Final Project Report

UNIVERSITY OF HERTFORDSHIRE

Department of Computer Science

MSc Data Science and Analytics

7COM1075 - Data Science and Analytics Masters Project

7th May 2021

Machine Learning Sentiments on Urdu Text

Student Name: Muhammad Emaaz

Student ID: 19023610

Supervised by: Dr. Na Helian

Abstract

Sentiment Analysis (SA) using text documents is a mature technique for detecting customer's sentiments by analyzing feedback, reviews, or comments. The amount of data being generated in today's era is enormous and keeping track of this information is getting tricky. Despite being a conventional technique, sentiment analysis lacked in providing its feature to multilingual languages, i.e., Urdu Language, which has 300 million-plus speakers worldwide, which drew the attention of many researchers in the past. SA has gained a significant amount of consideration; however, the frameworks and resources constructed for this technique have benefitted English and other western languages more than the languages that cannot be considered rich languages. In this research, SA has been applied on a data set of 50 thousand annotated Urdu movie revies using multiple feature extraction techniques such as TF-IDF, Bag of Words, and Word2Vec (Word to Vector). These feature extraction techniques were later used in building a machine learning model, including Support Vector Machine, Decision Tree, Logistic Regression and addition of Deep Learning Model, Long Short-Term Memory has also been introduced. These multiple feature extraction techniques and machine learning models were compared in Accuracy, Precision, Recall, F1-Score, Confusion Matrix, and ROC curve for graphical representation. The research findings show that the LSTM model using word2vec outperformed the other algorithms by generating the highest accuracy of 0.8794%, followed by SVM using TF-IDF which generated the second-best results and an accuracy of 0.8092% at cross-validation of 10 k-folds. This research paper presents a comprehensive comparison of the methods used throughout the project for sentiment analysis on Urdu movie reviews.

Keywords – Sentiment Analysis, Deep Learning, Machine Learning, Support Vector Machine, Decision Tree, Logistic Regression, TF-IDF, BoW, Word2Vec, Natural Language Processing

Acknowledgment

First and foremost, I would like to thank God Almighty for showing his blessings to complete my research work successfully.

I want to thank my project supervisor **Dr. Na Helian** and module leader **Dr. Helen Xiang** whose constant guidance and support kept me motivated and showed me the desired direction throughout the project. Their vision and sincerity have deeply inspired me.

In addition, I would like to thank my parents, friends, and family, whose continuous support, love, and care helped me in my difficult times and motivated me during COVID-19 times. Lastly, I would like to thank the University of Hertfordshire for giving me this opportunity of studying at their institute and polishing my skills and preparing me to enter the world of Data Science.

MSc Final Project Declaration

This report is submitted in partial fulfillment of the requirement for the degree of Master of Science in Data Science and Analytics at the University of Hertfordshire (UH).

It is my own work except where indicated in the report.

I did not use human participants in my MSc Project.

I hereby give permission for the report to be made available on the university website provided the source is acknowledged.

Table of Contents

1. Introduction 1
1.1 Background 1
1.2 Research Question 2
1.3 Hypothesis 2
1.4 Aims and Objectives 3
2. Literature Review 3
3. Methodologies 8
3.1 Dataset and Code 9
3.2 Preprocessing 9
3.3 Feature Extraction Techniques 12
3.4 Classification Models 14
3.5 Evaluation/Performance Measures 18
3.6 Legal and Ethical Issues 20
4. Classification Algorithms and Feature Extraction 21
4.1 Feature Extraction 21
4.2 Algorithms 23
5. Implementation 26
5.1 Importing Libraries and Preprocessing 26
5.2 Implementing Feature Extraction Techniques 27
5.3 Implementing Algorithms 30
5.4 Evaluation of Implemented Algorithms 35
5.5 Problems during Implementation 36
6. Experimental Design and Results 38
6.1 Experiment 1 – Evaluating Feature Extraction Techniques on different Classification Models 38
6.2 Experiment 2 – Experimenting SVM based feature extraction with different parameters of cost on TD-IDF and BoW 43
6.3 Experiment 3 – Experimenting Deep Learning Model with Word2Vec Technique 45
7. Conclusion and Future Work 49
References 50
Appendix A 52
Appendix B 71
Appendix C 75
Appendix D 78
Appendix E 78

List of Figures

Figure 1 Sentiment Analysis System Design 8
Figure 2 Urdu Text Stop words list 10
Figure 3 Code Snippet for Urduhack Lemmatizer 11
Figure 4 Lemmatization example for word جاگیر (Jagir, Land (Ali Daud, 2016)) 11
Figure 5 Tokenized form of Reviews 12
Figure 6 TF-IDF Calculation (Stecanella, 2019) 13
Figure 7 Steps for Bag of Words (Anon., 2018) 13
Figure 8 Working of Word2Vector (Nicholson, n.d.) 14
Figure 9 Optimal Separation Hyperplane between two classes (Reddy, 2018) 15
Figure 10 Working of LSTM (Anon., 2015) 16
Figure 11 Step 1 LSTM (Anon., 2015) 16
Figure 12 Step 2 LSTM (Anon., 2015) 17
Figure 13 Step 3 LSTM (Anon., 2015) 17
Figure 14 Step 4 LSTM (Anon., 2015) 17
Figure 15 Experimental Design Flow (1) 19
Figure 16 Experimental Design flow (2) 20
Figure 17 Fifty Thousand Positive and Negative Reviews 26
Figure 18 Code Snippet for Importing Libraries and Preprocessing (1) 26
Figure 19 Code Snippet for Importing Libraries and Preprocessing (2) 27
Figure 20 Equation for TF 27
Figure 21 Equation for IDF 27
Figure 22 Code Snippet for TF-IDF Model (sklearn) 28
Figure 23 Code Snippet for BoW Model 28
Figure 24 Code Snippet for Word2Vec Model 29
Figure 25 Code Snippet for Decision Tree Model 31
Figure 26 Logistic Regression Hypothesis Equation 31
Figure 27 Logistic Regression Cost Equation 32
Figure 28 Code Snippet for Logistic Regression 32
Figure 29 Code Snippet for SVM model 34
Figure 30 Code Snippet for LSTM model 35
Figure 31 Code Snippet for Classification Report / Evaluation 36

List of Tables

Table 1 Comparison of past papers 5
Table 2 Evaluation of classification models using TF-IDF (Without Tuning) 39
Table 3 Evaluation of classification models using TF-IDF (With Tuning) 40
Table 4 Evaluation of classification models using BoW (Without Tuning) 41
Table 5 Evaluation of classification models using BoW (With Tuning) 42
Table 6 Cost value optimization for SVM based Feature Extraction 44
Table 7 Evaluation of LSTM using Word2Vec Feature Extraction 45
Table 8 Detailed Long-Short Term Memory Summary Table 47

1. Introduction

1.1 Background

In today's world, roughly 6,500 languages are being spoken. English, one of which is a widely spoken language with up to 379 million native speakers. A notable amount of work has been conducted in natural language processing while focusing on western languages, unlike south Asian languages, e.g., Urdu. The study currently done on this common resource language is very limited and challenging. The current techniques which can perform well in English will not necessarily generate better results in other languages. The Urdu language has gained much attention from NLP researchers. The Urdu language is India and Pakistan's most widely spoken language, with 300 million-plus speakers worldwide (Che2, 2017). It is essential to focus on working with different languages as they represent different cultures and norms. It can open doors of opportunity for all those working in Data Science who are not familiar with other languages and feel comfortable working in their native language. This research aims to highlight the work done in this subject area and practically implement machine learning algorithms and analyze the outcomes to explain how much potential this language consists of and how well modern world algorithms can support the Urdu language. The prime focus will be on performing Sentiment Analysis on the Urdu text, and through our results, we will support the research question and hypothesis.

In Data Mining, Sentiment Analysis, or Opinion mining are the Natural Language Processing approaches (NLP). It is required to detect the polarity of a sentence (negative/positive/neutral). SA is mainly performed on textual data such as documents, text data, or review datasets. SA can be applied on different platforms to detect the customer's sentiment in feedback or comment. It is being used on different social media platforms (Facebook, Twitter, Instagram) to detect users' sentiments. It can help you make quick decisions by understanding the overall opinion of the user/customer. The data we use in today's world is 90% unstructured and unorganized. A substantial amount of data is generated regularly in an unstructured form; all of this incoming data is collected from different platforms such as social media, surveys, articles, documents, etc. The overall advantage of sentiment analysis consists of Data sorting at scale, Real-time analysis, and consistent criteria. The working of sentiment analysis can be described as compliments Natural language processing and machine learning algorithms to determine the tone behind any sentence, comment, and review. There are three different approaches in which sentiment analysis can fall, Rule-based approach (stemming, tagging, part-of-speech tagging, and parsing), Automatic approaches, and hybrid approaches (Anon., n.d.)

The Urdu language consists of unique features, and for that reason, additional efforts are required to process this language. One of the most important things to learn is that Urdu language script is written from right-to-left unlike traditional writing script from left-to-right. As the shape of the position is changed, it changes the shape of alpha bits. Stop words in the Urdu language are also different and challenging (Khairullah

Khan, 2018). Preprocessing of the Urdu language can get tricky due to the nature of the language. The Urdu language uses a different writing style called 'Nastaliq,' which follows an Arabic script. This writing style is very context sensitive. Due to this sensitivity, this language's characteristics are written in ligatures; these ligatures can use a single graphemes/character or a group of different characters to form a word. Characters in the Urdu language acquired different shapes that depend on the position in the ligature (survey, 2017)

In this project, the same path has been followed as the previous researchers. However, the dataset that has been used is of 50K movie reviews in Urdu, which is labeled with (Positive =1 and Negative = 0); due to using a fair amount of dataset, we can assume that our classification models can perform better than the previous approaches. The machine learning algorithms used for modeling are SVM, Decision Tree, and Logistic Regression. The feature extraction techniques that are used are TF-IDF, Bag of Words, and Word2Vector. We will experiment with different techniques and algorithms to identify which model can provide us the best accuracy and be a perfect fit for the Sentiment analysis on the Urdu text.

Major sections that will be showcased in this report are as follows:

1. The "Research Question" and "Aims and Objectives" will be highlighted in sections 1.2 and 1.3, respectively.
2. An explanation of the literature review performed will be shown in section 2.
3. A description of the methodology implemented for the competition of the project will be shown in section 3.
4. A detailed explanation of the machine learning algorithms and feature extraction techniques will be shown in section 4.
5. Implementation and evaluations performed will be shown in section 5.
6. Results and experiment details will be shown in section 6.
7. The conclusion of the project will be shown in section 7.

1.2 Research Question

RQ 1: *How does NLP impact sentimental analysis on Urdu text?*

RQ 2: *Which ML model(s) is most suitable for sentimental analysis on Urdu text?*

1.3 Hypothesis

Null Hypothesis (H_0): Many of the ML/NLP techniques cannot be applied on Urdu text.

Alternative Hypothesis(H_a): Many of the ML/NLP techniques can be applied on Urdu text.

1.4 Aims and Objectives

Aim: The Project aims to apply different Machine Learning (ML) Algorithms/Techniques such as Natural Language Processing (NLP) on the Urdu Language (Official language of Pakistan) for sentiment analysis.

Objectives:

1. To deeply understand the scripting and syntax of the Urdu language.
2. To verify the previous findings done in this area by performing practical implementations.
3. Understanding the dynamics of the language and experiment on different python platforms for compatibility.
4. Practically perform machine learning algorithms and natural language processing techniques on the Urdu language data sets.
5. Evaluate the outcomes of the research and analysis by the performance of implemented models.
6. Generate a comprehensive report of all the findings.

2. Literature Review

S.No	Paper Name	Year Published	Dataset/ Source	Preprocessing/Method	Algorithm/ Modeling/Approach	Evaluation
1	Associating targets with SentiUnits: *a step forward in sentiment analysis of Urdu text* (Martinez-Enriquez, 2012)	2012	Dataset used in this research is of electronic appliances and movie reviews.	Two parsing method is used to achieve extraction and linking. The parsing method is infused with lexicon method to find the value of orientation, either positive or negative. Text chunking using shallow parsing method is used. Beginning and end of grammatical phrases are identified without the parsing of full phrase structure.	Classification and Shallow parsing-based chunking.	Performance metrics is calculated.
2	Lexicon-based approach outperforms Supervised Machine Learning approach for Urdu Sentiment Analysis in multiple domains (Neelam Mukhtara, 2018)	2018	Dataset used in this research is from Urdu blog posts of different genres.	An analyzer is used for the processing and classification of positive and negative sentences. SVM, Decision tree and KNN with 10-fold cross validation is performed.	SVM, J48 Decision Tree and KNN are used as classification models. Sentiment analyzer and Sentiment lexicon is used.	Comparison of two approaches is made in terms of performance metrics.

3	Urdu Sentiment Corpus (v1.0): Linguistic Exploration and Visualization of Labeled Dataset for Urdu Sentiment Analysis (Muhammad Yaseen Khan, 2020)	2020	Tweet dataset	Visualization w.r.t the class distribution on the text, length of words and document, and mean absolute error. Evaluation of the dataset was done under Zipf's law. Textual similarities. POS analysis with universal PoS and language specific Pos tags. All methods are applied for visualization purposes only and not sentiment analysis.	**Textual Similarity:** Sørensen–Dice coefficient, Tversky index, Tanimoto Distance, Cosine similarity, Monge-Elkan **Evaluation of Corpus:** Zipf's law	Text similarity w.r.t the comparison on character-level and n-grams up to n=3. Term document incidence TDI for verifying the hypothesis. TF on Arabic script for visualization The pointwise mutual information for the relationship between two words.
4	Urdu Sentiment Analysis Using Supervised Machine Learning (Khan, 2017)	2017	Urdu blogs of different genres is used.	Three classifiers are used by the training data. 10-fold cross validation is performed for the evaluation of the classifiers.	Support Vector Machine, Decision Tree, K Nearest Neighbor	Performance metrics is calculated.
5	Lexicon based sentiment analysis for the Urdu language. (Bajwa, 2016)	2016	Urdu news website data set is used in this research from different news websites.	Preprocessing is performed in the first stage. After performing tokenization, tokens are passed for the polarity identification. Lexicon approach is used to identify the polarity of the sentences.	Urdu sentiment analyzer, Sentiment Lexicon	Performance metrics is calculated.
6	Sentiment Analysis on Urdu Tweets Using Markov Chains (Ghani, 2020)	2020	Urdu Tweets for sentiment analysis from GitHub	Training and Testing set is used for preprocessing. Markov chain model is used on the training set for learning. Evaluation is made using test set.	Classification	the polarity of the text. Positive, negative, and neutral Accuracy and F-score

7	Salience Analysis of NEWS Corpus using Heuristic Approach in Urdu Language (S. Abbas Ali, 2016)	2016	Urdu News corpus	Normalizing the text, Tokenization, POS to find entities, Word segmentation, Saliences founded.	Heuristic Approach	Polarity (degree of positivity or negativity), Parts of Speech, Occurrence in Heading, Total Occurrence in Corpus, Occurrence/Position in First Sentence, Occurrence/Position in Other Sentences, Ambiguity Correction

Table 1 Comparison of past papers

(Martinez-Enriquez, 2012) In this paper, a sentiment classification model that is motivated grammatically has been introduced. The focus has been SentiUnits identification instead of the words in the text. Expression of the grammatical structures in a sentence, such as opinion expression, can be defined as SentiUnits. The prosed method in this research uses shallow parsing-based methods to extract the targeted expressions and SentiUnits. An Urdu words sentiment-annotated Lexicon was developed to identify the text's polarity (+ or -) and the intensity score. The dataset used was movie reviews and electronic appliances. In their model, sentence polarity is calculated by using the polarities of its integral SentiUnits. Their model breaks up into four modules: PREPROCESSOR, EXTRACTOR, ASSOCIATOR, and CLASSIFIER. Firstly, the word boundaries and sentence segmentation of the essential words are identified by the PREPROCESSOR. Expressions are extracted from the phrases as SentiUnits by the EXTRACTOR module. A link is created between the extracted expressions and targeted sentences by the ASSOCIATOR module. Lastly, the CLASSIFIER module identifies the polarities by predicting the sentiment of the overall reviews. Identifying the beginning and end of a sentence is made without parsing using the Shallow parsing-based chunking. Finding the noun and adjective phrases in a given review sentence is done using the shallow Extractor parser. The results were evaluated using Accuracy, Precision, Recall, and F-measure/F1 score metrics.

