets for Subtask B. It is shown as follows.

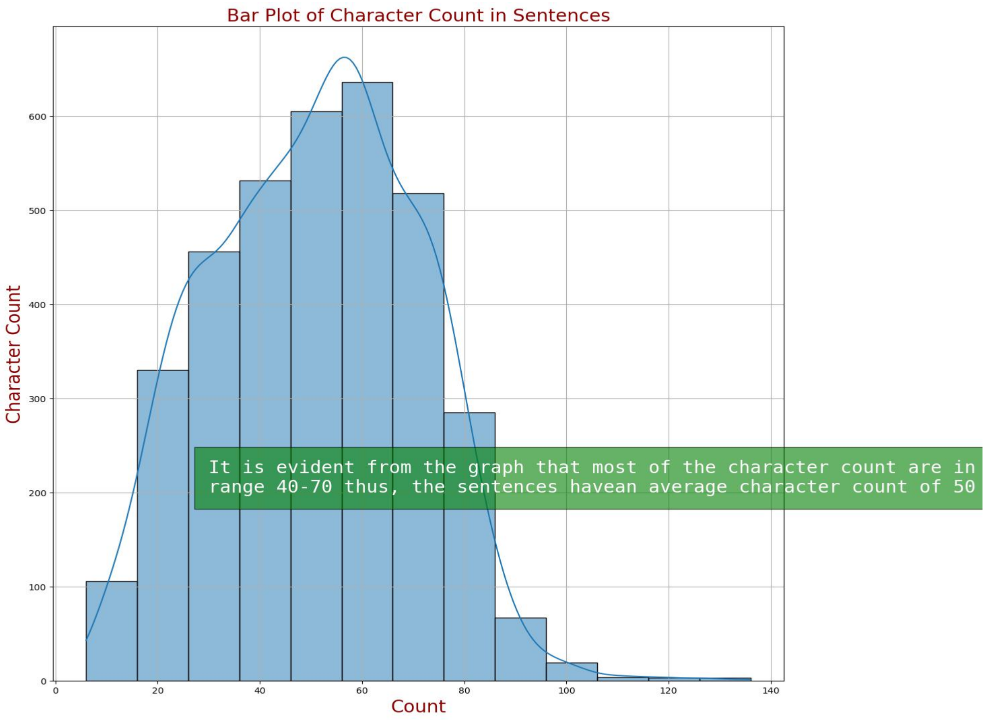


From EDA, we achieved the first step of finding the class imbalance. Later, the word count per sentence and the character count per sentence were analyzed. Properties like word count and character are important as they help in understanding words of which length are more frequent, and later on, using N-gram exploration, we would be able to find which were the words that occur more frequently. Knowing the frequency of a word is important as it can indicate whether that particular word is essential for the context of a sentence.
From 1(a) and 1(b) for task A the average word count per sentence was 81 and the average character count per sentence was 301.
From 2(a) and 2(b) for task B the average word count per sentence was 13 and the average character count per sentence was 50.


Bar Plot of Word Count in Sentences
Word Count
Count
0
1000
2000
3000
4000
5000
200
400
600
800
It is evident from the graph that most of the word count are in range 100-200 thus, the sentences havean average word count of 81


Bar Plot of Character Count in Sentences
Character Count
Count
0
1000
2000
3000
4000
It is evident from the graph that most of the character count are in range 300-400 thus, the sentences havean average character count of 301


Bar Plot of Word Count in Sentences
Word Count
Count
0
100
200
300
400
500
600
0
5
10
15
20
25
30
35
It is evident from the graph that most of the word count are in
range 12-18 thus, the sentences havean average word count of 13


Bar Plot of Character Count in Sentences
Character Count
Count
0
100
200
300
400
500
600
0
20
40
60
80
100
120
140
It is evident from the graph that most of the character count are in
range 40-70 thus, the sentences havean average character count of 50

1.5.1 N-Gram Exploration

N consecutive words occurring in a sentence are called N-Grams. These are N words grouped together when a sentence is broken into parts. For example: In the Sentence

" Today it will rain heavily"

The n-grams will be

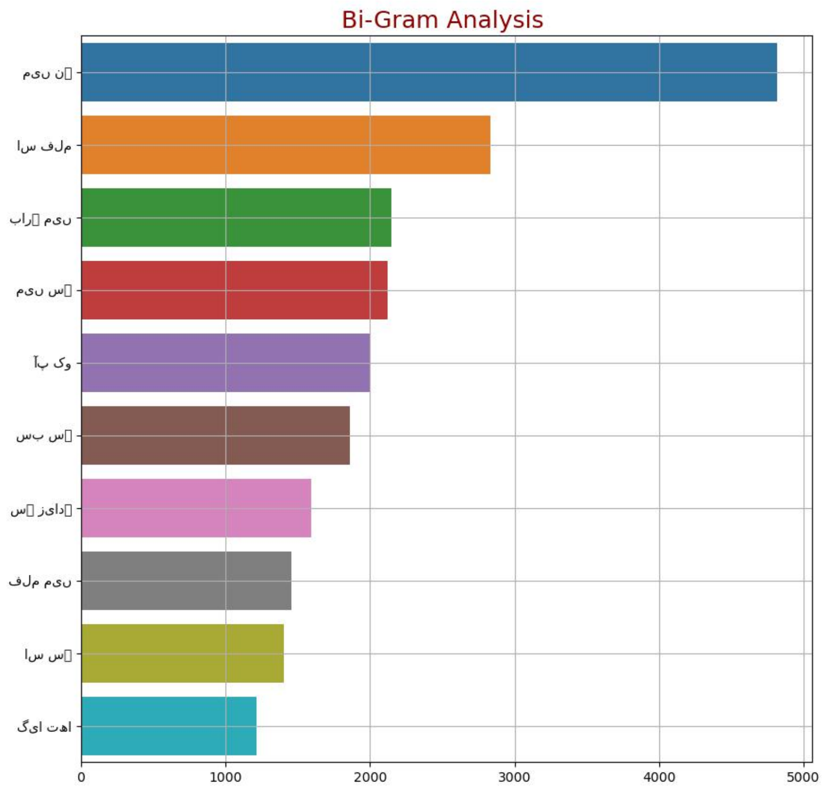
when n=1:
["Today","it","will","rain","heavily"]
when n=2:
["Today it","will rain","heavily"]
And so on...

Based on the value of N, we categorize N-Grams as Unigram (N=1), Bigram (N=2), Trigram (N=3), and so on. For the majority of the NLP tasks, Unigram, Bigram, and Trigram play a vital role and are of significant importance. The use of N-gram helps to represent the text data such that they can be easily analyzed and understood by Machine Learning algorithms. They can also be used for statistical representations. They are used to extract textual features and are deployed in several tasks like emotional classification, language modeling, etc.

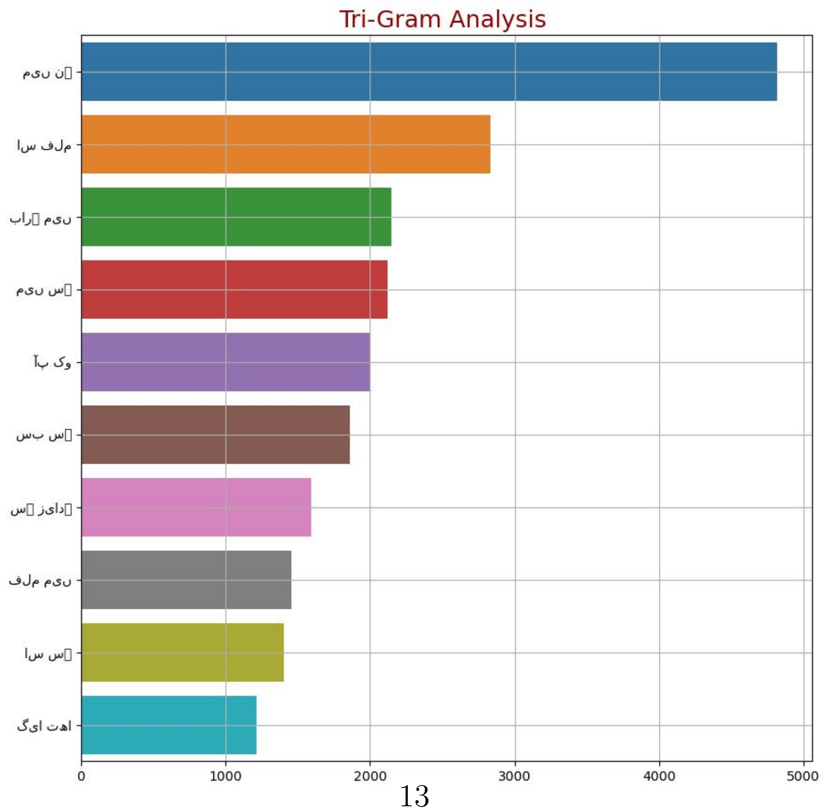
For our task A and B, we carried out Bigram and Trigram Analysis. Specifically for our Task A, we carried out a Chi-Square test to get highly correlated Unigrams and Bigrams for each label. That is we applied the statistical test of chi-square to understand which unigrams are highly correlated to the label "Anger", and which bigrams are highly correlated to the label "Sadness".