(Neelam Mukhtara, 2018) Sentiment Analysis/Opinion mining was performed using reviews and blogs in this paper. The authors made a comparison between the lexicon-based approach and the supervised machine learning (SML) approach. SML algorithms such as Support Vector Machine, Decision Tree, and K-nearest neighbor were used to calculate the performance metrics. Sentence polarity was measured using both approaches, and detailed evaluation was done. An Urdu language sentiment analyzer and Urdu Sentiment Lexicon were used for the lexicon-based approach's experimental

purpose. In this approach, the number of sentences used was 6025 from 151 different Urdu blogs extracted for multiple sources. Three different humans did annotation, and the Kappa statistic was used for inter-rater reliability. For sentiment analysis Urdu sentiment lexicon was constructed, and the Analyzer was used for performing the analysis. For the SML approach preprocessing was performed by removing stop words, classifying sentences, and feature selection was made at ten-fold cross-validation to evaluate the three classifiers one by one. In conclusion, according to the experiments, the lexicon-based approach outperformed the SVM approach, whereas KNN performed better in accuracy than the other two algorithms.

(Muhammad Yaseen Khan, 2020) The core purpose of this research paper was to visualize the Urdu Twitter dataset. The researchers have shown the visual insights of the dataset at different levels. Textual similarities and manifold learning have been shown at document level as well as word level. Lexicon sentiment and part of speech analysis have also been included to understand the Urdu Corpus better. Preprocessing and data cleaning was performed on the Twitter dataset before tokenization. The dataset was labeled with Positive, Negative, and Neutral polarities. The corpus's evaluation was made under Empirical laws such as Zipf's law for the dataset's visual representation. Token-based similarities were concluded using the Tversky index to measure the tweets' similarities by adjusting the alpha and beta parameters. The visualization of the vectors in a higher-dimensional space can become very tricky and challenging, and for that purpose, Manifold learning algorithms come into play to overcome this problem. The authors have covered this problem in a detailed manner and shown it with a visual representation. The world-class association was covered with a statistical method's help by building a sentiment lexicon using the labeled corpus. All the different tasks being performed for analysis were w.r.t to the visualization of the dataset at each point.

(Khan, 2017) The Urdu language sentiment analysis was performed using three different classifiers in this paper, Support Vector Machine, Decision Tree, and K-Nearest Neighbor. A comparison of all the classifiers was made using the performance metrics by extracting valuable features and information. The analysis was made on the same data set as (Neelam Mukhtara, 2018). Weka is used for the classification of the labeled dataset by converting it into attribute relation format. The model's training and testing were made at ten-fold cross-validation, and the performance metrics for the analysis of the results were used. Multiple experiments were taken place by tuning the total number of features and attributes before the conclusion was made. The performance of all three classifiers used in this research did not show competitive results due to the dataset's small size. Most of the results that were analyzed performed poorly at an accuracy rate of under 50%. According to the conclusion in this research, the K-NN classifier performed better than the other two classifiers, but the results were still not up to the mark.

(Bajwa, 2016) An application using the Lexicon-based method has been created to analyze the Urdu comments on multiple news websites. The system functions to detect the text and then allocate the polarity to the specific sentence (neutral, positive, or negative). The dataset or comments that have been used in this research has been retrieved from Urdu news website such as dawn news, BBC Urdu, and different user opinions form blog posts. The functioning of the system follows a procedure in which the comments are preprocessed in the first stage. Sentence tokens are formed and later passed for the polarity identification stage. The sentiment lexicon assigns word polarity. Polarity assigning is made by 1 = Positive, -1 = Negative and 0 = Neutral. Once the polarities calculation is made, the overall sentence's polarity is identified by weighing positive or negative indications. The authors experimented on the dataset of 124 Urdu comments only, which is insufficient to do sentiment analysis. The overall accuracy retrieved was 66% only. Since the Urdu language analysis is technical and challenging, there is still much improvement required.

(Ghani, 2020) The approach used in this research was based on Markov chains used to predict Urdu tweets' sentiments. The Twitter network was used for the extraction of labeled Urdu tweets corpus. The tweet set consisted of 1400 tweets that human annotators labeled. The training set was of 980 tweets, and the rest were used for training. Preprocessing was made on the dataset, and all the non-Urdu characters, URLs, Emoticons, and Diacritics were removed. Partitioning of the dataset was made after removing stop words, and the training set was used for Markov chain modeling. Markov chain model steps were as follows: (a) A word list was formed for a single sentiment polarity label. (b) Once the labeled polarity was assigned, a probability matrix was built to transition each character for a single character. In the evaluation phase, a single instance of the test went through transition matrices learned during the labeled sentiment modeling. The maximum score of the transition probability was assigned to the test document. According to the research, the Markov chain method performed better than the Lexicon and machine learning-based method in terms of accuracy.

(S. Abbas Ali, 2016) Important entities and Salience of Urdu News corpus were experimented with to find the polarities (positive or negative) in this research. The steps are taken to determine salience involved finding important entities, speech tagging, and assignment of polarities to the news corpus. Normalization of the text was skipped as the dataset used was of news. Tokenization or breaking sentences into chunks was performed. Part of speech tagging was done to find essential entities on which the whole news sentences were dependent, and finally, Saliences (words that define polarities) were found in the final stage. A statistically based part of speech tagger that was pre-developed by the University of Engineering and Technology Lahore was used. The primary parameter for identifying the entities was Proper Noun, Most repetitive one and News Headings. For the allocation of normalized weights, 100 was divided into four parts to assign each N-Gram, in which 40 was allocated to Bi-Grams, 30 to Trigrams, 20 to Four-Grams, and 10 to five-Grams. A heuristic approach was followed in this research to find Salience in Corpus of Urdu News. The total accuracy generated by using this

approach was 84.5% that, according to the authors, was promising, and more experiments will be performed using computational-based and machine learning-based approaches.

The amount of work that has already been carried out in Urdu language sentiment analysis is not sufficient. After researching about the related work, we found out, many methods and approaches that have taken place in order to improve the accuracy by implementing different classification models and methods; such as Lexicon based, Heuristic, supervised machine learning approaches, and Markov chain models; thus, the dataset that has been used in most of the cases was not competent. We know that we will require a fair amount of dataset for carrying out adequate research and testing, which can help us gain better accuracy and results in terms of performance metrics. The work done in this field is significantly diversified as the Urdu language has not been an easy language for experimental purposes in sentiment analysis, which led to follow the same path of using machine learning algorithm on a more extensive dataset and also experiment with deep learning algorithms. This aims to perform better and generate more accurate classification models. The research and literature review carried out for this project has helped a lot in figuring out what else can be experimented with and implemented for better results and forming a methodology to be followed, explained in the next section.

3. Methodologies

The implementation of the project has been done in various stages. The steps include importing the dataset, preprocessing the Dataset (Removal of stop words, Lemmatization/Stemming), Implementing the feature extraction technique, applying classification mode, and evaluating the model design, results, and conclusion. The figure below illustrates the stages involved in the project.

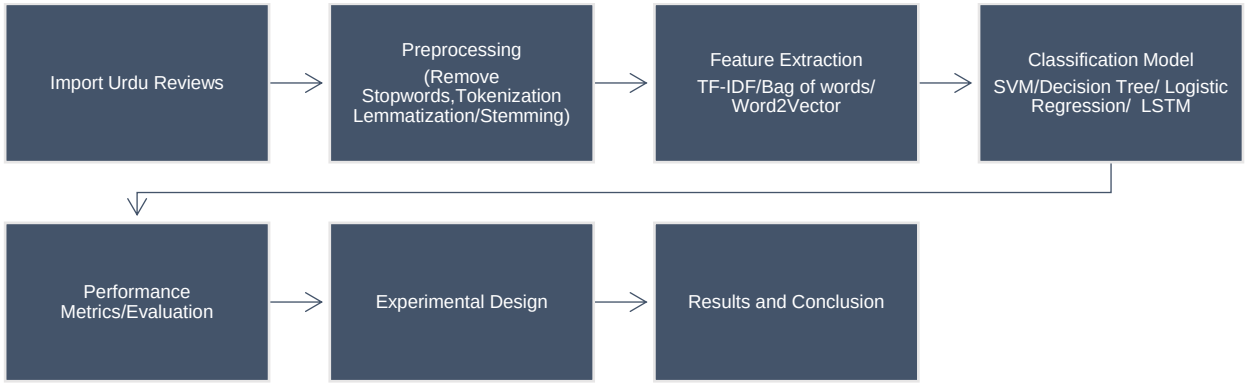


Figure 1 Sentiment Analysis System Design

This part of the section will describe the data collection and code used, Pre-processing and cleaning steps that were taken, description of the feature extraction techniques used, Models used for classification, and performance metrics used to compare the models' accuracy using different techniques.

3.1 Dataset and Code

The steps included for conducting the research are as follows:

1. The dataset used to perform sentiment analysis on the Urdu language has been collected from GitHub (akkefa, n.d.). The dataset consists of 50K Urdu movie reviews and is labeled with (positive = 1 and negative = 0) polarities equally divided with 25000 each. It was challenging to find a fair number of Datasets as work done in this area is very limited, and datasets currently available on the internet are not sufficient for carrying out the research.
2. An open-source code from Urdu NLP(Irfan, 2020), Kaggle (Verma, 2018) and Kaggle (Christmann, 2018) has been used during this project. Modification and implementation were done according to the requirement of the project. Most of the work was done in the English Language, so different experiments were performed in making the code work.
3. Testing of different feature extraction techniques and classification models is done on Python language. The reason for choosing this language was as it was a new language during the masters and the concepts were quite clear.
4. A comparison of different algorithms using different feature extraction techniques has been made. Accuracy generated by multiple algorithms using different techniques is observed.

3.2 Preprocessing

Data preprocessing is a crucial step in Machine Learning that enhances the data quality, promoting meaningful insights from the data. We can refer to preprocessing as steps taken for cleaning or organizing the raw data to convert it into a readable format and make it suitable for building a classification model over it. In other words, it can be said that it is a data mining technique to convert raw data into a predictable and analyzable form (Goyal, 2020). Data cleaning or organizing using the Urdu text can become challenging due to the language's nature. Steps such as Removal of stop words, Tokenization, and Stemming/Lemmatization described in this section are as follows:

3.2.1 Stop words (conjunction words) (ہاروف جار) Haroof Jar

Stop words are usually words known as common words in a language that completes a sentence. In Natural Language Processing, It Is required to filter out these words after or before the text's processing. In the Urdu language, Haroof Jar plays a vital role in completing a sentence. Stop words do not hold any role in the classification of sentences by the model rather than saving computation time. The idea behind removing such words is to reduce the classifier's feature space, which helps generate accurate results (Khan, 2017). (مگر, Magar, But), (آتا, Aata, Comes), (ہیں, Hain, Are), (یہ, Yeh, This) are the examples of stop words in Urdu text.

```
# Remove stop words from text
from typing import FrozenSet

# Urdu Language Stop words list
STOP_WORDS: FrozenSet[str] = frozenset("""
 آ آئی آئیں آئے آتا آتی آتے آس آمدید آنا آنسہ آنی آنے آپ آگے آہ آہا آیا اب ابھی ابے
 ارے اس اسکا اسکی اسکے اسی اسے اف افوہ البتہ الف ان اندر انکا انکی انکے انہوں انہی انہیں اوئے اور اوپر
 اوہو اہا اہا اپنا اپنوں اپنی اپنے اپنےآپ اکثر اگر اگرچہ اہاہا ایسا ایسی ایسے ایک بائیں بار بارے بالکل باوجود باہر
 بج بجے بخیر بشرطیکہ بعد بعض بغیر بلکہ بن بنا بناؤ بند بڑی بھر بھریں بھی بہت بہتر بہتیرا تاکہ تاہم تب تجھ
 تجھی تجھے ترا تری تلک تم تمام تمہارا تمہاروں تمہاری تمہارے تمہیں تو تک تھا تھی تھیں تھے تیرا تیری تیرے
 جا جاؤ جائیں جائے جاتا جاتی جاتے جاتی جاتے جب جبکہ جدھر جس جسے جن جناب جنہوں جنہیں جو جہاں جی جیسا
 جیسوں جیسی جیسے حالانکہ حالاں حصہ حضرت خاطر خالی خواہ خوب خود دائیں درمیان دریں دو دوران دوسرا دوسروں دوسری دوں
 دکھائیں دی دیئے دیا دیتا دیتی دیتے دیر دینا دینی دینے دیکھو دیں دیے دے ذریعے رکھا رکھتا رکھتی رکھتے رکھنا رکھنی
 رکھنے رکھو رکھی رکھے رہ رہا رہتا رہتی رہتے رہنا رہنی رہنے رہو رہی رہیں رہے ساتھ سامنے ساڑھے سب سبھی
 سراسر سمیت سوا سوائے سکا سکتا سکتے سہ سہی سی سے شاید شکریہ صاحب صاحبہ صرف ضرور طرح طرف طور علاوہ عین
 فقط فلاں فی قبل قطا لئے لائی لائے لاتا لاتی لاتے لانا لانی لانے لایا لو لوجی لوگوں لگ لگا لگتا
 لگتی لگی لگیں لگے لہذا لی لیا لیتا لیتی لیتے لیکن لیں لیے لے ماسوا مت مجھ مجھی مجھے محترم محترمہ محض
 مرا مرحبا مری مرے مزید مس مسز مسٹر مطابق مل مکرمی مگر مگھر مہربانی میرا میروں میری میرے میں نا نزدیک
 نما نہ نہیں نیز نیچے نے و وار واسطے واقعی والا والوں والی والے واہ وجہ ورنہ وغیرہ ولے وگرنہ وہ وہاں
 وہی وہیں ویسا ویسے وہیں پاس پایا پر پس پلیز یون یونی یونے پھر یہ پہلا پہلی پہلے پیر پیچھے چاہئے
 چاہتے چاہیئے چاہے چلا چلو چلیں چلے چناچہ چند چونکہ چکی چکیں چکے ڈالنا ڈالنی ڈالے کئے کا کاش کب کبھی
 کدھر کر کرتا کرتی کرتے کرم کرنا کرنے کرو کریں کرے کس کسی کسے کم کن کنہیں کو کوئی کون کونسا
 کونسے کچھ کہ کہا کہاں کہہ کہی کہیں کہے کی کیا کیسا کیسے کیونکر کیونکہ کیوں کیے کے گئی گئے گا گنا
 گو گویا گی گیا ہائیں ہائے ہاں ہر ہرچند ہرگز ہم ہمارا ہماری ہمارے ہمی ہمیں ہو ہوئی ہوئیں ہوئے ہوا
 ہوبہو ہوتا ہوتی ہوتیں ہوتے ہونا ہونگے ہونی ہونے ہوں ہی ہیلو ہیں ہے یا یات یعنی یک یہ یہاں یہی یہیں
""".split())


def remove_stopwords(text: str):
    return " ".join(word for word in text.split() if word not in STOP_WORDS)
```

Figure 2 Urdu Text Stop words list

The following figure consists of the Urdu text stop words list that is used in this project is taken from (Karim, 2020)

3.2.2 Lemmatization

An essential activity in data preprocessing is lemmatization. The core objective of lemmatization is word standardization which means to reduce a word into its origin or root. For the most part, lemmatization alludes to doing things appropriately with jargon and morphological examination of words, typically planning to eliminate inflectional endings and restore the base or word reference type word is known as the lemma. Lemmatization usually deals with textual data before Data Mining and Natural Language Processing. Words are reduced to its stem, base or root form while performing lemmatization, e.g., the lemma of فلمیں (Filmen, Movies) is فلم (Film, Movie) and lemma of خوفناک (Khaufnaak, Horrifying) is خوف (Kauf, Horror). Lemmatization was performed with the Urduhack library's help as most lemmatizers are for English and other European languages (Ali Daud, 2016).

```
from urduhack.models.lemmatizer import lemmatizer
def lemitizeStr(str):
    lemme_str = " "
    temp = lemmatizer.lemma_lookup(str)
    for t in temp:
        lemme_str += t[0] + " "

    return lemme_str
```

Figure 3 Code Snippet for Urduhack Lemmatizer

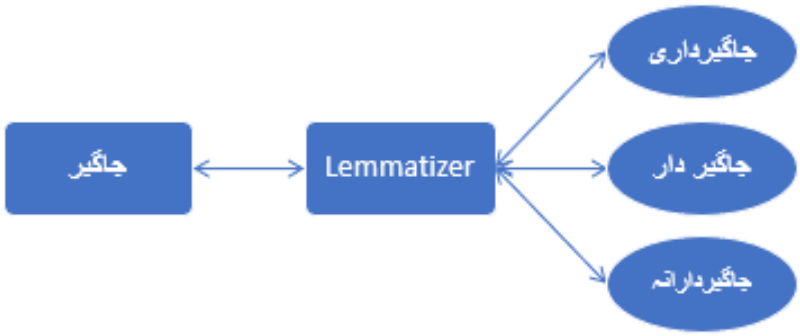


Figure 4 Lemmatization example for word جاگیر (Jagir, Land (Ali Daud, 2016))

As shown in the figure the Urdu word جاگیر (Jageer, Land) is the root word of three different words جاگیرداری (Jageerdari, Feudal state), جاگیر دار (Jageedar, Landlord) and جاگیردارانہ (Jageerdarana, Feudal) (Ali Daud, 2016).

3.2.3 Tokenization or Word Segmentation

Tokenization is one of the methods used for processing data in Natural Language Processing. Tokenization is performed by breaking down the raw text into chunks. These broken chunks of texts are called "Tokens." It helps in understanding the context of the sentence or in developing a model for the language processor. Tokenization helps in understanding the meaning of the given text by analyzing the sequence of the words. e.g. میرا نام محمد ایماز ہے (My name is Muhamamd Emaaz) into ‘ہے’ ‘ ایماز’ ‘محمد’ ‘نام’ ‘میرا’ (‘My’ ‘Name’ ‘Is’ ‘Muhammad’ ‘Emaaz’) (Chakravarthy, 2020). In Urdu text, there is no concept of spacing between the words written. A native speaker can quickly identify the start and end of a word, but it can become challenging for a machine to interpret Urdu words without segmentation of those words. There are two types of characters in the Urdu language, namely 'joiner' and 'non-joiner.' When a word ends with a joiner character, only then inter-word spacing is used, and there is no space required if a word ends with a non-joiner character. For proper tokenization of Urdu text, it is necessary to manipulate space between words (Zobia Rehman, 2011). NLTK library is used to perform tokenization on Urdu text in this project.

	review	sentiment	encoded_sentiments	text_cleaned
0	...میں نے اسے 80 کی دہائی کے وسط میں ایک کیبل گائ	1	1	...میں, نے, اسے, کی, دہائی, کے, وسط, میں, ایک, ک]
1	...چونکہ میں نے 80 کی دہائی میں اسپیکٹر گیجٹ کارٹ	0	0	...چونکہ, میں, نے, کی, دہائی, میں, اسپیکٹر, گیجٹ]
2	... ایک ایسے معاشرے کی حالت کے بارے میں تعجب کرتا	1	1	...ایک, ایسے, معاشرے, کی, حالت, کے, بارے, میں, ت]
3	...مفید البرٹ پیون کی طرف سے ایک اور ردی کی ٹوکری	0	0	...مفید, البرٹ, پیون, کی, طرف, سے, ایک, اور, ردی]
4	...یہ کولمبو ہے جس کی ہدایتکاری اپنے کیریئر کے اب	1	1	...یہ, کولمبو, ہے, جس, کی, ہدایتکاری, اپنے, کیری]

Figure 5 Tokenized form of Reviews

As shown in the figure NLTK library was used for tokenization, and after preprocessing the reviews, tokenization was performed, and the directory was saved as text_cleaned.

3.3 Feature Extraction Techniques

Feature extraction is used to create new features from the existing numbers of features used in a dataset and later discard the original features. A summarized form of features is created from the combination of original datasets. Feature Extractions used in this project are as follows:

1. TF-IDF (Term Frequency – Inverse Document Frequency)

   In a collection of documents, TF-IDF evaluates how much relevance there is of a word in a document. Two metrics are multiplied to ensure how many times a word is repeated in a document and IDF across the whole document. This technique is mainly used to analyze text and word scoring for the algorithms used in Machine Learning. The main task of TF-IDF is to search documents and retrieve information. It functions by expanding consistently by the number of times a word appears in a record and is balanced by the number of documents that contain the word; common words such as (and, where, is) are ranked low although they appear multiple times, there presence in a document does not carry that much weight. If the word "System" repeatedly appears in a set of documents and has less appearance in other documents, it can be said that the word carries some weight and is relevant.

   The calculation takes place by multiplying two metrics:

- Term Frequency: We can calculate frequency in several ways; one of them is counting instances in a word that appears in a document. The length can adjust the frequency of the document.

- Inverse Document Frequency: The IDF of a word can be calculated across a set of documents. This is done by calculating how common rare words are available in the entire document set. There are more common words if it is close to 0. We can calculate the metrics by dividing the total number of records by the total

number of records containing a word and later calculating the algorithm. (Stecanella, 2019).

The calculation for the accuracy of TF-IDF is done as follows:

$$tf\ idf\ (t, d, D) = tf\ (t, d)\ .\ idf\ (t, D)$$

Where:

$$tf\ (t, d) = log\ (1 + freq\ (t, d))$$

$$idf\ (t, D) = log\ (\ \frac{N}{count\ (d \in D : t \in d)}\ )$$

Figure 6 TF-IDF Calculation (Stecanella, 2019)

2. Bag of Words (BoW)
   The Bag-of-words model, also known as BoW, in short, is a technique used for feature extraction from the text used for modeling, such as using machine learning algorithms. It is a flexible and straightforward approach, and it can be used in various ways to extract texts from a document. It represents texts that describe how many times a word occurs in a document ignoring the order in which they appear. The BoW can be used in multiple applications, i.e., Natural language processing, Information retrieval from documents, or Document classifications. Steps that are followed in BoW are as follows:
   - Clean Text
   - Tokenization
   - Building Vocabulary
   - Generate Vocabulary or Create Document Vectors

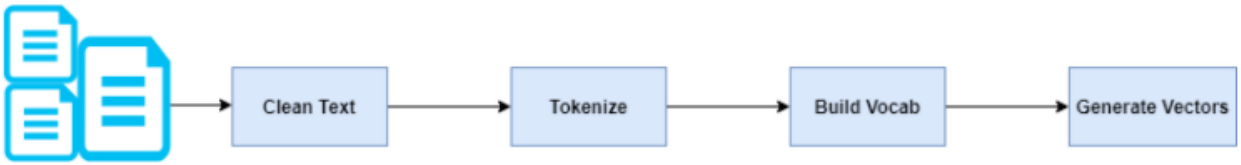


Figure 7 Steps for Bag of Words (Anon., 2018)

The reason for calling it a "bag" is as the model discards any information related to the structure or order of words in a specific document. It aims to focus on only those words that occur in a document and not wherein the document. Unlike other feature extraction models, Bow suffers from few limitations, including Vocabulary, Sparsity, and meaning (Brownlee, 2017).

3. Word to Vector (Word2Vector)
   Word to vector models is the ones that use neural networks to process a large amount of textual data. In fact, a word to vector model, instead of being a single algorithm, includes two different learning models within itself: Skip-gram and Continuous Bag of Words (CBOW). The prediction of words given in context is calculated by CBOW and oppositely skip gram function by calculating context appearing in a word. All this is processed by feeding that dataset into a single learning model, and finally, vectors are generated by the model. The process includes building vocabulary using the training text dataset and learning the vector representation of each word within that step. Word2Vector model can also calculate the cosine distance between each word, which gives us an edge of clustering the similar words related to their distance. In similar words, the projection of original feature dimensions is made to a new lower dimension (Long Ma, 2015).

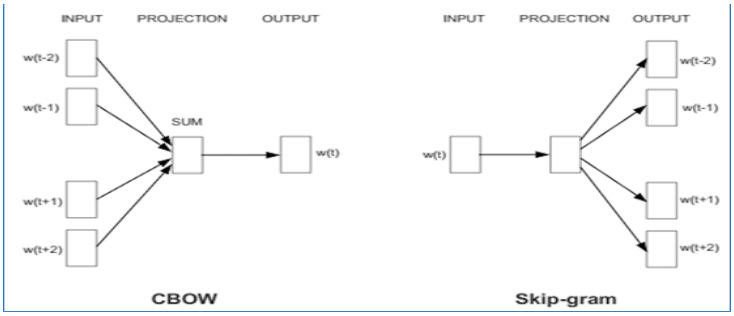


Figure 8 Working of Word2Vector (Nicholson, n.d.)

3.4 Classification Models

The classification models used in this project to demonstrate the accuracy of sentiment analysis using different feature extraction techniques are as follows:

1. Support Vector Machine (SVM)

   SVM is one of the supervised machine learning algorithms; it can be used for performing both Classification and Regression. When a labeled dataset is used for prediction, it is known as "Classification", and when a continuous value is being predicted, it is called "Regression". A hyperplane is found by performing SVM classification which is used to differentiate the classes that were plotted in n-dimensional space. Mathematical functions are used to draw a hyperplane which are called "Kernels". Following are the type of kernels in SVM:

- Linear Kernel
- Sigmoid Kernel
- RBF Kernel
- Non-linear Kernel
- Polynomial Kernel

For non-linear problems, the "Radial Basis Function" is an ideal kernel and is mostly used when there is not enough information or knowledge about the data. For linearly separable problems, the "Linear" kernel is used more often. (Reddy, 2018)

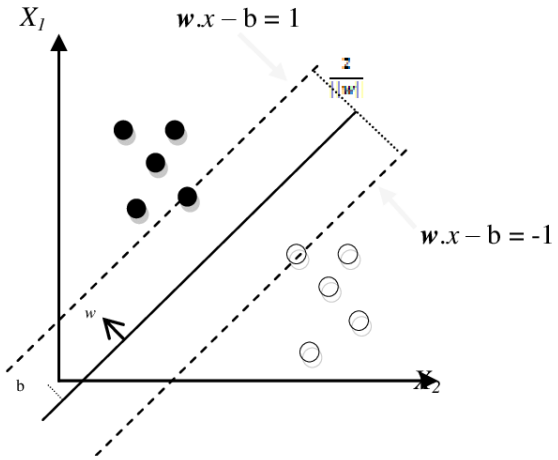


Figure 9 Optimal Separation Hyperplane between two classes (Reddy, 2018)

2. Decision Tree (DT)
   Decision Trees are a type of non-parametric supervised machine learning algorithm that can perform both Classification and Regression. The function of DT is to learn from data with a set of if-then-else decision rules. The complexity of a decision rule increases as the tree keeps getting deeper and the model keeps getting fitter. A tree-like structure is built while building a classification or regression model. Data is divided into smaller subsets, and at the same time decision tree is gradually developed. There are two nodes' decision and leaf nodes as an outcome as a result. The decision node consists of two or more branches, and the leaf node represents a decision or classification. The top node of a decision tree is known as a root node. Decision Trees can handle both numerical and categorical data (Sehra, 2018).

3. Logistic Regression (LR)
   Logistic Regression is one of the widely used algorithms for classification problems. It shares the same underlying technique as Linear Regression. Logit function is used in this classification method, and therefore it represents the name "Logistic." LR is usually used when the target variable or dependent variable is categorical (Narkhede, 2018). There are few types of LR which are as follows:
   - Binary Logistic Regression
   - Multinomial Logistic Regression
   - Ordinal Logistic Regression

   Since LR is used for the classification problem, this algorithm can be used for modeling with different feature extraction techniques in Sentiment Analysis.

4. Long Short-Term Memory (LSTM)
   Long Short-Term Memory networks are often known as "LSTMs,". These networks are an augmentation of simple Recurrent Neural Networks (RNNs). These networks are designed to tackle long-term dependency problems, and their main aim is to remember information for a more extended period. An LSTM cell contains a simple RNN cell, second state vector = Long term memory, Forget gate, Input gate, and output gate. These gates are connected with dense sigmoid layers and act as filters, and these filters decide what information needs to be forgotten, what should be the required input, and what output should be generated (Anon., 2015)

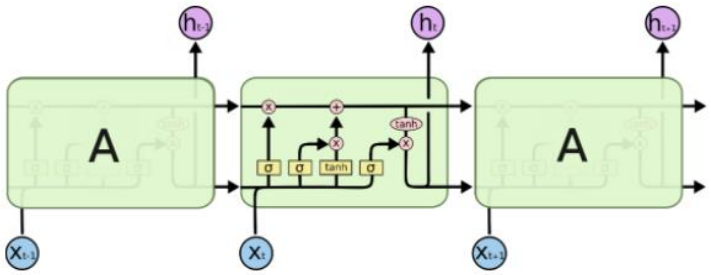
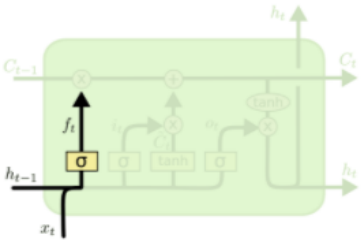


Figure 10 Working of LSTM (Anon., 2015)

A step by step working of LSTM networks is shown below:

Step 1: It is required to decide what information will not be considered in the cell state. A sigmoid layer does this decision making also called "forget gate." It investigates the input and results in output as a number between 1 and 0 for each cell state number. Number 1 means "Keep this information," and 0 means "Discard this information."