Chi-Square Test is used to find the difference between the expected and observed data but it can also be used to get the correlation among the categorical variables. Using the frequencies or term counts we figure out the highly correlated Unigrams and Bigrams for each label.

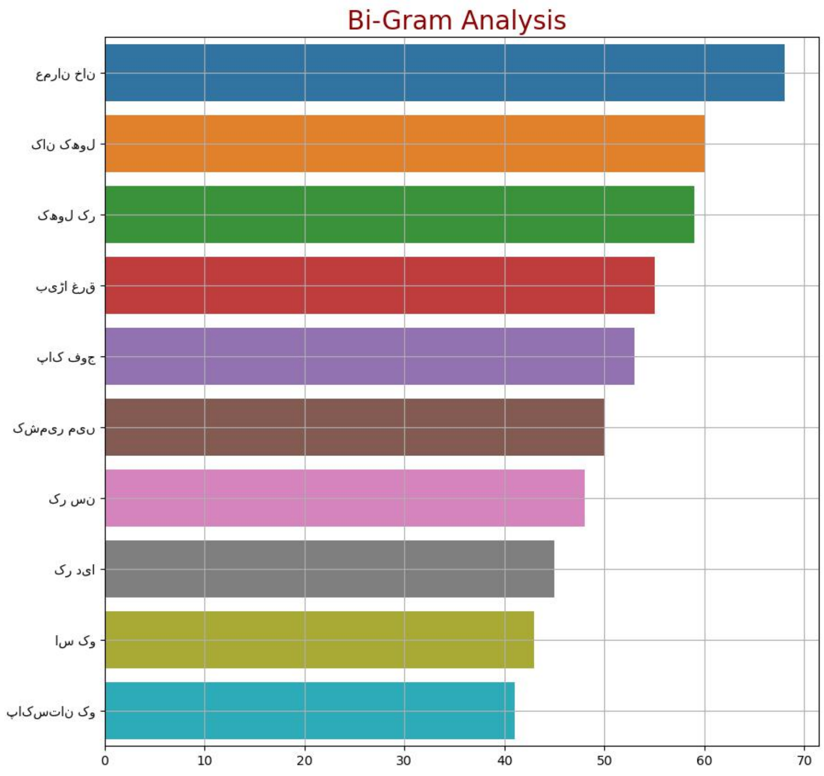
$$\chi^2 = \frac{(Observed - Expected)^2}{Expected}$$



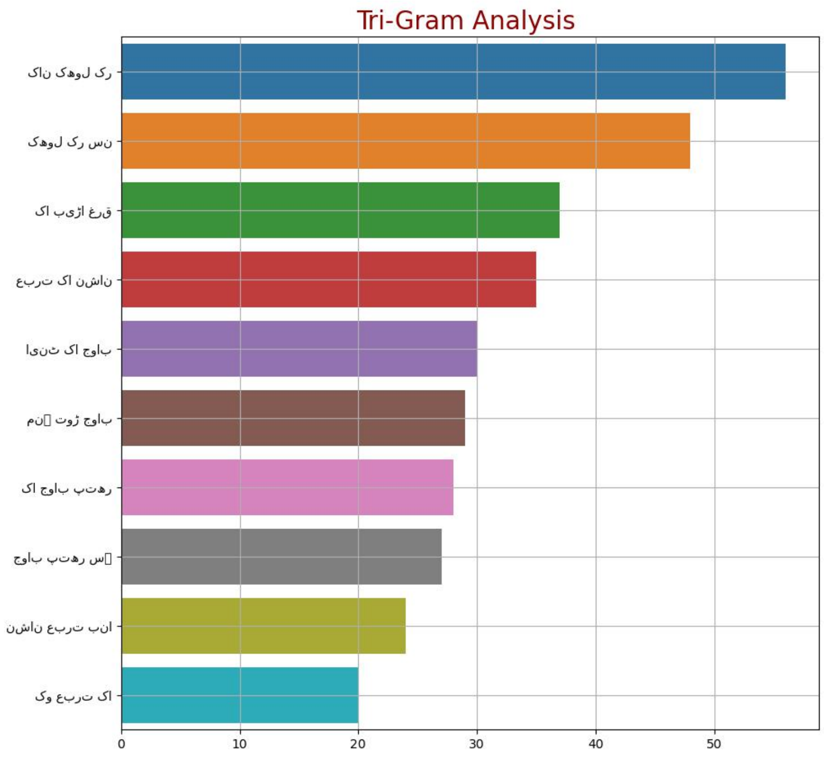
(a) Bigram Analysis Task A



(b) Trigram Analysis Task A



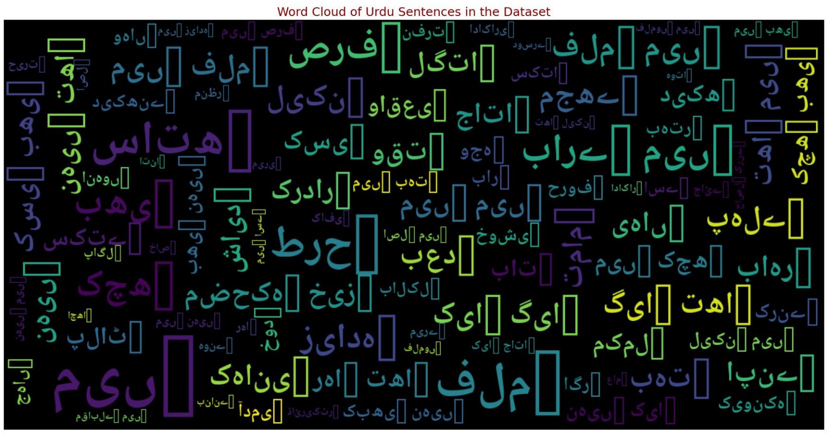
(a) Bigram Analysis Task B



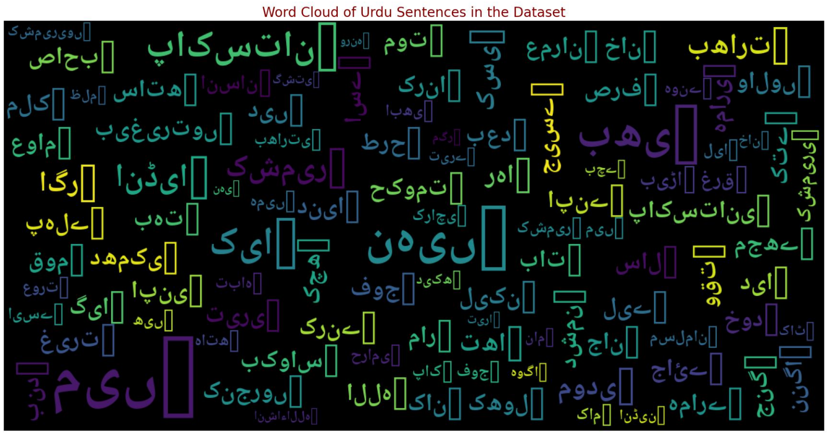
(b) Trigram Analysis Task B

1.5.2 Word Cloud

Word Cloud is one of the most essential Visualization tools in the field of NLP. Word Cloud represents the most frequently used words in a document with different types of colors and font sizes. The smaller the word's font size, the lesser the word's importance in that document. In our task, we created a word cloud of both tasks, A and B. But some words were not understood by python due to their encoding, but since they possessed importance in the context, they were included in the word cloud. Also one can also check that some bigrams and trigrams found during the n-gram exploration have also been included in the word cloud. Note that stopwords can pose a significant obstruction during the creation of a word cloud because they occur very frequently but are not necessary; hence they were removed before the word cloud visualization.



(a) Word Cloud Task A



(b) Word Cloud Task B

1.6 Sentence Embedding

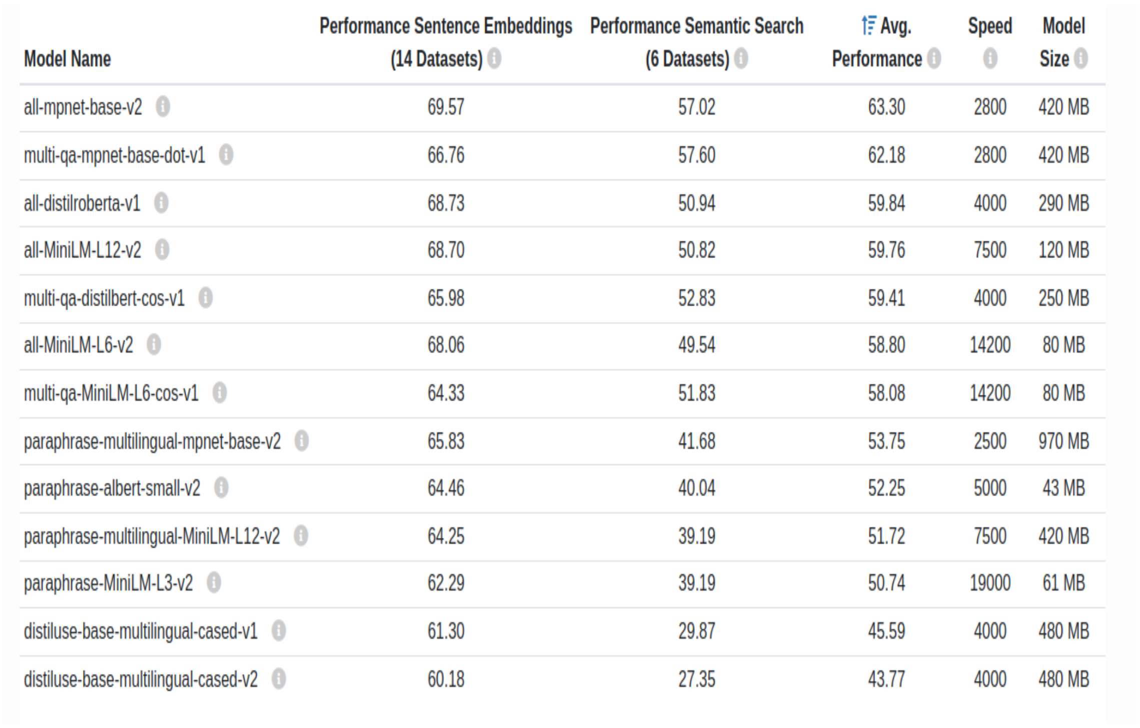
Sentence embedding refers to the process of representing a sentence as a fixed-length vector of numbers. The idea is to capture the meaning of a sentence in a way that machine learning models can easily compare and process. There are various techniques for generating sentence embeddings, but one common approach is to use pre-trained models such as word2vec, GloVe or BERT, which are trained on large amounts of text data. For instance, BERT (Bidirectional Encoder Representations from Transformers) uses a deep neural network with a transformer architecture to generate sentence embeddings. The model is pre-trained on a large corpus of text, which enables it to learn the relationships between words in context.The sentence is first tokenized into a sequence of sub-words to generate a sentence embedding using BERT. Then, the tokens are passed through the BERT model, which produces contextualized embeddings for each token. These embeddings are then combined into a fixed-length vector using a pooling operation, such as averaging or max-pooling, to obtain the sentence embedding.

For our task, we have used Sentence BERT embeddings, particularly sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2, also referred to as SBERT. It is a modification made over BERT, and it uses Siamese and triplet network structures to identify semantically similar sentence embeddings using the cosine similarity rule. Sentence embeddings produced are a multidimensional vector of fixed length, and thus, cosine similarity rules appear to be an efficient metric to find the similarity among sentences.

Cosine similarity assists in determining similarities between two vectors $x\ and\ y$ irrespective of their sizes. Its formula is stated as:-

$$\mathrm{Cos(x,y)} = \frac{x \cdot y}{\|x| \times \|y\|}$$

The significant advantage of using the SBERT was due to its meager time complexity. It has been experimentally found that standard BERT-based encoding used 65 hours of time on Normal V100 GPU for finding two highly similar sentences in a corpus of 10,000 sentences, whereas SBERT, on the other hand, takes only 5 seconds to do so. This motivated me to consider a particular SBERT for our task. Also,sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2 have been trained on over 50+ languages, including Urdu, and give a decent performance in terms of metric and time as compared to other models trained on similar languages and datasets. The following figure shows a comparison between different multilingual SBERT-based embeddings. Note that all the models are not trained in the Urdu language.



Model Name	Performance Sentence Embeddings (14 Datasets)	Performance Semantic Search (6 Datasets)	Avg. Performance	Speed	Model Size
all-mpnet-base-v2	69.57	57.02	63.30	2800	420 MB
multi-qa-mpnet-base-dot-v1	66.76	57.60	62.18	2800	420 MB
all-distilroberta-v1	68.73	50.94	59.84	4000	290 MB
all-MiniLM-L12-v2	68.70	50.82	59.76	7500	120 MB
multi-qa-distilbert-cos-v1	65.98	52.83	59.41	4000	250 MB
all-MiniLM-L6-v2	68.06	49.54	58.80	14200	80 MB
multi-qa-MiniLM-L6-cos-v1	64.33	51.83	58.08	14200	80 MB
paraphrase-multilingual-mpnet-base-v2	65.83	41.68	53.75	2500	970 MB
paraphrase-albert-small-v2	64.46	40.04	52.25	5000	43 MB
paraphrase-multilingual-MiniLM-L12-v2	64.25	39.19	51.72	7500	420 MB
paraphrase-MiniLM-L3-v2	62.29	39.19	50.74	19000	61 MB
distiluse-base-multilingual-cased-v1	61.30	29.87	45.59	4000	480 MB
distiluse-base-multilingual-cased-v2	60.18	27.35	43.77	4000	480 MB

Figure 1.4: Performance of different SBERT

These models find semantically similar sentences within one language or across languages: Semantics similar sentences means sentences with similar meaning in different languages are mapped close to each other in the vector space. The languages they are trained over are 50+ languages: ar, bg, ca, cs, da, de, el, en, es, et, fa, fi, fr, fr-ca, gl, gu, he, hi, hr, hu, hy, id, it, ja, ka, ko, ku, lt, lv, mk, mn, mr, ms, my, nb, nl, pl, pt, pt-br, ro, ru, sk, sl, sq, sr, sv, th, tr, uk, ur, vi, zh-cn, zh-tw. Every two alphabet character is the language code of a particular language, **ur** represents the Urdu language.

1. **distiluse-base-multilingual-cased-v1**: Multilingual knowledge distilled version of multilingual Universal Sentence Encoder. Supports 15 languages: Arabic, Chinese, Dutch, English, French, German, Italian, Korean, Polish, Portuguese, Russian, Spanish, and Turkish.

2. **distiluse-base-multilingual-cased-v2**: Multilingual knowledge distilled version of multilingual Universal Sentence Encoder. This version

supports 50+ languages but performs a bit weaker than the v1 model.

3. **paraphrase-multilingual-MiniLM-L12-v2** - Multilingual version of paraphrase-MiniLM-L12-v2, trained on parallel data for 50+ languages.

4. **paraphrase-multilingual-mpnet-base-v2** - Multilingual version of paraphrase-mpnet-base-v2, trained on parallel data for 50+ languages.