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 11 Step 1 LSTM (Anon., 2015)

Step 2: The next step is to decide what information needs to be stored in the cell state. This is done by a sigmoid layer called the "input gate layer," and values are updated accordingly. A vector is created by the tanh (tangent h) layer for the new candidate values, which should be considered in the cell state, and later combination of these two layers is done to update the state.

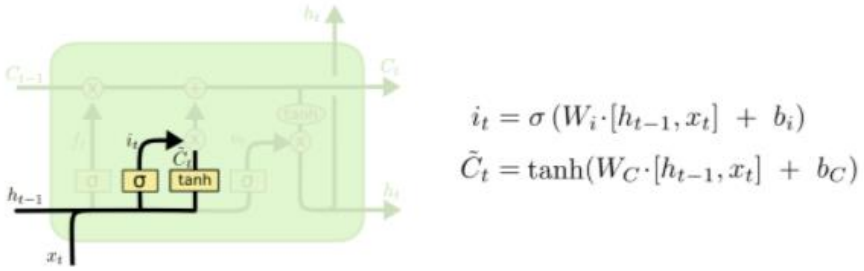


Figure 12 Step 2 LSTM (Anon., 2015)

Step 3: In this step, old cell states are updated into new cell states. In the previous step, it was decided what has to be done and, in this step, we need to implement it. The old states are multiplied by the forgotten matrix that was decided not to be considered. Then the updated state is added, which is the new value.

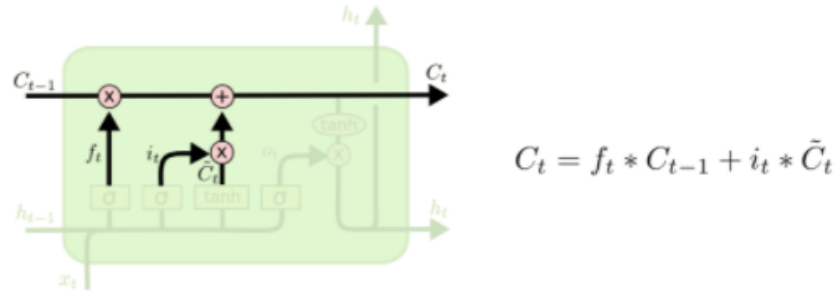


Figure 13 Step 3 LSTM (Anon., 2015)

Step 4: In the last step, what to output is decided. The output will reflect a filtered cell state. Firstly, the sigmoid layer is executed, which decides what parts of the cell state will generate output. Then the cell state goes through the tanh layer (for pushing the values between -1 and 1) and is later multiplied by the results generated by the sigmoid gate so that only the decided parts are used as an output (Anon., 2015).

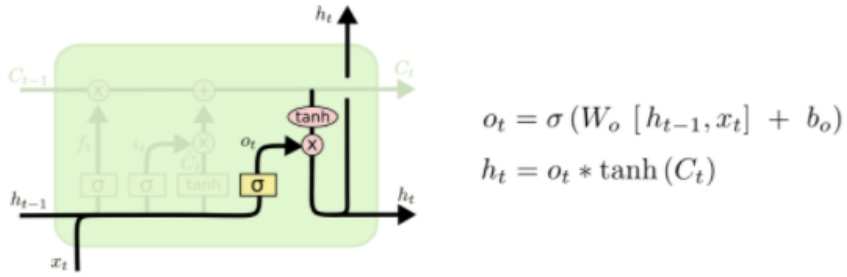


Figure 14 Step 4 LSTM (Anon., 2015)

3.5 Evaluation/Performance Measures

To workout, any technique, evaluation of the generated models must be done based on the requirements. After performing preprocessing and applying data mining techniques, the effectiveness of the process needs to be calculated. The classification model's evaluation gives us the results in the form of class or values. This process of evaluation also depends on different performance measures known as evaluation metrics. Performance measures used in classification are accuracy, recall, precision, and F1-Measure. Different types of models are generated for the evaluation of the dataset. Evaluation of the model can be performed as follows:

1. Train/Test Split:

   In this practice, the dataset is divided into testing and training datasets. Classification is performed on a trained Dataset, and testing is performed on another. This process gives us a predicted accuracy which varies if the datasets are divided into different ratios. It is a prevalent practice done during the evaluation phase and is very efficient.

2. K-fold cross-validation:
   It is a statistical method used for evaluating the skills of machine learning models. An average result is generated after applying this method which runs systematically for k-times on train/test split. Results generated by this method usually have lower bias and comparatively slower than other methods.

Once the model's evaluation is done, the next step is the evaluation of the result produced by the model. The generated results can also be called evaluation metrics. This include:

1. Confusion Matrix:
   A tabular form of the table is generated, which shows the classification model's overall performance in a confusion matrix. A 2*2 matrix is formed for a 2-class problem, i.e., in our case (Positive and Negative). It is an efficient way the evaluate a classification model. A summarized report is formed, which is classified into four categories:
   - True Positive (TP) – Positive values correctly classified.
   - True Negative (TN) – Negative values correctly classified.
   - False Positive (FP) – Positive values incorrectly classified.
   - False Negative (FN) – Negative values incorrectly classified.

		Actual	
		Positive	Negative
Predicted	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

2. Accuracy
   A proportion of accurate results generated by the model among the total number of cases inspected is called classification Accuracy.

   Accuracy (A) = (True Positive + True Negative) / (TP+FP+FN+TN)

3. Precision
   The proportion of the predicted positive class that truly belongs to the true positives.

   Precision (P) = (True Positive) / (True Positive + False Positive)

4. Recall
   The proportion of correctly classified true positives is called recall.

   Recall (R) = (True Positive) / (True Positive + False Negative)

5. F1-Measure
   The harmonic mean of recall and precision is called F1-Measure. It is a number between 0 and 1 in the classification models.

$$F_1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

6. Receiver Operating Characteristics (ROC)
   Receiver operating characteristics or a ROC curve illustrates graphical representation of binary classifier system through its diagnostic abilities. It is created by various threshold setting by plotting false positive rates (FPR) against True positive rates (TPR).

   Appendix D contains the list of all the external libraries used.
   Experimental design performed for this project is shown in the figure below.

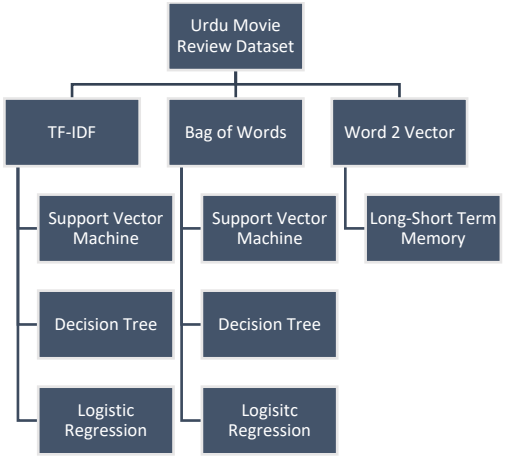


Figure 15 Experimental Design Flow (1)

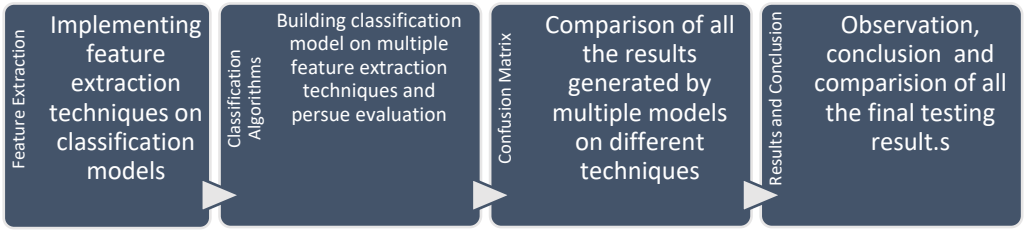


Figure 16 Experimental Design flow (2)

3.6 Legal and Ethical Issues

The study of opinion, mood, and attitude expressed through text can be called sentiment analysis. Emotions behind the words can be distinguished using machine learning algorithms and suggest whether a word is positive, negative, or neutral. This technology has raised concerns about the ethics and accuracy of this technology and calls in the question regarding customer's consent. Despite becoming a successful technology sentiment analysis still, there are many issues associated with it. Few of the legal, ethical, social, and professional issues are discussed below.

a) One of the issues is that not all words can have a specific meaning and can vary according to the domain or context of the sentence. Being aggressive is not considered a good trait but being aggressive on the football pitch can win games.
b) Suppose the sentiment analysis focuses on human emotions, expressions, or emoticons. In that case, it can generate biased results as "wink emoji" can be interpreted as a positive or negative expression, and the real meaning will be depending on the context. This type of analysis can become challenging for the machine.
c) Machine learning algorithms usually fail in differentiating between irony and sarcasm. Deep learning models have shown progress related to this problem so that less biased results can be generated and cause fewer false classification problems.
d) This technology is growing in banking, financial services, insurance, and pharma companies, which helps a business monitor the global market. These companies hold personal reviews and views of the customers and create data security concerns.
e) The findings made by sentiment analysis can mostly be 65-70% accurate, according to Susan Etlinger, research analyst from Altimeter Group. Findings made using this technology provide insights, but due to its biased results, all the findings need to be validated with other evidence.
f) Compliance is one of the other major concerns in this technology as privacy laws made by the European Union do not allow one to monitor the sentiments levels of the employees. In the US, it is legal to monitor any communications that occur on their systems.

g) Sentiment Analysis can create biased results. Due to its incapability of understanding emotion and expression accurately, it cannot be considered a groundbreaking technology that can generate accurate results.
h) A risk of misclassification can highly take place. e.g. The model falsely analyzes positive sentiments as negative and a review which is technically positive but misclassified as negative. These problems can be solved using more focused models i.e. Deep learning models such as LSTM or Convolutional neural networks.

4. Classification Algorithms and Feature Extraction

4.1 Feature Extraction

4.1.1 Term Frequency-Inverse Document Frequency (TF-IDF)

The most common feature extraction technique used in text classification is TF-IDF, but it has not been fully utilized for sentiment analysis purposes. It collects two different scores, Term Frequency (TF) and Inverse Document Frequency (IDF). (Tim O'Keefe, 2006). The Steps for implementing TF-IDF is given below:

Step 1: Upload Training and Testing Urdu dataset.

Step 2: Encoding the sentiments (positive and negative) into mathematical form (1 and 0).

Step 3: Preprocessing is performed on the dataset

a) Removing unwanted data such as Urdu and English language special characters and numbers (if any).
b) Tokenization is performed on the clean dataset.
c) All the Urdu language stop words are removed.

Step 4: Stemming and lemmatization are performed on the cleaned dataset.

Step 5: Transforming the text data into vectors by applying the TF-IDF feature extraction technique for the model to understand.

Step 6: Calculation is performed in the Urdu dataset (*TF-IDF("Dataset")* = *TF('Dataset')* IDF('Dataset')*)

Step 7: Apply the classification model to the train/test dataset.

Step 8: Evaluate the score/accuracy w.r.t performance metrics

4.1.2 Bag of Words (BoW)

A standard method used for classification and feature selection is Bag of Words or BoW. This method has its demand and carries good capabilities for classification and selection of features by creating bags for each instance type. Bag of Words model can also be described as a simplified representation used in Natural Language Processing (NLP) and Information Retrieval (IR) (Wisam A. Qader, 2019). The Steps for implementing BoW is given below:

Step 1: Upload Training and Testing Urdu dataset.

Step 2: Encoding the sentiments (positive and negative) into mathematical form (1 and 0).

Step 3: Preprocessing is performed on the dataset.

a) Removing unwanted data such as Urdu and English language special characters and numbers (if any).
b) Tokenization is performed on the clean dataset.
c) All the Urdu language stop words are removed.

Step 4: Stemming and lemmatization are performed on the cleaned dataset.

Step 6: A dictionary of word frequency is created.

Step 7: Convert all the text documents into the lower case using CountVectorizer, and conversation and transformation are performed on the "lemmatized text."

Step 8: Apply the classification model to the train/test dataset.

Step 9: Evaluate the score/accuracy w.r.t performance metrics

4.1.3 Word to Vector (Word2Vec)

Word to Vector is another method considered while working with textual data. The method converts the input taken as a word corpus and returns the output in a vector format of each word in the corpus. The vectors that are being used for the representation of words are called neural word embeddings. The word 2 vector feature extraction technique aims to convert words into numbers to train words against multiple words in the neighboring of the input corpus (Ali, 2019). The steps for implementing Word2Vector is given below:

Step 1: Upload Training and Testing Urdu dataset.

Step 2: Encoding the sentiments (positive and negative) into mathematical form (1 and 0).

Step 3: Preprocessing is performed on the dataset

a) Removing unwanted data such as Urdu and English language special characters and numbers (if any).
b) Tokenization is performed on the clean dataset using the spacy library.
c) All the Urdu language stop words are removed.

Step 4: Stemming and lemmatization are performed on the cleaned dataset.

Step 5: Creating a word 2 vector model using genism and set the hyperparameters for vocabulary size, Dimensions = 128, and max length.

Step 6: Building vocabulary or convert corpus into a one-hot encoded representation for word 2 vector model for the training dataset.

Step 6: Using Keras for tokenizing the encoded texts and creating a word 2 vector matrix.

Step 7: Applying Long-Short Term Memory (LSTM) model as a deep neural network layer for word embeddings.

Step 8: Evaluating the results generated by the deep neural network on the Validation Accuracy and Validation Loss on different learning rates or the number of epochs.

4.2 Algorithms

4.2.1 Support Vector Machine (SVM)

Support Vector Machine is used to increase the accuracy of the classification model. To produce the classification model into a higher-dimensional space, we use SVM. The Kernel used for SVM in this model is "Linear" as it is a classification problem. SVM uses parameters such as cost and gamma. For better optimization, it is essential to find the optimal values for the parameter; for that reason, we used grid search to find the best optimal parameter. It is most likely that SVM will generate better results in comparison to other classification models. The Steps for implementing SVM is given below:

Step 1: After preprocessing the Urdu review dataset, we apply the classification model.

Step 2: Dataset is split into training and testing dataset in the ratio of (70-30)

Step 3: Apply the classification model using different feature extraction techniques (BOW and TF-IDF)

Step 4: Grid search is performed on the training dataset with 10 folds cross-validation by taking parameters for the cost (0.1) and gamma (1).

Step 5: Results are generated and stored in the form of performance metrics.

Results: SVM can be considered as the most efficient classification model. They are expected to produce better results but will vary by using different extraction techniques.

4.2.2 Decision Tree

In data mining, Decision tree (DT) plays an important role for classification. Decision tree consists of high classifying speed, robust learning ability and simple construction. Like other algorithms this algorithm also has its pros and cons and they can produce unsatisfactory results in practical applications. Decision tree can produce biased results by ignoring attributes which may include less values and consider attributes with more values (Mr. Brijain, 2014). The steps for implementing DT is given below:

Step 1: After preprocessing the Urdu review dataset, we apply the classification model.

Step 2: Dataset is split into training and testing dataset in the ratio of (70-30)

Step 3: Apply the classification model using different feature extraction techniques (BOW and TF-IDF)

Step 4: Hyperparameters of decision tree are set as 'criterion': ['gini','entropy'], 'max_features': ["auto","sqrt","log2"], 'min_samples_leaf': range(1,100,1) , 'max_depth': range(1,50,1)

Step 5: Tuned parameters for testing includes 10 folds cross-validation (cv) and number of iterations were set from 1-10 for each iteration on each CV.

Step 6: Results are generated and stored in the form of performance metrics.