Paraphrase-multilingual-mpnet-base-v2 is a pre-trained transformer-based language model for generating sentence embeddings and detecting semantic similarity between sentences. It is built on top of the multilingual version of the Megatron language model and is trained on a large corpus of text in multiple languages. The architecture of paraphrase-multilingual-mpnet-base-v2 consists of a multi-layer transformer encoder, where each layer has a self-attention mechanism and a feedforward neural network. The self-attention mechanism allows the model to attend to different parts of the input sequence and capture the relationships between words in context. The encoder is pre-trained using a masked language modeling (MLM) objective, where a subset of tokens in the input sequence is masked, and the model is trained to predict the masked tokens. This objective encourages the model to learn a general representation of language that can be used for various downstream tasks, including paraphrase detection and semantic similarity. In addition to the MLM objective, the model is also pre-trained on a next sentence prediction (NSP) task, where the model is trained to predict whether two sentences are adjacent in the input sequence. This objective helps the model capture the relationships between consecutive sentences and learn to generate sentence embeddings that capture the semantic similarity between sentences. Overall, the architecture of paraphrase-multilingual-mpnet-base-v2 is similar to other transformer-based language models such as BERT. Still, it is designed to be multilingual, which makes it suitable for processing text in multiple languages.

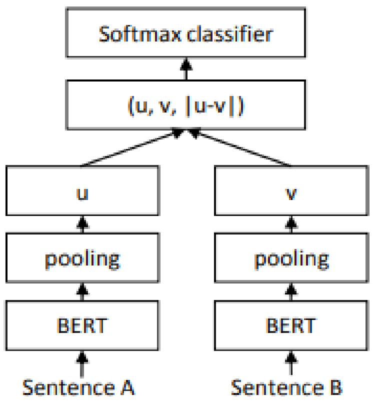


Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

1.7 Evaluation criteria

Task A is a Multilabel classification, and Multilabel classification tasks are those in which each instance may belong to more than one class or label. In such tasks, the evaluation metrics used are different from those used in binary or multiclass classification tasks. Here are some of the commonly used evaluation metrics for multilabel classification tasks:

1. Hamming Loss: Hamming loss measures the fraction of incorrectly predicted labels. The proportion of the labels is mispredicted compared to the total number of labels. Hamming loss values range from 0 to 1, with 0 indicating perfect classification.

2. Precision, Recall, and F1-score: Precision, Recall, and F1-score can also be used for multilabel classification tasks. However, they are calculated separately for each label, and then averaged across all labels using micro or macro averaging. Micro-averaging calculates the metrics for each instance-label pair and then averages them, while macro-averaging calculates the metrics for each label and then averages them.

Task B is binary classification, and there are several evaluation criteria that can be used for text classification tasks, depending on the specific requirements of the task. Here are some of the most common evaluation criteria:

1. Accuracy: Accuracy measures the proportion of correctly classified instances over the total number of instances. It is the most commonly used evaluation metric for classification tasks.

2. Precision: Precision measures the proportion of true positives (i.e., correctly classified positive instances) over the total number of predicted positive instances. Precision is functional when the cost of a false positive is high, such as in medical diagnosis.

3. Recall: Recall measures the proportion of true positives over the total number of actual positive instances. The recall is valid when the cost of a false negative is high, such as in detecting fraud.

4. F1-score: The F1-score is the harmonic mean of precision and recall. It is a single number that balances both precision and recall and is useful when the classes are imbalanced.

5. Confusion matrix: A confusion matrix is a table that summarizes the number of true positives, true negatives, false positives, and false negatives. It provides a more detailed evaluation of the model's performance and can be used to calculate other evaluation metrics such as precision, recall, and F1-score.

6. ROC curve: The Receiver Operating Characteristic (ROC) curve is a graphical representation of the performance of a binary classifier. It plots the actual positive rate against the false positive rate at different classification thresholds and can be used to compare the performance of different classifiers.

Overall, the choice of evaluation metric depends on the specific requirements of the task and the cost associated with different types of errors.

1.8 Label Powerset

Task A is the Multilabel Emotion Classification problem, as discussed earlier. There are various machine learning techniques that can be used to solve a multilabel classification problem. Here are some common approaches:

1. Binary Relevance: This is a simple approach where each label is treated as a binary classification problem. A separate binary classifier is trained for each label. The final output is a binary vector indicating the presence or absence of each label.

2. Label Powerset: This is another approach that transforms a multilabel classification problem into a multi-class classification problem. Each combination of labels is mapped to a unique class, and a multi-class classifier is trained on the transformed data.

3. Classifier Chains: This approach creates a chain of binary classifiers, where the output of one classifier is used as an input for the next classifier. The order of the chain can be determined by domain knowledge or using a search algorithm.

4. Multi-Label k-Nearest Neighbors (MLkNN): This is a variant of k-Nearest Neighbors that is designed for multilabel classification. It selects k nearest neighbors for each instance and predicts the labels based on the labels of the neighbors.

5. Neural Networks: Deep learning models such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have been shown to perform well in multilabel classification problems.

6. Ensemble methods: Ensemble methods such as Random Forests and Boosting can be used to improve the performance of multilabel classification models by combining the predictions of multiple models.

We have decided to use the Label Powerset technique and deployed techniques like Neural Networks to solve this problem. Now the question arises of what the Label Powerset technique is?

So, Label Powerset is a multilabel classification technique that transforms a multilabel classification problem into a multi-class classification problem. It does this by mapping each combination of labels to a unique class. In other words, given a set of possible labels, the Label Powerset technique creates a new set of classes, where each class represents a unique combination of labels. For example, if there are three possible labels (A, B, and C), then there are eight possible combinations of labels: {} {A}, {B}, {C}, {A, B}, {A, C}, {B, C}, and {A, B, C}. The Label Powerset technique would create eight classes, one for each combination. Once the classes have been created, a standard multi-class classifier can be trained on the transformed data. When making predictions on new data, the multi-class classifier outputs a class, which is then mapped back to the original set of labels. One of the advantages of the Label Powerset technique is that it preserves the correlation between labels, unlike other multilabel classification techniques such as Binary Relevance and Classifier Chains. However, it suffers from the curse of dimensionality since the number of possible combinations of labels grows exponentially with the number of labels, leading to a large number of classes and increased computation time.

1.9 Experiments

1.9.1 Classification of text using TF-Idf Vectorization and Machine Learning

TF-IDF (Term Frequency-Inverse Document Frequency) vectorization is a natural language processing technique (NLP) technique to transform text data into a numerical format that machine learning models can use. The idea behind TF-IDF is to represent each document as a vector in a high-dimensional space, where each dimension represents a unique word in the corpus. The importance of the corresponding word in the document determines the value of each dimension.
The "Term Frequency" component of TF-IDF measures the frequency of a word in a document. A word that appears more frequently in a document is considered more critical and will have a higher value in the corresponding dimension of the vector. The result is a value between 0 and 1 that indicates the relative frequency of the term in the document.

$$TF = \frac{i}{d}$$

Where $i =$ "Number of times the term appears in the document;
, and $d =$ " Total number of terms in the document."

The "Inverse Document Frequency" component of TF-IDF measures how rare a word is in the corpus as a whole. Words that frequently appear in many documents are considered less important and will have a lower value in the corresponding dimension of the vector.

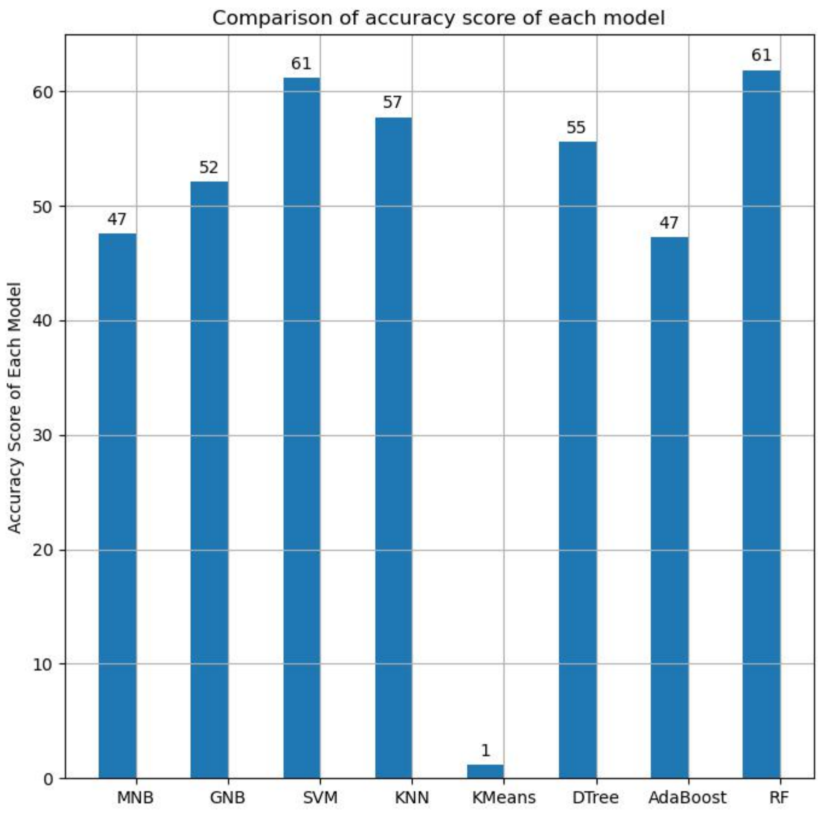
$$IDF = \ln(\frac{N}{df + 1})$$

Where $i =$ "the total number of documents in the corpus;
, and $d =$ " df is the number of documents that contain the term.."

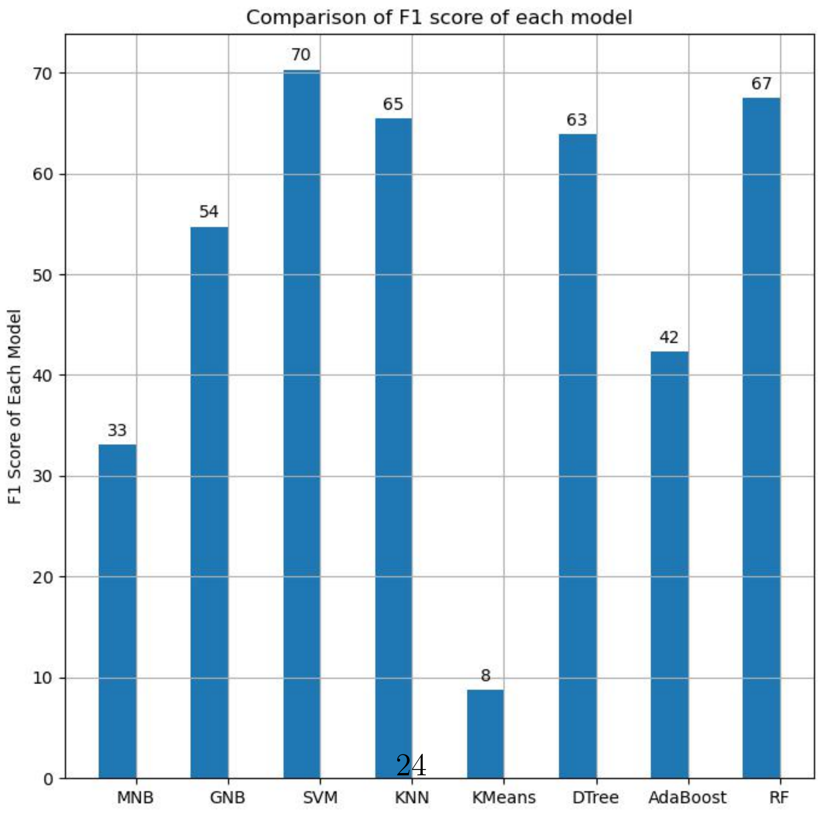
The combination of these two components results in a numerical representation of each document that captures the importance of each word in the document relative to its importance in the corpus as a whole.

$$TF = TF \times IDF$$

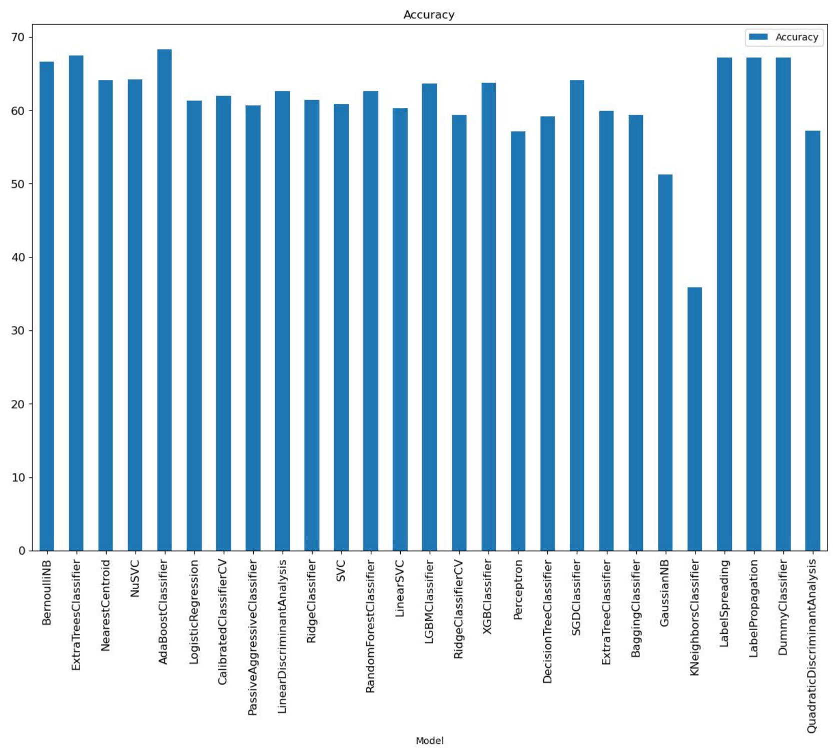
After vectorizing the Urdu text, we fit it into different machine-learning models to get their performance. The best accuracy for Task A that could be achieved using this method was 61, and the best F1- Score measured using this method was 71. Similarly, for Task B subtask 1 the best accuracy and F1-Score gained using this method were 68 and 68, respectively; for Subtask 2 the best accuracy and f1 score obtained were 60 and 56, respectively.



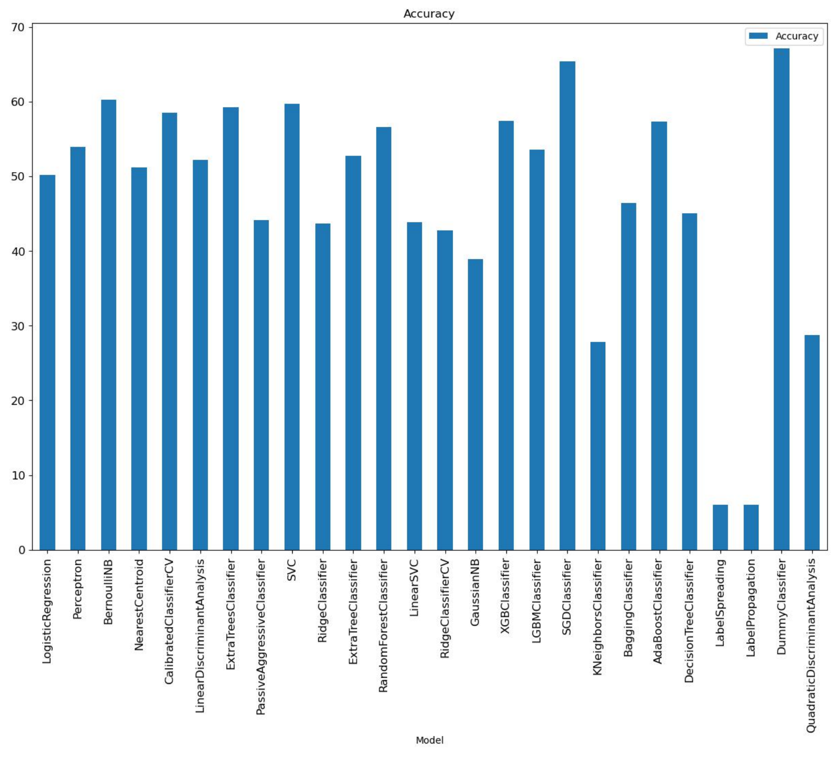
(a) Accuracy of different models for Task A