4.2.3 Logistic Regression

In sentiment analysis Logistic Regression (LR) can play a vital role as its most used algorithm when the output's nature is of binary number 0 or 1 and, in our case, negative or positive. It helps to differentiate the class or another. LR is a useful machine learning algorithm that utilizes the sigmoid function and works well on binary classification problems (Anon., 2019). The steps for implementing LR are given below:

Step 1: After preprocessing the Urdu review dataset, we apply the classification model.

Step 2: Dataset is split into training and testing dataset in the ratio of (70-30)

Step 3: Apply the classification model using different feature extraction techniques (BOW and TF-IDF)

Step 4: Hyperparameter for Logistic Regression are set as 'C': [0.001, 0.01, 0.1, 1, 10] , 'penalty':['l1','l2']. Where the best results were generated at penalty of 11.

Step 5: Tuned parameters for testing includes Cost value of 0.001, 0.01, 0.1, 1, 10 at 10 cross-validation for each value of cost.

Step 6: Results are generated and stored in the form of performance metrics.

4.2.4 Long-Short Term Memory (LSTM)

Traditional methods for text analysis are very limited and cannot deal with a large amount of data to retrieve critical information; therefore, we have to jump towards deep learning modes, and Recurrent Neural Network (RNN) can be a good model for text analysis. Long-Short term memory (LSTM) neural network structure is different from traditional RNN. The structure for LSTM allows the discovery of both short and long patterns in a dataset and terminates the gradient vanishing problem by training RNN. The results generated by the LSTM model are promising in the field of sentiment analysis or language modeling (Dan Li, 2016). The steps for implementing LSTM are given below:

Step 1: After preprocessing the Urdu review dataset, we apply the classification model.

Step 2: Tokenization is performed on the text using the spacy library

Step 3: Apply the classification model from genism using the word2vector feature extraction technique.

Step 4: Vector training is performed using TensorFlow for encoded padding.

Step 5: Dataset is split into training and testing dataset in the ratio of (80-20)

Step 6: TensorFlow libraries are imported for layer embeddings, Activation functions, models, optimizers, initializers, and regularizers.

Step 7: Implementation of Long-Short term sequential model with layer embeddings of (256, 128, and 64), activation function (tanh and sigmoid), and dropout value of (0.3) are set as parameters for the model.

Step 8: Hyperparameters are tuned and tested at different epochs for (1-10) at a validation split of (0.2) for each learning rate.

Step 9: Plots for accuracy and loss are generated using plt.plot.

Step 10: Results are generated and stored in the form of performance metrics (Without confusion matrix).

The LSTM model tends to generate better results, but the computational time for this model is too high, and for that reason, the testing of this model for the research conducted is limited.

5. Implementation

The description of the project implementation is shown in this chapter. The source code for the following implementation sections 5.1, 5.2, 5.3, and 5.4 can be found in Appendix A and B.

5.1 Importing Libraries and Preprocessing

Performing sentiment analysis on the Urdu text, it was essential to gather a valid dataset. The dataset for Urdu text used in the research is 50K movie reviews that are annotated by Positive = 1 and Negative = 0. Before applying any feature extraction or classification algorithm, it was necessary to implement the required libraries to accept the Urdu text and implement the required preprocessing techniques. The dataset of 50 thousand was equally divided into 50% Positive and 50% negative reviews to reduce the biases of any classifier.

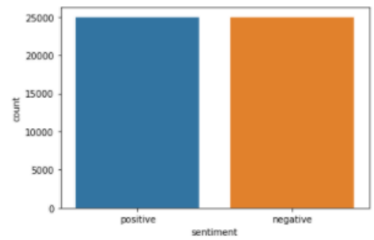
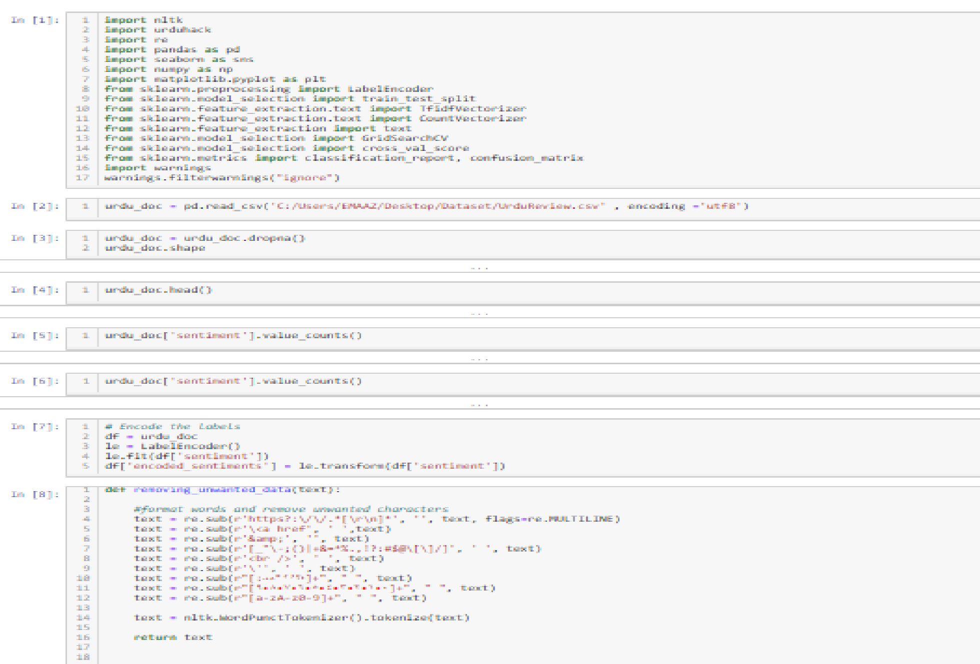


Figure 17 Fifty Thousand Positive and Negative Reviews

The two essential libraries imported to perform analysis on the Urdu text were NLTK and Urduhack. The preprocessing involved removing unwanted characters, removing stop words, lemmatization, and division of training and testing set into the ratio of 70-30.



```
In [1]:
import nltk
import urduhack
import re
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction import text
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
import warnings
warnings.filterwarnings("ignore")

In [2]:
urdu_doc = pd.read_csv('C:/Users/EMAAZ/Desktop/Dataset/UrduReview.csv" , encoding ='utf8')

In [3]:
urdu_doc = urdu_doc.dropna()
urdu_doc.shape
...

In [4]:
urdu_doc.head()
...

In [5]:
urdu_doc['sentiment'].value_counts()
...

In [6]:
urdu_doc['sentiment'].value_counts()
...

In [7]:
# Encode the Labels
df = urdu_doc
le = LabelEncoder()
le.fit(df['sentiment'])
df['encoded_sentiments'] = le.transform(df['sentiment'])

In [8]:
def removing_unwanted_data(text):

    #format words and remove unwanted characters
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
    text = re.sub(r'\<a href', ' ',text)
    text = re.sub(r'&amp;', '', text)
    text = re.sub(r'[_"\-;()|+&=*%.,!?:#$@\[\]/]', ' ', text)
    text = re.sub(r'<br />', ' ', text)
    text = re.sub(r'\'', ' ', text)
    text = re.sub(r"[illegible]", " ", text)
    text = re.sub(r"[illegible]", " ", text)
    text = re.sub(r"[a-zA-Z0-9]+", " ", text)

    text = nltk.WordPunctTokenizer().tokenize(text)

    return text
```

Figure 18 Code Snippet for Importing Libraries and Preprocessing (1)

```
In [11]:
# Remove Stop words from text
from typing import FrozenSet

# Urdu Language Stop words list
STOP_WORDS: FrozenSet[str] = frozenset("""
...
""".split())


def remove_stopwords(text: str):
    return " ".join(word for word in text.split() if word not in STOP_WORDS)
```

```
In [12]:
from urduhack.models.lemmatizer import lemmatizer
def lemitizeStr(str):
    lemma_str = " "
    temp = lemmatizer.lemma_lookup(str)
    for t in temp:
        lemma_str += t[0] + " "

    return lemma_str
```

```
In [13]:
df['review'] =  df['review'].apply(remove_stopwords)
```

```
In [14]:
df['lemmatized_text'] = df['review'].apply(lemitizeStr)
```

```
In [15]:
urdu_doc.head()
```

```
...
```

```
In [16]:
print (urdu_doc['lemmatized_text'])
```

```
...
```

```
In [17]:
print("Samples per class: {}".format(np.bincount(urdu_doc.encoded_sentiments)))

Samples per class: [25000 25000]
```

Figure 19 Code Snippet for Importing Libraries and Preprocessing (2)

5.2 Implementing Feature Extraction Techniques

a. Implementing TF-IDF

TF and IDF are two different terms in which TF stands for Term Frequency which provides us with the frequency of the word given in a document in the corpus. It is a balance of how many times a word apereas in a document by comparing it to the total number of words in the entire document. The TF increases as the number of words increases in the document. Every single document has its term frequency. IDF stands for Inverse Document Frequency, which calculates the weight of the rare words that occur across the comprehensive documents in the corpus. The IDF score depends on how rare the word is in the corpus.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Figure 20 Equation for TF

$$idf(w) = log(\frac{N}{df_t})$$

Figure 21 Equation for IDF

The implementation of TF-IDF can be done in two ways. Manually implements the extraction technique on the coding platform, which is time-consuming and requires all the formulas and equations to be implemented through codes and scripts. The other way of implementing TF-IDF is by using the sklearn class, which does all the calculation automatically and generates the desired results. The drawback of using the sklearn class is that the values might differ from the manual implementation as sklearn implements a smoother version of IDF and some other optimization.

```
In [18]:
train_review, test_review, train_sentiments, test_sentiments = train_test_split(df['lemmatized_text'], df['encoded_sentimen

In [19]:
# training: tf-idf
max_feature_num = 300
train_vectorizer = TfidfVectorizer(max_features=max_feature_num)
train_vecs = train_vectorizer.fit_transform(train_review)
test_vecs = TfidfVectorizer(max_features=max_feature_num,vocabulary=train_vectorizer.vocabulary_).fit_transform(test_review
```

Figure 22 Code Snippet for TF-IDF Model (sklearn)

b. Implementing BoW

The implementation of this feature extraction technique requires converting the text into vector/number format, which keeps a record of how often the most frequent word is being occurred in the document. The whole process can be summed up to extract features from the text document and later use these features for training machine learning models or algorithms. The practical implementation of the Bag of Words requires importing the required libraries, preprocessing the text/document, implement the BoW model, and training and testing the extraction model on the classifier.

```
In [18]:
BOW_convert = CountVectorizer(tokenizer=lambda doc: doc, lowercase=False)
x = BOW_convert.fit_transform(df['lemmatized_text'])

words = BOW_convert.get_feature_names()
print("The total number of columns are =",len(words))
print('Shape of arrays =',x.toarray().shape)

The total number of columns are = 303
Shape of arrays = (50000, 303)

In [19]:
import sklearn.model_selection

train,test= sklearn.model_selection.train_test_split(df, train_size = 0.7, random_state=4) #Split and Shuffle

print('Training data set shape =',train.shape)
print('Testing data set shape =', test.shape)

Training data set shape = (35000, 5)
Testing data set shape = (15000, 5)

In [20]:
BOW_convert = CountVectorizer(tokenizer=lambda doc: doc, lowercase=False)

X_train = BOW_convert.fit_transform(train['lemmatized_text'])

X_test = BOW_convert.transform(test['lemmatized_text'])

Y_train=train['encoded_sentiments']
Y_test=test['encoded_sentiments']
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)

(35000, 282)
(35000,)
(15000, 282)
(15000,)
```

Figure 23 Code Snippet for BoW Model

c. Implementing word2vec

Machines/Computers can only understand the language of numbers. The encoding of textual data into numbers can result in a unique outcome in many ways. The three techniques used for this task are Bag of words, TF-IDF, and Word2Vec (Word to Vector). Out of all these three techniques, word2vec's performances are incredibly well in Natural Language Processing tasks. The implementation of word2vec requires data preparation, creation of word2vector model, generation of variables, generation of training data, and tuning of parameters for the desired results. The preprocessing scripting requires converting the text into lower case and removing all the special characters, digits, and unnecessary spaces from the text. Tokenization occurs using the spacy library, which helps convert articles into sentences and remove all the stop words. We use the genism model, which Is a straightforward method to create word2vec models. The list of Urdu words is passed through the Word2Vec model class of the genism.models' package. Parameters for the vocabulary and dimension size are set accordingly.

```
In [19]:
import spacy
def tokenizer(str):
    nlp = spacy.blank('ur')
    doc = nlp.tokenizer(str)
    return [i.text for i in doc]
df["tokens"] = df["lemmatized_text"].apply(tokenizer)

In [20]:
import gensim

model_word2vec = gensim.models.Word2Vec(sentences=df["tokens"], size=128, window=5, workers=10, min_count = 1)

In [21]:
model_word2vec.wv.most_similar("مرد")

Out[21]:
[('0.7806915640830994', 'مردوں'),
 ('0.7236655354499817', 'خواتین'),
 ('0.7122538089752197', 'عورتوں'),
 ('0.6766682267189026', 'عورتیں'),
 ('0.6337434649467468', 'مردانہ'),
 ('0.6280367374420166', 'سیب'),
 ('0.6168665885925293', 'بوڑھی'),
 ('0.5978045463562012', 'گدورے'),
 ('0.5814326405525208', 'ننگے'),
 ('0.5664310455322266', 'ٹیپ')]

In [22]:
VOCAB_SIZE = len(model_word2vec.wv.vocab)
DIMENSIONS = 128
MAX_LEN = max([len(x) for x in df["tokens"]])

In [23]:
VOCAB_SIZE, DIMENSIONS, MAX_LEN

Out[23]:
(120815, 128, 1769)

In [24]:
from keras.preprocessing.text import Tokenizer
token = Tokenizer()
token.fit_on_texts(df["tokens"])
encoded = token.texts_to_sequences(df["tokens"])

In [25]:
words2vec_matrix = np.zeros((VOCAB_SIZE+1,DIMENSIONS))
for word, index in token.word_index.items():
    try:
        words2vec_matrix[index] = model_word2vec.wv[word]
    except:
        print(index, word)
```

Figure 24 Code Snippet for Word2Vec Model

5.3 Implementing Algorithms

a. Implementing Decision Tree

The classification model is selected once the process of implementing the feature extraction technique is completed. In this project, the Decision Tree algorithm is used as a classification model for both the TF-IDF and Bag of Words technique. The idea of using a Decision Tree as a classification algorithm was due to the less computational time of the classifier and the fact that this classifier results in the output in the form of negative and positive or True and false value, which was essential for our project as in sentiment analysis we work on the annotated data of 0 and 1 for the representation of negative or positive reviews.

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False,
random_state=None, splitter='best')
```

The classifier was implemented using the sklearn.tree package and later the model was fitted over the training data. Prediction of probabilities takes place once the model is trained.

```
y_prob = model_tree.predict_proba(test_vecs)[:,1] # This will give you positive class
prediction probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to give class
predictions.
model_tree.score(test_vecs, y_pred)
```

The tuned parameter for the Decision tree classifier included criterion, max features, min sample, and max depth. All these parameters were kept the same for both the feature extraction techniques. The positive class prediction probabilities and model tree score of the testing dataset is recorded in the form of classification report.