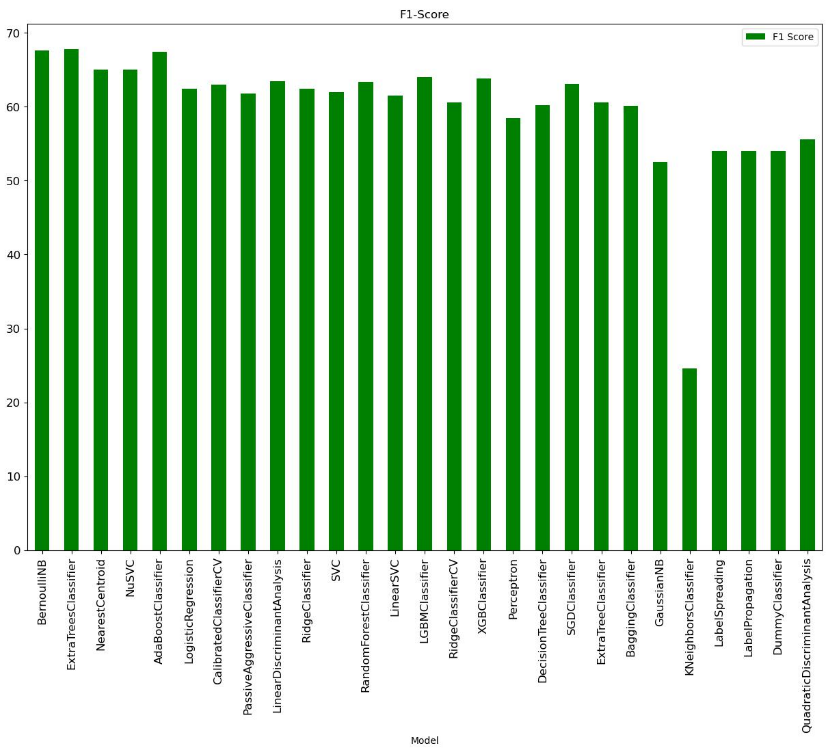
(b) F1-Score for different models for Task A



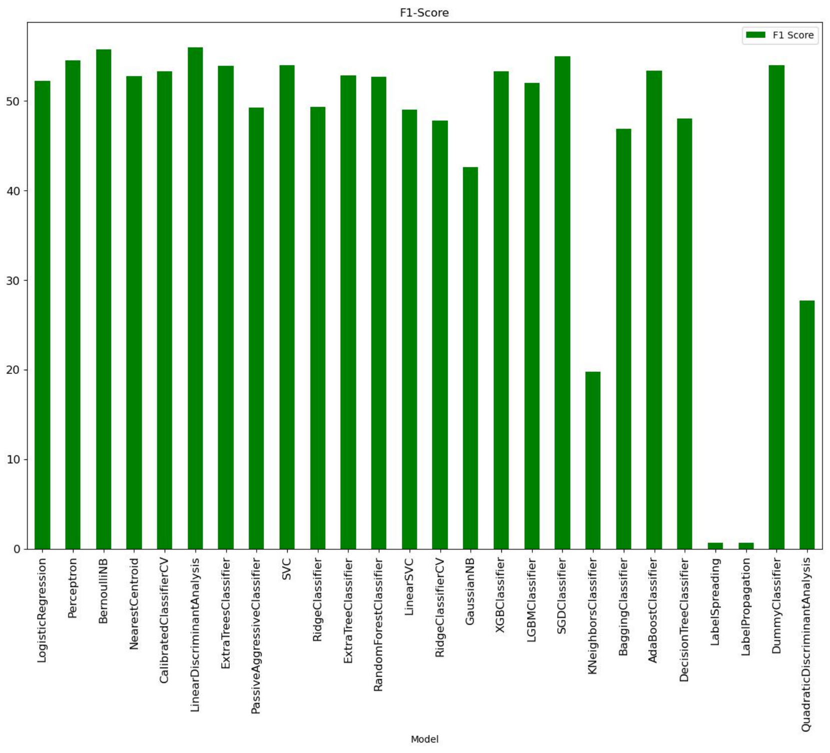
(a) Accuracy of different models for Task B 1



(b) Accuracy for different models for Task B 2



(a) F1-Score of different models for Task B 1



(b) F1-Scoer for different models for Task B 2

1.9.2 Deep Neural Network for Classification

Deep neural networks are a powerful tool for sentence classification. They can learn to automatically extract features from raw text data and make accurate predictions based on those features. However, designing and training a good DNN model was a complex and time-consuming process, requiring careful consideration of factors like model architecture, preprocessing techniques, and hyperparameter tuning. To overcome it, we made use of Optuna for hyperparameter tuning. Optuna is a Python library for hyperparameter optimization that uses the Bayesian optimization technique. Bayesian optimization is a sequential model-based optimization technique that builds a probabilistic objective function model and uses it to guide the search for the best set of hyperparameters. Bayesian optimization aims to find the set of hyperparameters that maximizes the expected improvement over the current best solution.

Here's a high-level overview of how Optuna works for hyperparameter tuning:

1. **Define the objective function**: We define the objective function f(x) as a function of the hyperparameters x. Our goal is to find the set of hyperparameters that maximizes this function.

2. **Probabilistic Model**: We define a probabilistic model of the objective function, which we represent as a Gaussian process. The Gaussian process is characterized by a mean function $\mu(x)$ and a covariance function $k(x, x')$. Given a set of hyperparameters x, the Gaussian process defines a distribution over the possible values of f(x).

3. **Acquisition function**: We define an acquisition function $\alpha(x)$, which represents the expected improvement over the current best solution. The acquisition function is defined as the expected value of the improvement function I(x), which is defined as:

   $$I(x) = max(0, f(x) - f(x\_best))$$

   where x_best is the set of hyperparameters that currently gives the best performance.

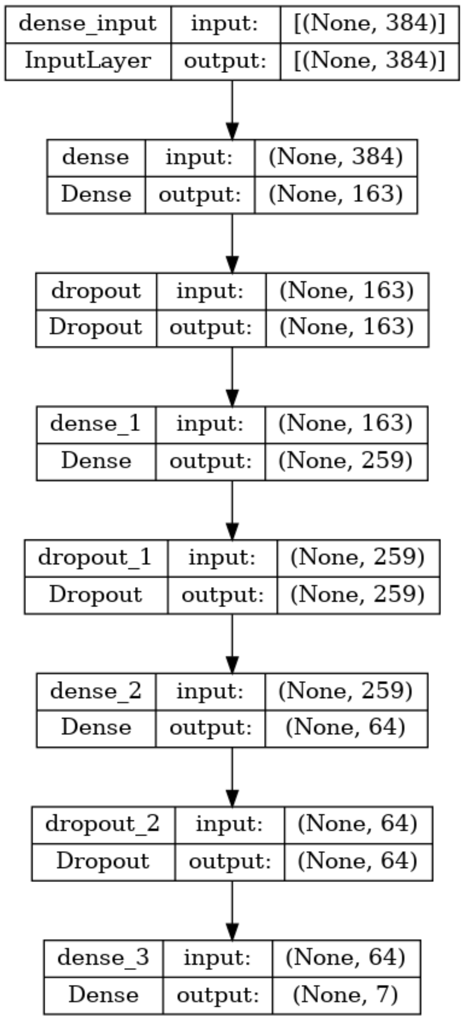
4. **Optimization**: We iteratively sample a new set of hyperparameters x_t by optimizing the acquisition function $\alpha(x)$ using a numerical optimizer. The optimization problem can be formulated as:

   $$x\_t = argmax \quad \alpha(x)$$

5. **Updating the probabilistic model**: After obtaining the new set of hyperparameters x_t, we evaluate the objective function $f(x\_t)$ and update the Gaussian process model using the new data point $(x\_t, f(x\_t))$.

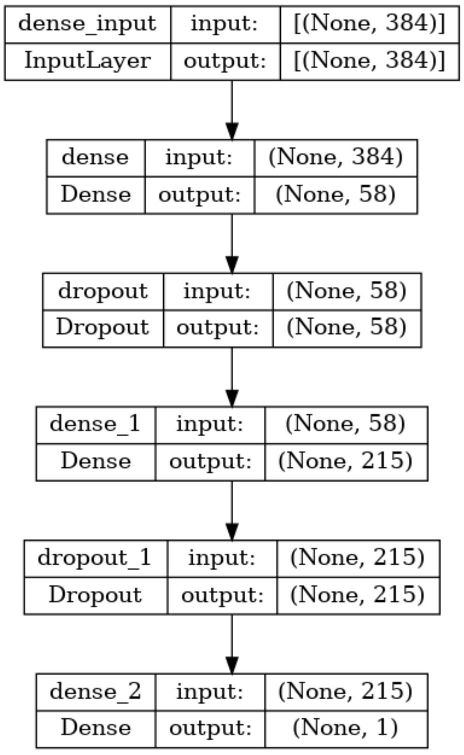
6. **Repeat**: We repeat steps 4 and 5 until a stopping criterion is met (e.g., a maximum number of iterations or a minimum improvement threshold).

Further, using the optuna, we hyper-tuned parameters like the number of hidden layers, the number of units in the hidden layer, the activation function of the hidden layer, and the dropout value. The hyper-tuned model was



dense_input | input: | [(None, 384)]
InputLayer | output: | [(None, 384)]
dense | input: | (None, 384)
Dense | output: | (None, 163)
dropout | input: | (None, 163)
Dropout | output: | (None, 163)
dense_1 | input: | (None, 163)
Dense | output: | (None, 259)
dropout_1 | input: | (None, 259)
Dropout | output: | (None, 259)
dense_2 | input: | (None, 259)
Dense | output: | (None, 64)
dropout_2 | input: | (None, 64)
Dropout | output: | (None, 64)
dense_3 | input: | (None, 64)
Dense | output: | (None, 7)

Figure 1.8: Model Architecture for task A



dense_input | input: | [(None, 384)]
InputLayer | output: | [(None, 384)]
dense | input: | (None, 384)
Dense | output: | (None, 58)
dropout | input: | (None, 58)
Dropout | output: | (None, 58)
dense_1 | input: | (None, 58)
Dense | output: | (None, 215)
dropout_1 | input: | (None, 215)
Dropout | output: | (None, 215)
dense_2 | input: | (None, 215)
Dense | output: | (None, 1)

Figure 1.9: Model Architecture for task B

further tuned using the K-fold Cross-validation, and the final tuned model was used for prediction. We used "Adam" as an optimizer that allowed the model to converge efficiently in terms of time taken. We also used the early callbacks that prevented the model from overfitting. We also made use of ReducePlateau, which monitored the validation loss and kept updating the learning rate of the model by a factor of 0.1, allowing the model not to overshoot the minimum and converge to a better minimum. The F1 score for task A achieved was 68, and for task B subtask 1, the accuracy achieved was 72 with a ROC AUC score of 69. The input features for this experiment were sentence embeddings generated earlier.

1.9.3 Ensemble Method for classification

Ensemble methods for classification involve combining the predictions of multiple individual classifiers in order to produce a final prediction. The idea behind ensemble methods is that by combining the predictions of multiple classifiers, we can improve the overall performance of the classification system.

There are several different types of ensemble methods for classification, but one of the most common is the "majority voting" approach. In this approach, we train multiple classifiers on the same dataset using different models or different hyperparameters. Then, we take the majority vote on the individual classifiers' predictions at prediction time. For example, if we have three classifiers and they predict "class 1", "class 1", and "class 2", respectively, then the majority vote would be "class 1", and that would be our final prediction. We have used Majority voting for classification tasks. We have used a common ensemble model for both tasks.
There were two instances where the ensemble machine-learning model was deployed. The first being when the input was Tf-Idf vectorization of Urdu text, which gave a performance of accuracy of 59.48 and F1-Score of 67.05 for task A, and it gave an accuracy of 68.88 and F1-Score of 69.50 for subtask 1 of Task B; also, this method helped gain accuracy of 54.22 and F1-Score of 52.65 for the subtask 2 of Task B. The second was when the input sequence was given as sentence embedding. This variation gave an accuracy of 58.88 and an F1-Score of 68.96 for task A. For subtask 2 of task B, this variation gave an accuracy of 57 and an F1-Score of 31.89; for subtask 1 of task B, this variation gave an accuracy of 71 and an f1-Score of 71.52.

1.9.4 Text Classification using N-gram feature

This technique was deployed exclusively for the subtask of task B. Here we created five feature combinations of the N-grams, unigram, unigram_bigram, bigram, bigram_trigram, and trigram. For every combination of the features, we evaluated different machine-learning models to evaluate the result. Later, we used Halv Grid Search CV to hyper-tune the models and then used that model to get the test scores.
Halving Grid Search CV is a machine learning algorithm for hyperparameter tuning that reduces the number of hyperparameter combinations that need to be evaluated at each iteration. The algorithm selects a random subset of hyperparameters and evaluates their performance on a validation set. Based on this performance, the algorithm discards the worst-performing hyperparameters and selects the best-performing ones for the next iteration. The process is repeated until a single set of hyperparameters is selected as the best. The steps to perform Halving Grid Search Cross Validation are :-

1. Define the hyperparameter space: Define a set of hyperparameters, H, and their corresponding ranges of values, r_i. Select a set of candidate values for each hyperparameter that covers the range r_i.

$$H = \{h\_1, h\_2, ..., h\_n\}$$

r_i = [a_i, b_i]

H_i = {v_1, v_2, ..., v_m}

h_i $\in H\_i, where \;\; v\_j \in [a\_i, b\_i]$

2. Initialize the search: Choose a small fraction of the hyperparameter combinations to evaluate in the first iteration.

$$S\_1 = \{(h\_1, h\_2, ..., h\_n)\_1, (h\_1, h\_2, ..., h\_n)\_2, ..., (h\_1, h\_2, ..., h\_n)\_s\}$$

3. Evaluate the model: Train the model using each hyperparameter combination in S_1 and evaluate its performance on a validation set.

E_1 = {e_1, e_2, ..., e_s}

e_i = f(S_1(i))

f(S_1(i)) is the validation error of the model trained using the hyperparameter combination S_1(i)

4. Discard the worst-performing hyperparameters: Select the best-performing hyperparameters from S_1 based on their validation performance. Discard the worst-performing hyperparameter

$$\text{S\_2} = \{\text{S\_1(i) — e\_i} \leq e\_t\}$$

e_t is a threshold value that selects the top-performing hyperparameters

5. Iterate: Repeat steps 2-4 on a larger subset of the training data using the selected hyperparameters. At each iteration, the number of hyperparameter combinations is halved.

$$S\_k + 1 = \{S\_k(i) \;\; such \;\; that \;\; e\_i \leq e\_t\}$$

S_k+1 is a random subset of S_k that contains only the best-performing hyperparameters from S_k

$$\text{e\_k+1} = \{\text{f(S\_k+1(i))} \quad \text{such} \quad \text{that S\_k+1(i)} \in S\_k + 1\}$$

e_k+1 is the validation error of the model trained using the hyperparameter combination S_k+1(i)

k = k+1

6. Select the best model: Once all hyperparameter combinations have been evaluated, select the best model based on its performance on the validation set.

$$S\_final = \{S_k(i) | e_i \leq e_t\}$$

best_hyperparameters = argmin(e_final)

e_final is the validation error of the model trained using the hyperparameter combination S_final(i)

The Halving Grid Search CV was used because this algorithm reduces the number of hyperparameter combinations that need to be evaluated at each iteration, thereby reducing the overall training time and computational resources required for hyperparameter tuning. Task B performance are: {Subtask 1}: Accuracy: 69.51 , F1- Score: 49.03 , ROC_AUC: 60.61, Best_Feature: Unigram_Bigram {Subtask 2}: F1- Score: 69.99, Best_Feature: Unigram

1.9.5 Multilabel Emotion Classification using Bag Of Words

Bag of Words (BoW) is a technique used in Natural Language Processing (NLP) to represent text data as numerical vectors. We used it to perfrom the multilabel classification. To develop Bag of Words we followed the following steps:-

1. Create a Vocabulary: The first step is to create a vocabulary of all unique words in your text corpus. This is done by iterating over all the documents in your corpus and adding each unique word to the vocabulary. Let's represent the vocabulary as a set of words:

2. Count Word Occurrences: For each document in your corpus, count the number of times each word appears. Let's represent the count of word w in document d as count(d, w).

3. Create BoW Vectors: Once you have counted the occurrences of each word in each document, you can create a BoW vector for each document. The BoW vector is a vector of length N, where N is the size of the vocabulary, and each element in the vector represents the count of a particular word in the document. Let's represent the BoW vector for document d as bow(d).