```
In [26]:
from sklearn.tree import DecisionTreeClassifier

model_DD = DecisionTreeClassifier()


tuned_parameters= {'criterion': ['gini','entropy'], 'max_features': ["auto","sqrt","log2"],
                   'min_samples_leaf': range(1,100,1) , 'max_depth': range(1,50,1)
                  }

In [27]:
DD_model= RandomizedSearchCV(model_DD, tuned_parameters,cv=10,scoring='accuracy',n_iter=1,n_jobs= -1,random_state=5)

In [28]:
DD_model.fit(train_vecs, train_sentiments)

...

In [29]:
print(DD_model.best_score_)

0.6870285714285714

In [30]:
print(DD_model.best_params_)

...

In [31]:
y_prob = DD_model.predict_proba(test_vecs)[:,1] # This will give you positive class prediction probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to give class predictions.
DD_model.score(test_vecs, y_pred)

...

In [32]:
confusion_matrix=metrics.confusion_matrix(test_sentiments,y_pred)
confusion_matrix

...

In [33]:
auc_roc=metrics.classification_report(test_sentiments,y_pred)
auc_roc

...

In [34]:
auc_roc=metrics.roc_auc_score(test_sentiments,y_pred)
auc_roc

...

In [35]:
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(test_sentiments, y_prob)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc

...

In [36]:
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate,true_positive_rate, color='red',label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

Figure 25 Code Snippet for Decision Tree Model

b. Implementing Logistic Regression

Logistic Regression (LR) is another algorithm used to distinguish between two classes or two categories. Logistic Regression can accept limited numbers of dependent variables only, and, i.e., it is considered categorical. In the project, the outcome is negative and positive, so it can be called Binary Logistic Regression. Sklearn library is used to implement the LR algorithm; once the model is built, and parameters are tuned; the results are recorded in the classification matrix.

The cost and hypothesis function for Logistic Regression can be described as

$$h(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$h(x) = \begin{cases} > 0.5, & \text{if } \theta^T x > 0 \\ < 0.5, & \text{if } \theta^T x < 0 \end{cases}$$

Figure 26 Logistic Regression Hypothesis Equation

$$cost = \begin{cases} -log(h(x), & \text{if } y = 1 \\ -log(1 - h(x)), & \text{if } y = 0 \end{cases}$$

Figure 27 Logistic Regression Cost Equation

In [36]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn import metrics

LR_model= LogisticRegression()

tuned_parameters = {'C': [0.01] ,
                    'penalty':['l1']
                        }
```

In [37]:
```
from sklearn.model_selection import GridSearchCV

LR= GridSearchCV(LR_model, tuned_parameters,cv=10)
```

In [38]:
```
LR.fit(train_vecs,train_sentiments)
```

...

In [39]:
```
print(LR.best_params_)
```

...

In [40]:
```
print(LR.best_score_)
```

...

In [41]:
```
LR.score(test_vecs, y_pred)
```

...

In [42]:
```
confusion_matrix=metrics.confusion_matrix(test_sentiments,y_pred)
confusion_matrix
```

...

In [43]:
```
auc_roc=metrics.classification_report(test_sentiments,y_pred)
auc_roc
```

...

In [44]:
```
auc_roc=metrics.roc_auc_score(test_sentiments,y_pred)
auc_roc
```

...

In [45]:
```
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(test_sentiments, y_prob)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
```

...

In [46]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate,true_positive_rate, color='red',label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

...

Figure 28 Code Snippet for Logistic Regression

c. Implementing Support Vector Machine

The implementation of this algorithm was done in Python coding in various steps. For the implementation of the model, packages from the sklearn model were used. Firstly, the grid search method was applied to the training dataset using the function grid search from sklearn.

```
tuned_parameters = {
'C': [1.0], 'kernel': ['linear'],
'C': [1.0], 'gamma': [0.1], 'kernel': ['linear'],
}
from sklearn.model_selection import RandomizedSearchCV
model_svm = RandomizedSearchCV(svm_model,
tuned_parameters,cv=10,scoring='accuracy',n_iter=1)
```

Executing the following code will give us an optimal value of cost and gamma. The training of the SVM model is done using these parameters from the sklearn package.

```
from sklearn.svm import SVC
svm_model= SVC()
```

The SVM model parameters include the value of cost, gamma, and the kernel function (Linear for this Project). The fit() method will be performing the training, and this training requires the processed training data by the vectorizer and the correct class labels.

To Find the best optimal values for cost and gamma, a grid search was performed using multiple gamma values [0.001, 0.01, 0.1, 1, 10] and cost [0.1 – 1]. For further experiments, only the best parameters were considered.

After all, the required preprocessing, parameter tuning, and model classification, the results, and quality of the classifier are generated using the classification report and the quality (Accuracy, Precision, Recall, and F1-score) of the model on the desired param.

SVM MODEL

In [68]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [69]:
```
from sklearn.svm import SVC
svm_model= SVC()
```

In [70]:
```
tuned_parameters = {
 'C': [1.0], 'kernel': ['linear'],
 'C': [1.0], 'gamma': [0.1], 'kernel': ['linear'],
    }
```

In [71]:
```
from sklearn.model_selection import RandomizedSearchCV

model_svm = RandomizedSearchCV(svm_model, tuned_parameters,cv=50,scoring='accuracy',n_iter=1)
```

In [72]:
```
model_svm.fit(train_vecs, train_sentiments)
print(model_svm.best_score_)
```

...

In [73]:
```
print(model_svm.best_params_)
```

```
{'kernel': 'linear', 'gamma': 0.1, 'C': 1.0}
```

In [74]:
```
y_pred= model_svm.predict(test_vecs)
print(metrics.accuracy_score(y_pred,test_sentiments))
```

...

In [75]:
```
confusion_matrix=metrics.confusion_matrix(test_sentiments,y_pred)
confusion_matrix
```

...

In [76]:
```
auc_roc=metrics.classification_report(test_sentiments,y_pred)
auc_roc
```

...

In [77]:
```
auc_roc=metrics.roc_auc_score(test_sentiments,y_pred)
auc_roc
```

...

In [78]:
```
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(test_sentiments, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
```

...

In [79]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate,true_positive_rate, color='red',label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

Figure 29 Code Snippet for SVM model

d. Implementing Long-Short Term Memory

Implementing a deep learning algorithm was to experiment a large size of Urdu textual data for Sentiment Analysis. It is a type of RNN network that is mainly used to grasp long-term dependence. The idea was to build a deep learning model that could classify the sentiments for the Urdu Movie review dataset. Once the required libraries and dataset is imported, the preprocessing of the reviews takes place. The LSTM model architecture is defined in the next step with the help of tensor flow and Kera's two most essential libraries for the interface of artificial neural networks. Sentiments/Textual data is encoded in the next step using a label encoder, and hyperparameters are set according to the need of the project.

The number of dimensions used in the project is 128. It is common to take the size of the dimension as a power of 32 (64, 128, 256). It will probably increase the cache utilization during the movement of the data and reduce the bottlenecks. The distances of words embedding space will not be affected by choosing the size of the dimensions though it can increase the amount of computational time with quite a margin.

In [32]:
```
lstm = Models.Sequential()

lstm.add(Layers.Embedding(VOCAB_SIZE+1,DIMENSIONS,
                          embeddings_initializer = Init.Constant(words2vec_matrix),
                          input_length=MAX_LEN, trainable=False ))

lstm.add(Layers.Bidirectional(Layers.LSTM(256, activation='tanh')))

lstm.add(Layers.Dense(128, activation='tanh'))
lstm.add(Layers.Dropout(0.3))

lstm.add(Layers.Dense(64, activation='tanh'))
lstm.add(Layers.Dropout(0.3))

lstm.add(Layers.Dense(1, activation='sigmoid'))

lstm.summary()
```

In [33]:
```
from keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import EarlyStopping


lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
es_callback = EarlyStopping(monitor='val_loss', patience=3)
LSTM_NET = lstm.fit(train_sentences, train_tags, epochs=10, validation_split=0.2, callbacks=[es_callback], shuffle=False)
```

In [34]:
```
plt.plot(LSTM_NET.history['accuracy'])
plt.plot(LSTM_NET.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

In [35]:
```
plt.plot(LSTM_NET.history['loss'])
plt.plot(LSTM_NET.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

In [36]:
```
print(classification_report(lstm.predict(test_sentences).round(), test_tags))
```

Figure 30 Code Snippet for LSTM model

5.4 Evaluation of Implemented Algorithms

a) Building Classification Models

To examine the overall accuracy and efficiency of the desired algorithms and feature extraction techniques, the dataset was divided into testing and training set in 2:1. The feature extraction technique was implemented on the training dataset, and the classification models were implemented on the training dataset. The classifiers used in this Project were SVM, Decision Tree, Logistic Regression, and one deep learning model, Long-Short term memory.

b) Evaluating Models/Classification Report

Models used for training on the training dataset are then tested on the testing dataset to complete the evaluation process, and thus evaluation is completed. The classification report consists of Accuracy, Precision, Recall, F1-Score, confusion matrix, and ROC curve for graphical representation.

```
In [21]:
confusion_matrix=metrics.confusion_matrix(test_sentiments,y_pred)
confusion_matrix

Out[21]: array([[6013, 1507],
                [1315, 6165]], dtype=int64)

In [22]:
auc_roc=metrics.classification_report(test_sentiments,y_pred)
auc_roc

Out[22]: '              precision    recall  f1-score   support\n\n           0       0.82      0.80      0.81      7520\n           1
0.80      0.82      0.81      7480\n\n    accuracy                           0.81     15000\n   macro avg       0.81      0.81
0.81     15000\nweighted avg       0.81      0.81      0.81     15000\n'
```

Figure 31 Classification Report

```
In [42]:
confusion_matrix=metrics.confusion_matrix(test_sentiments,y_pred)
confusion_matrix

In [43]:
auc_roc=metrics.classification_report(test_sentiments,y_pred)
auc_roc

In [44]:
auc_roc=metrics.roc_auc_score(test_sentiments,y_pred)
auc_roc

In [45]:
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(test_sentiments, y_prob)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc

In [46]:
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate,true_positive_rate, color='red',label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

Figure 31 Code Snippet for Classification Report / Evaluation

5.5 Problems during Implementation

The simplest of projects cannot be completed without facing any hurdles or barriers. The project was also involved in facing problems during the implementation part at different phases. These obstacles were solved with the help of different research work and online materials and by looking deeply into the topic of interest. The working of all the algorithms was understood individually and implemented accordingly. The supervisor's weekly

meetings and the discussion helped understand the rights and wrongs during the implementation and experimental phase. The following problems were faced during the implementation of the project.

1. Finding the appropriate Python libraries that could understand the Urdu language text data was one of the complex tasks throughout the project. The libraries used in the project do not function on the latest Python version. Python 3.7 was used to make these libraries run.

2. Python language was used to implement all the algorithms, and it was chosen based on the knowledge acquired throughout the master's program; still, there was a lot to learn and understand. Since the project involves different data science techniques, which had to be understood before starting the project, all the required knowledge was taken from different online tutorials and research papers.

3. The project includes a deep learning model used for sentiment analysis, which was a difficult task to achieve since it involved Neural Networks. Figuring out how to make the model work by using Urdu text was a hurdle. The supervisor's assistance and guidance helped overcome the problems that were raised during the projects. The implementation of LSTM is a time-consuming process as each iteration took 18 hours to generate results, due to which the testing of the algorithm done was limited.

4. Support Vector Machine was a tricky algorithm to deal with as it involved finding the appropriate optimal values of cost and gamma. Working on SVM and understanding the logic behind the algorithm was a bit time-consuming but was resolved by reading out from different sources. The kernel function used for SVM was "Linear," so it was essential to understand the kernel functions and it is mathematics.

5. One problem during the implementation phase was SVM's grid search, as finding the optimal values by experimenting with different cost and gamma values to find the most suitable parameters was a time-consuming task. The execution of grid search took place at 10 folds cross-validation and involving five values for cost and five values for gamma to find the optimal values and execution; each iteration took 45-50 mins which was a long and slow-moving process just like LSTM.

6. During implementation the project faced multiple machine crashes due the excessive amount of computational time and continuous running of the PC. Due to having low amount of machine power this hurdle was tackled accordingly without further problems.

6. Experimental Design and Results

The project's experimental design was divided into three experiments. The aim of each experiment was different from the other, and the way of conducting each experiment also varied. The dataset used in these experiments is the same, but different parameters were set. This section contains a detailed description of the experiments performed.

6.1 Experiment 1 – Evaluating Feature Extraction Techniques on different Classification Models

This experiment is performed with a specific aim to find the most suitable classification algorithm for the feature extraction technique. This experiment will give us an idea of a comparative analysis between three feature extraction techniques against three classification models. This experiment compares the algorithms with and without tuning the hyperparameters and recording the results accordingly.

The evaluation of the implemented algorithms was done using three classification models: Decision Tree, Logistic Regression, and Support Vector Machine. The sentiment analysis was based on accuracy, precision, recall, f1-Score, roc curve, and confusion matrix. The classification models were imported in python code and other language processing libraries. External libraries were used for Urdu Language sentiment analysis.

A dataset of 50 thousand Urdu movie reviews was taken, which went through the required preprocessing, and all the required work was completed before applying any feature extraction technique. The dataset was split into 70-30% (Training and Testing set). The max number of features was set as 300 (300 features are common practice). The three algorithms and two feature extraction techniques (TF-IDF and BoW) were implemented, and different classifiers are made on the dataset. Once the classification model was built, the model was then tested, and the results generated were compared and observed.

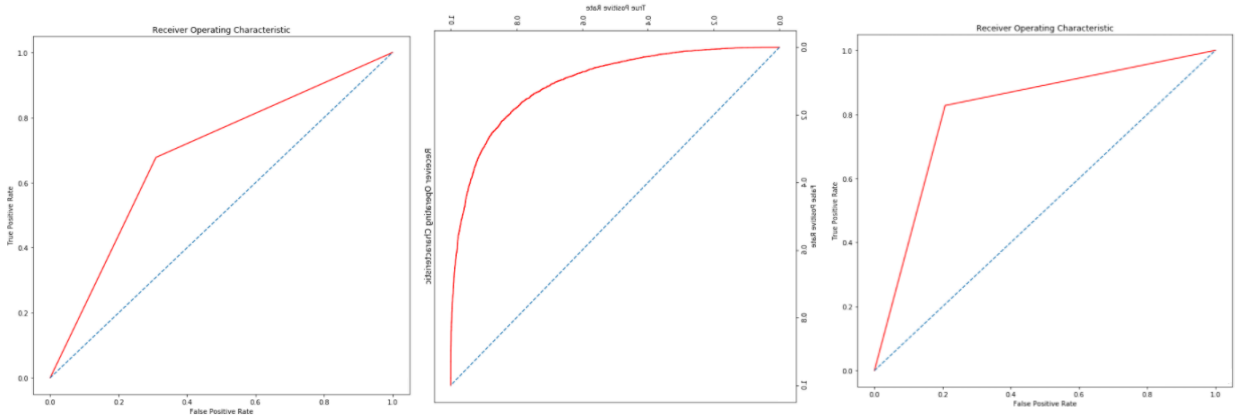
The following tables show the results generated by the models built using Decision Tree, Logistic Regression, and Support Vector Machine.

Table 2,3,4 and 5 below show the comparison of the results generated by the 3 classification models using two feature extraction techniques based on accuracy, precision, recall, f1-score, and confusion matrix. The tables consist of results generated without and with tuning the hyperparameters of the classification models.

1. Evaluating Using Term Inverse Frequency Inverse Document (Without Tuning)

Classification	Decision Tree	Logistic Regression	Support Vector Machine
Accuracy	0.684315621799977	0.81412783379792	0.798364507
Precision	0.68	0.81	0.81
Recall	0.68	0.81	0.81
F1-Score	0.68	0.81	0.81
Confusion Matrix	[5196, 2324] [2411, 5069]	[6013, 1507] [1315, 6165]	[[5773 1697] [1326 6204]

Table 2 Evaluation of classification models using TF-IDF (Without Tuning)



(ROC Curve of the all three classifiers according to Accuracy and table numbering)

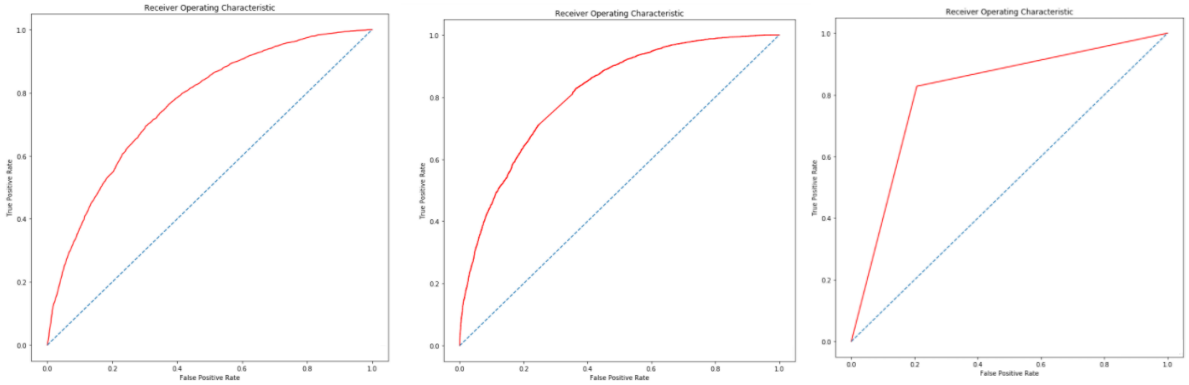
Table 2 below communicates the comparison between the classification models using the TF-IDF feature extraction without tuning the algorithms. On applying this technique, Logistic Regression and SVM works almost the same in terms of accuracy, and both the models are working better than the Decision Tree. Both highly efficient algorithms work better in terms of Precision, Recall, and F1 Score. Support Vector Machine is implemented with gird search for tuning later.

The receiver operating characteristic curve for the algorithms is shown according to the accuracy generated by all the untuned algorithms. Logistic Regression and Support Vector Machine can be graphically seen generating higher accuracy. The reason why LR and SVM shows better ROC curves is due the threshold classification as both the algorithms have higher classification threshold means more negative item are classified correctly.

2. Evaluating Using Term Inverse Frequency Inverse Document (With Tuned Hyperparameters)

Classification	Decision Tree	Logistic Regression	Support Vector Machine
Accuracy	0.68253	0.766662	0.809296148
Precision	0.68	0.76	0.81
Recall	0.68	0.76	0.81
F1-Score	0.68	0.76	0.81
Confusion Matrix	[5304, 2216] [2241, 5239]	[6059, 1590] [1350, 6195]	[5976 1494] [1366 6164]

Table 3 Evaluation of classification models using TF-IDF (With Tuning)



(ROC Curve of the all three classifiers according to Accuracy and table numbering)

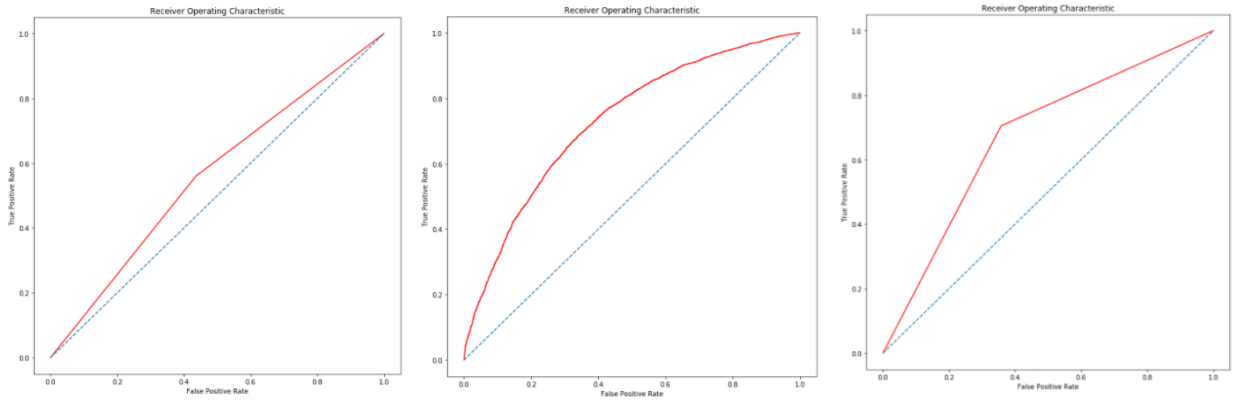
Table 3 below demonstrates the tuned algorithms' comparison, which surprisingly does not generate better results than table 2. SVM performed better after it was implemented with grid search, it has better values of the evaluation matrix; Logistic Regression works better than Decision Tree in terms of Accuracy, Precision, Recall, and F1 score in this case as well. Since SVM has a higher Accuracy rate, the further experiment conducted will be dedicated to the specific algorithm for another experiment.

The receiver operating characteristic curve for the algorithms is shown according to the accuracy generated by all the tuned algorithms. SVM remains almost the same for the tuned and untuned experiment. Decision Tree's results were unchanged, but Logistic Regression did not perform well compared to table 2, which was surprising.

3. Evaluating Using Bag of Words Technique (Without Tuning)

Classification	Decision Tree	Logistic Regression	Support Vector Machine
Accuracy	0.5620027733748547	0.6667142857142857	0.610381
Precision	0.56	0.67	0.62
Recall	0.56	0.67	0.61
F1-Score	0.56	0.67	0.60
Confusion Matrix	[4223, 3266] [3304, 4207]	[4903, 2586] [2330, 5181]	[8320, 9250] [4492, 12938]

Table 4 Evaluation of classification models using BoW (Without Tuning)



(ROC Curve of the all three classifiers according to Accuracy and table numbering)

Table 4 below show the results generated by all the three classification models using the Bag of Words technique. The overall performance of this technique did not generate good results, and it can be considered as the BoW technique did not perform well using the Urdu text due to its nature of extraction. Comparatively, SVM and Logistic Regression generated neck to neck results with a slight difference in the accuracy, and the rest of the performance measures are shared with the same values. Decision Tree's performance is even worse than the technique used in Table 2 and Table 3 in terms of all measures.

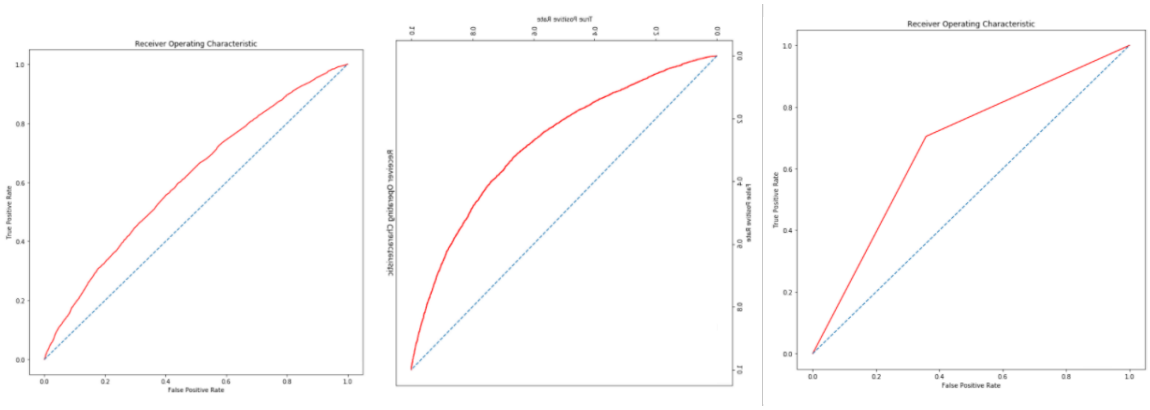
The graphical representation of this technique can show the below-average results immediately in terms of accuracy for all the classifiers. The lower the classification threshold is the more items are classified as positive; hence it increases both true

positives and false positives. It can be considered as not so beneficial for the Urdu text analysis.

4. Evaluating Using Bag of Words Technique (With Tuned Hyperparameters)

Classification	Decision Tree	Logistic Regression	Support Vector Machine
Accuracy	0.57405	0.66792	0.670381
Precision	0.56	0.67	0.67
Recall	0.56	0.67	0.67
F1-Score	0.56	0.67	0.67
Confusion Matrix	[4330, 3120] [3245, 4335]	[4950 2525] [2229, 5534]	[11016, 6554] [5069 , 1236]

Table 5 Evaluation of classification models using BoW (With Tuning)



(ROC Curve of the all three classifiers according to Accuracy and table numbering)

Table 5 results are generated using the Bag of Words technique by tuning the parameters of Decision Tree and Logistic Regression and Support Vector Machine performs better after applying grid search and increaser the accuracy up by 7%. Tables 4 and 5 have a noticeable difference of points in terms of the performance measures. Decision Tree has an increase of 1% accuracy by tuning the algorithm, and Logistic Regression decreases 1%. The results generated by BoW techniques lack classification and did not work well using the Urdu texts for sentiment analysis.

The difference between Table 2,3,4 and 5 can be seen through the ROC curves, and the first glance can show which algorithm on which feature extraction technique performed well in terms of the performance measures.

After concluding this experiment, the findings show that Logistic Regression and SVM both algorithms are working much better than Decision Tree. Support Vector Machine's performance during this test has been competitive and better than the other algorithms. Using SVM to build a classifier is comparatively time-consuming yet it gives better results in terms of the performance measures.

After concluding the findings, SVM's performance is the best in our case. It can be used for further experiments using different feature extraction techniques and tuning parameters to generate even better results.

6.2 Experiment 2 – Experimenting SVM based feature extraction with different parameters of cost on TD-IDF and BoW

This experiment was conducted to locate the optimal parameter of cost for SVM model training to generate results for sentiment analysis on Urdu text.

Cost	TF-IDF(Gamma = 0.1, Kernel = Linear)	Bag of Words(Gamma = 0.1, Kernel = Linear)
0.1	Acc: 0.810266 [5962, 1558] [1288, 6192]	Acc: 0.673466 [4810, 2679] [2219, 5292]
0.2	Acc: 0.811866 [5999, 1521] [1301, 6179]	Acc: 0.672531 [4820, 2643] [2216, 5252]
0.3	Acc: 0.810933 [5994, 1526] [1310, 6170]	Acc: 0.664621 [4899, 2626] [2249, 5296]
0.4	Acc: 0.811666 [5995, 1525] [1300, 6180]	Acc: 0.672735 [4910, 2689] [2232, 5242]
0.5	Acc: 0.811333 [5989, 1531]	Acc: 0.671133 [4886, 2630]

	[1299, 6181]	[2209, 5282]
0.6	Acc: 0.811066 [5990, 1530] [1304, 6176]	Acc: 0.671073 [4811, 2682] [2217, 5292]
0.7	Acc: 0.810733 [5984, 1536] [1303, 6177]	Acc: 0.67243 [4815, 2684] [2221 5291]
0.8	Acc: 0.807085 [5986, 1534] [1305, 6175]	Acc: 0.67108 [4812, 2675] [2259 5280]
0.9	Acc: 0.810533 [5987, 1533] [1309, 6171]	Acc: 0.672531 [4817, 2677] [2212, 5291]
1	Acc: 0.8108 [5986, 1534] [1304, 6176]	Acc: 0.662213 [4808, 2681] [2255, 5293]

Table 6 Cost value optimization for SVM based Feature Extraction

For this experiment, the classification model SVM was used for training and testing the dataset. The size of the dataset used was 50 thousand Urdu movie reviews that were split into 70-30% training and testing set. This experiment is performed to find the optimal values of the cost that can be used for training the model after pre-preprocessing the dataset.

Table 6 above contains the experimentation results for the different cost values, and a fixed gamma value of 0.1 and the SVM kernel used was linear. The cost value varies from 0.1 – 1 with an interval of 0.1. The confusion matrix and overall accuracy demonstrate the results generated by the experiment.

The classification model was implemented on two feature extraction techniques TD-IDF and BoW, with a fixed set of parameters and varying cost values. A grid search was performed on the model to find the suitable optimal values of cost and gamma.

Due to the time limit and less computation power, the experiment was limited to varying the cost values between 0.1 and 1 with a fixed gamma value. However, a difference can be seen in the results achieved by this experiment.

A trend of decline can be noticed as the value of cost increases, and misclassification of the dataset is getting inferior in both cases. The results generated by using the TF-IDF technique are better than the BoW with quite a margin and can be observed in accuracy. The optimal value of cost set as 0.1 and 0.2 can produce better results, as the overall accuracy is better with this set of parameters.
Based on the findings from this experiment, it can be said that TD-IDF on SVM outperforms the Bag of Words technique, as the margin of accuracy between the two-feature extraction technique is extensive. The SVM model is considered to generate the most accurate results, and TF-IDF on SVM can be considered a good classification model in this experiment. The SVM model on TF-IDF gave better accuracy, precision, and recall than Bag of words. For the increase in performance, functions like Grid Search can obtain an optimal value and later implement the extraction techniques.
For future experiments with the SVM model, a fixed parameter of 0.1 or 0.2 can be used with gamma 0.1 without modifying and other parameters to obtain good results. It can be said that SVM with the Bag of words feature extraction technique does not perform well, and for future experiments, techniques like the word to vector can also take place and experimented.

6.3 Experiment 3 – Experimenting Deep Learning Model with Word2Vec Technique

This experiment aims to apply one of the popular word embeddings from text methods Word2Vec (Word to Vector), on a deep learning model with Long short-term memory. LSTM model is trained at different learning rates or the number of epochs to record values concerning the accuracy, recall, f1-score, and error rate.

Word 2 Vector – Long Short-Term Memory					
Epoch/Learning Rate	Accuracy	Precision	Recall	F1-Score	Error Rate %
1	0.8191	0.83	0.81	0.81	0.1809
2	0.8416	0.84	0.85	0.84	0.1584
3	0.8685	0.86	0.86	0.86	0.1315
4	0.8712	0.87	0.87	0.87	0.1288
5	0.8525	0.85	0.85	0.85	0.1475
6	0.8794	0.87	0.87	0.87	0.1206

Table 7 Evaluation of LSTM using Word2Vec Feature Extraction

For this experiment, the deep learning model LSTM was used for training and testing the dataset. The dataset used was similar to the dataset used in all the other experiments, and the train and test size of the dataset was also the same of 70% training and 30%

testing. The experiment was performed to find the highest accuracy by training the algorithm at different learning rates.
Table 7 illustrates the experimental results generated by the model in terms of a performance measure, and the error rate was calculated manually. The size of dimensions used in this experiment is 128, as it is a common practice to use this dimension size while working with neural networks. The dimension size of 64 and 256 can also be used as they both are the power of 32, but this will create a difference in the training time and not affect the accuracy.