4. Initialize the BoW vector to all zeros: $bow(d) = [0, 0, ..., 0]$

5. For each word w in document d, increment the count of w in the BoW vector: $bow(d)[i] = count(d, word\_i)$

The results achieved for this are Accuracy: 54.46, F1-score macro: 60.27, F1-score micro: 69.56, F1-score weighted: 69.43, Hamming Loss: 0.110

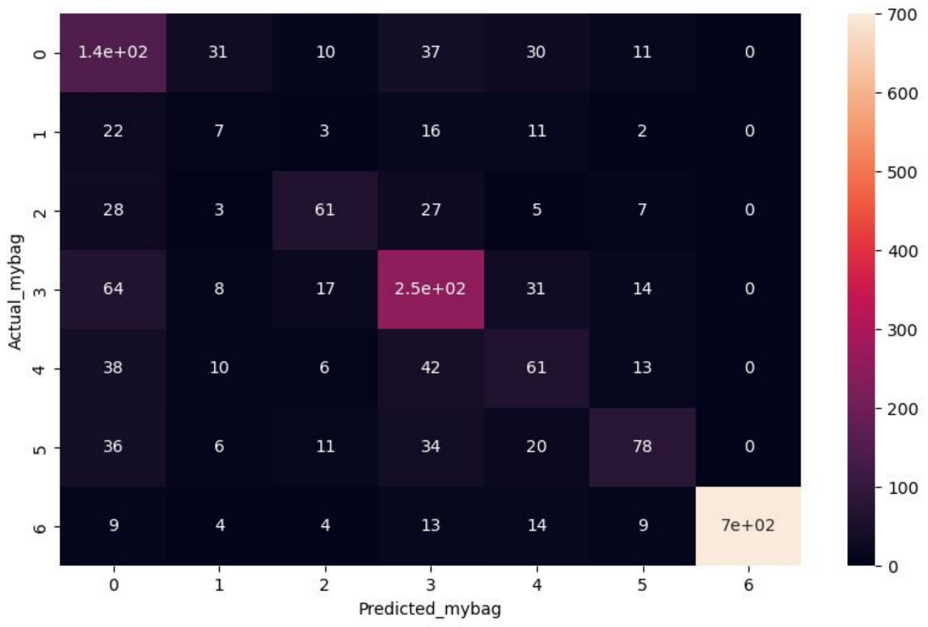


Figure 1.10: Confusion Matrix

1.9.6 LSTM and CNN For Task A

LSTM (Long Short-Term Memory) models for text classification are a type of deep learning architecture that is used for processing sequential data, such as text. These models are capable of capturing the long-term dependencies in the text data and have been shown to achieve state-of-the-art performance on a variety of text classification tasks.

In an LSTM model and 1D CNN model for text classification, the input text is typically first preprocessed and tokenized, and then transformed into a numerical representation, such as word embeddings. The LSTM layer(s) then take in the sequence of word embeddings and process them, maintaining an internal memory state that allows them to capture long-term dependencies in the text.

The output of the LSTM layer(s) is then passed to one or more fully connected layers, which are used to produce the final classification output. The fully connected layers typically apply a nonlinear activation function, such as a sigmoid or softmax function, to the output of the LSTM layer(s) to produce a probability distribution over the possible classes.

The output of the convolutional layer(s) is then passed to one or more pooling layers, which are used to reduce the dimensionality of the feature maps. The pooled output is then flattened and passed to one or more fully connected layers, which are used to produce the final classification output. One advantage of LSTM models for text classification is that they can handle variable-length inputs, which is important in text classification, where the length of the text may vary widely. They are also able to capture contextual information in the text, which is important for many classification tasks where the meaning of the text is heavily dependent on the context.

One advantage of 1D CNN models for text classification is that they can capture local patterns in the text, such as n-grams or short phrases, that are important for classification. They are also relatively computationally efficient and can be trained on large datasets.

Typically, the input to the model is a sequence of vectors or embeddings, and the output is a set of predicted labels or class probabilities. The model is trained using a combination of supervised learning techniques, such as backpropagation through time, and optimization methods, such as gradient descent. We used "Adam" as an optimizer and early callbacks to prevent the model from overfitting.

We separately first deployed LSTM and 1D CNN for the classification task, and they were able to achieve an accuracy of 68.36 and 65.18 respectively, also the obtained F1-Scores were 67 and 63, respectively.

Chapter 2

Conclusion

The experimentation for the following project was rigorous and involved various techniques, from developing the machine learning classifiers to training the deep neural network. Also, hybrid models like LSTM-CNN, DNN with machine learning classifiers, and the ensemble of multiple weak machine learning classifiers were deployed to get the best results possible. For all three tasks, accuracy could be achieved more significantly than the results displayed over the EmoThreat@FIRE 2022 website, and the F1-Score generated could not be obtained the best. However, the scope of improvements is still present in terms of using more complex and deep models, and better sentence embeddings if found. One can also, use some standard scaling over the data or maybe use character-level features to improve the performance of classifiers.

Acknowledgment

I want to extend my sincere regards and a thanks note to Dr. Kirti Kumari, under whose supervision this project was carried out. Her sincere guidance and assistance made it possible for me to complete the project. Her innovative way of sharing and imparting knowledge has given me deep insights into the theoretical and practical aspects of the field. Her constructive criticism and support made it possible for me to reach this far and get the results. Her feedback and insights have helped me develop my strengths and improve my weaknesses, and I feel more confident in my abilities. I will carry the knowledge and experience that I have gained during my internship with me throughout my career.

References

1. https://www.kaggle.com/datasets/rtatman/urdu-stopwords-list
2. https://github.com/Xangis/extra-stopwords/blob/master/README.md
3. https://github.com/frappe/fonts/blob/master/usr$_{s}hare_{f}onts/noto/NotoKufiArabic-$ $Regular.ttfhttps : //towardsdatascience.com/multi - class - text - classification - with - scikit - learn - 12f1e60e0a9f$
4. https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17
5. https://github.com/anuragshas/nlp-for-urdu/blob/master/classification/Urdu$_{C}lassification_{M}o$ $//analyticsindiamag.com/how - to - use - graph - neural - networks - for - text - classification/$
6. https://github.com/yao8839836/text$_{g}cnhttps : //github.com/harshgeek4coder/MultiClass - NLP - Classification - Hybrid - Neural - Models - Analysis - with - Optimizers$
7. https://www.analyticsvidhya.com/blog/2022/01/multi-label-text-classification-using-transfer-learning-powered-by-optuna/
8. https://scikit-learn.org/stable/modules/multiclass.html
9. https://stackabuse.com/python-for-nlp-multi-label-text-classification-with-keras/
10. https://neptune.ai/blog/exploratory-data-analysis-natural-language-processing-tools
11. https://www.youtube.com/playlist?list=PLoROMvodv4rPLKxIpqhjhPgdQy7imNkDn
12. https://www.youtube.com/playlist?list=PLyqSpQzTE6M$_{E}cNgdZ2qOtTZe7YI4Eedbhttps : //towardsdatascience.com/multi - class - text - classification - with - lstm - 1590bee1bd17$
13. https://towardsdatascience.com/preprocessing-text-in-python-923828c4114f
14. https://medium.com/analytics-vidhya/tweet-analytics-using-nlp-f83b9f7f7349

15. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9044368/
16. https://stackoverflow.com/questions/52490708/how-to-apply-regular-expression-in-urdu-text-using-python
17. https://stackoverflow.com/questions/33404752/removing-emojis-from-a-string-in-python
18. https://huggingface.co/sentence-transformers/stsb-xlm-r-multilingual
19. https://github.com/google-research/bert/blob/master/multilingual.md
20. https://www.analyticsvidhya.com/blog/2021/09/optimize-your-optimizations-using-optuna/
21. https://optuna.org/
22. https://broutonlab.com/blog/efficient-hyperparameter-optimization-with-optuna-framework
23. https://neptune.ai/blog/optuna-vs-hyperopt
24. https://engineering.fb.com/2018/01/24/ml-applications/under-the-hood-multilingual-embeddings/
25. https://nlp.johnsnowlabs.com/2020/12/01/urduvec$_1 40M_3 00d_u r.htmlhttps : //ai.facebook.com/blog/introducing - many - to - many - multilingual - machine - translation/$
26. https://github.com/facebookresearch/fairseq/tree/main/examples/m2m$_1 00https : //keras.io/examples/nlp/multi_label_classification/$
27. https://urduhack.readthedocs.io/en/stable/reference/normalization.html: :text=The
28. https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
29. https://nlp.stanford.edu/IR-book/html/htmledition/dropping-common-terms-stop-words-1.html