Word2Vec extraction technique was used to build an LSTM model as using this technique, we can use predefined vector, or we can create our vectors. After a brief research, most of the Neural Network models used for sentiment analysis on different languages implemented Word2Vec as their core feature extraction technique. Both techniques blend well together and generate extraordinary results.

Based on the findings from this experiment, it can be said that precisely after six iterations, the model reaches its highest possible accuracy and lowest error rate and terminates the program. Overall, the LSTM model managed to generate the highest accuracy rate of 87%.

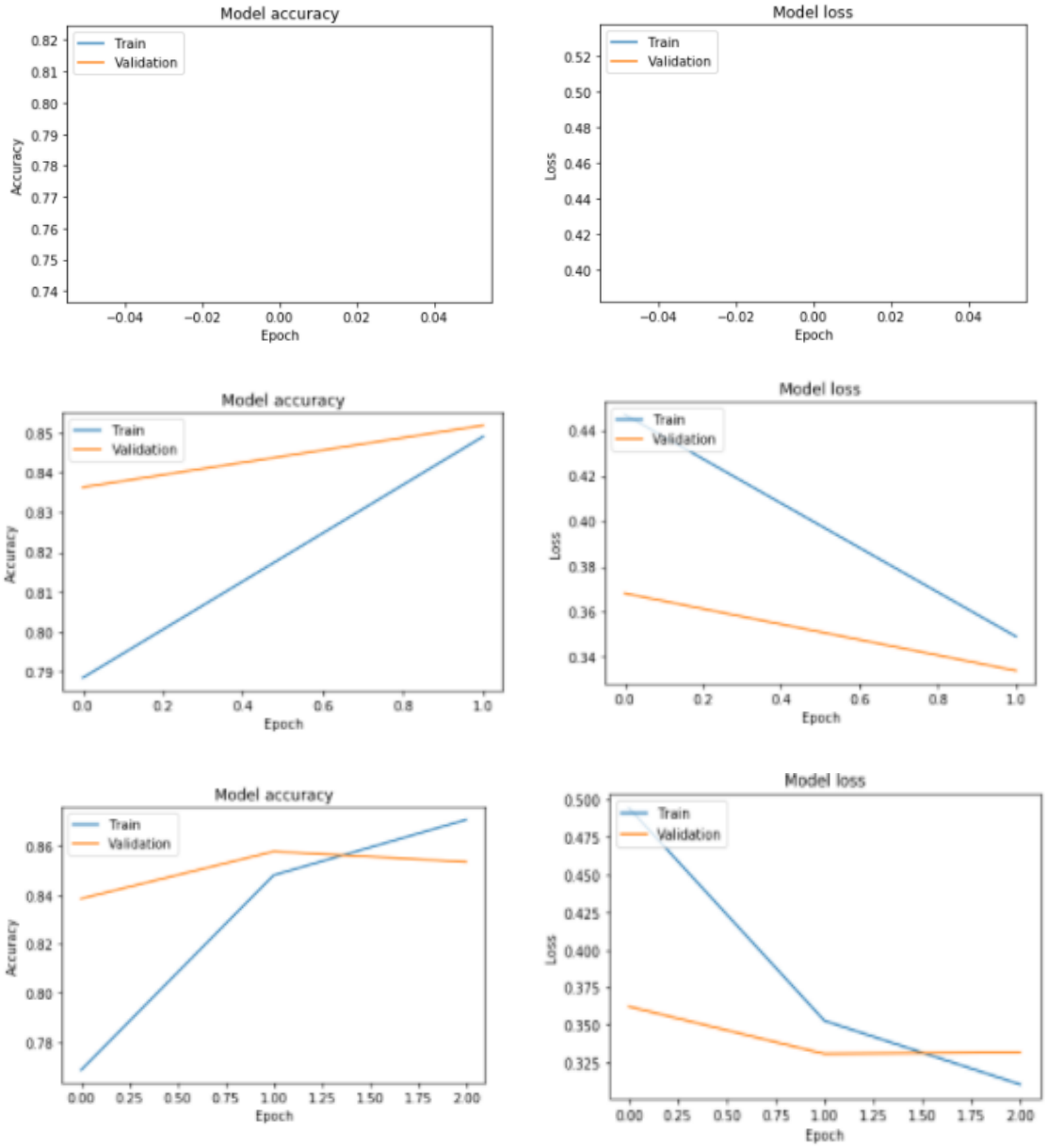
The model was built and executed multiple times at different numbers of epochs. It took almost 24 hours to execute one iteration. In this project, the execution took place 6-time means (21 days) for generating the results shown in the above table, which is a time-consuming process. The LSTM model outperforms all the other competitive models to make a model learn for long-term dependencies. LSTM's ability to remember, forget, and update the information makes it one step ahead of the Recurrent Neural Networks and remaining Machine Learning Algorithms.

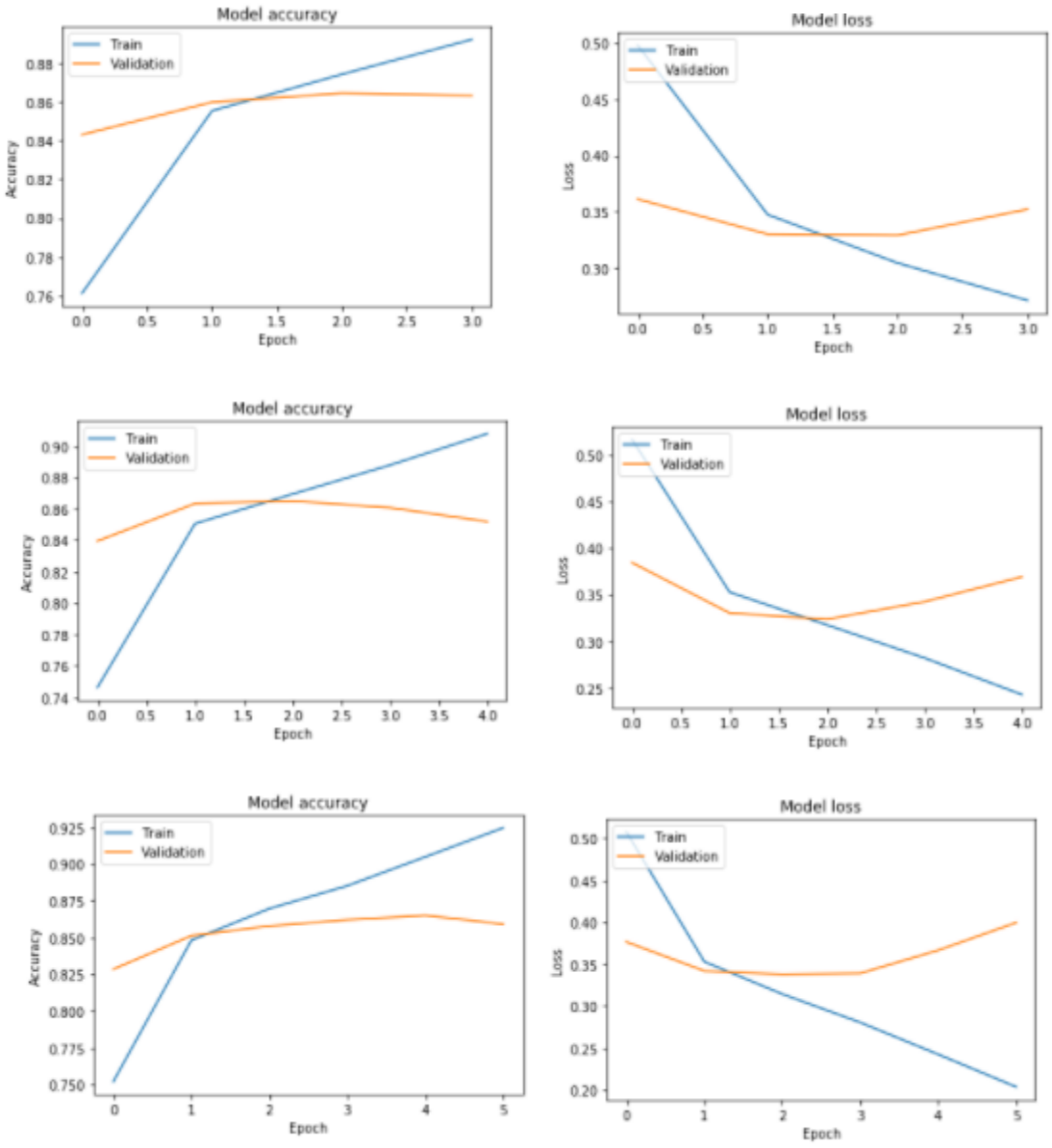
Model: "sequential"		
_____		
Layer (type)	Output Shape	Param #
==========		
embedding (Embedding)	(None, 1769, 128)	15464448
_____		
bidirectional (Bidirectional	(None, 512)	788480
_____		
dense (Dense)	(None, 128)	65664
_____		
dropout (Dropout)	(None, 128)	0
_____		
dense_1 (Dense)	(None, 64)	8256
_____		
dropout_1 (Dropout)	(None, 64)	0
_____		
dense_2 (Dense)	(None, 1)	65

===
Total params: 16,326,913
Trainable params: 862,465
Non-trainable params: 15,464,448

Table 8 Detailed Long-Short Term Memory Summary Table

The table 8 above shows the complete layer summary of the LSTM model created during the implementation time. It contains the values of the total parameters, trainable parameters, and not trainable parameters used. The dimensions of our word embedding layer are 128.





The following are the graphical representation of the LSTM model at each iteration. The ROC curves for both Model Accuracy and Model Loss are represented in a sequential form from table 7. The difference can be noticed in each ROC curve, how accuracy and loss fluctuate due to different learning rates and training of the algorithm. The model performance is best at the learning rate of 6 and terminates when it reaches the highest accuracy generated by the LSTM model.

7. Conclusion and Future Work

The experiments carried out reveal that the LSTM model combined with the Word2Vec technique outperforms the advanced feature extraction and algorithms like TF-IDF and BoW that are commonly practiced in this field.

After these experiments, it can be concluded that sentiment analysis done using word2vecotor and TF-IDF works better in terms of combined accuracy. The classification results generated are comparatively better than BoW.

Sentiment Analysis classification using TF-IDF on Support Vector Machine and Logistic Regression without tuned parameters generates almost the same results, with Logistic Regression performing better with a slight margin. With tuned parameters, SVM performs better than both Logistic Regression and Decision Tree by generating an overall accuracy of 0.8092%.

BoW technique, on the other hand, failed to generate an impressive result with all the machine learning models. SVM and Logistic Regression performs below average but works almost the same with maximum accuracy of 0.67%. Decision Tree unexpectedly failed to generate excellent outcomes with both the feature extraction technique since SVM achieved the highest accuracy rate.

Finding the optimal cost value was also conducted on SVM algorithm and blended with TF-IDF and BoW techniques. However, the LSTM model individually with Word2Vec feature extraction generated an accuracy of 0.8794% with the lowest error rate of 0.1206% at the learning rate of 6 epochs. When it comes to Machine Learning, SVM can be implemented for efficient results, and the key would be to use the optimal value of cost and gamma.

It can be said that the impact of NLP on the Urdu text is maximal and the results generated on some techniques and algorithms are satisfactory and gives us the prime insights using Urdu as a core language; LSTM as a deep learning model using Word2Vec can be called as the most suitable model for sentiment analysis on the Urdu text and SVM can be an alternate model while using TF-IDF as a feature extraction technique.

The dataset used in this project was of only 50 thousand annotated Urdu movie reviews. However, due to not having any other dataset of similar size or larger, the experiments were limited, but further research can be carried out on more significant and more than 2 class problems and the third annotation of the "Neutral" class. Deep learning models like 1D Convolutional Neural Networks or Recurrent Neural Networks can also be experimented for better comparison of the results. LSTM can be tweaked and can be experimented with dimensions of 64 and 256 for further analysis. SVM-based sentiment analysis can be integrated and modified with techniques and with a combination of different parameters. Currently, this project works on using only one dataset, and the results and experiments are limited due to not having enough data on this language. The Urdu language possesses the potential to be worked on, and the research can be extended for better and stun results.

References

Ali Daud, W. K. C., 2016. Urdu language processing: a survey.

Ali, Z., 2019. *Medium.* [Online]
Available at: https://medium.com/@zafaralibagh6/a-simple-word2vec-tutorial-61e64e38a6a1
[Accessed 2021].

Anon., 2015. *colah's blog.* [Online]
Available at: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
[Accessed 2021].

Anon., 2018. *freecodecamp.* [Online]
Available at: https://www.freecodecamp.org/news/an-introduction-to-bag-of-words-and-how-to-code-it-in-python-for-nlp-282e87a9da04/

Anon., 2019. *kambria.io.* [Online]
Available at: https://kambria.io/blog/logistic-regression-for-machine-learning/
[Accessed 2021].

Anon., n.d. *MonkeyLearn.* [Online]
Available at: https://monkeylearn.com/sentiment-analysis/
[Accessed March 2021].

Bajwa, Z. U. R. a. I. S., 2016. *Lexicon-based Sentiment Analysis for Urdu.* s.l., s.n.

Brownlee, J., 2017. *Machine Learning MAstery.* [Online]
Available at: https://machinelearningmastery.com/gentle-introduction-bag-words-model/#:~:text=A%20bag%2Dof%2Dwords%20model%2C%20or%20BoW%20for%20short,as%20with%20machine%20learning%20algorithms.&text=A%20bag%2Dof%2Dwords%20is,of%20words%20within%20a%20document.
[Accessed 2021].

Chakravarthy, S., 2020. *Towards Data Science.* [Online]
Available at: https://towardsdatascience.com/tokenization-for-natural-language-processing-a179a891bad4

Che2, A. D. ·. W. K. ·. D., 2017. Urdu language processing: a survey. *Artif Intell Rev 47, 279–311.*

Dan Li, J. Q., 2016. *Text Sentiment Analysis Based on Long Short-Term Memory.* Beijing, IEEE.

Ghani, Z. N. ·. S., 2020. Sentiment Analysis on Urdu Tweets Using Markov Chains. *SN Computer Science .*

Goyal, K., 2020. *upGrad blog.* [Online]
Available at: https://www.upgrad.com/blog/data-preprocessing-in-machine-learning/
[Accessed 2020].

Irfan, M., 2020. *UrduNLP.* [Online]
Available at: https://www.urdunlp.com/2020/02/urdu-sentiment-classification.html
[Accessed 2020].

Karim, S. M. M., 2020. *Github.* [Online]
Available at: https://github.com/SyedMuhammadMuhsinKarim/Urdu-Stop-Words/blob/main/stop_words.txt
[Accessed 2020].

Khairullah Khan, A. U. R. A. K. U. K. B. S. K. a. K., 2018. Urdu Sentiment Analysis. *International Journal of Advanced Computer Science and Applications.*

Khan, N. M. a. M. A., 2017. Urdu Sentiment Analysis Using Supervised Machine. *World Scientific Publishing Company.*

Khan, N. M. a. M. A., 2017. Urdu Sentiment Analysis Using Supervised Machine. *World Scientific .*

Long Ma, Y. Z., 2015. *Using Word2Vec to process big text data.* s.l., s.n.

Martinez-Enriquez, A. Z. S. ·. M. A. ·. A. M., 2012. Associating targets with SentiUnits: a step forward. *Artif Intell Rev.*

Mr. Brijain, R. P. ,. M. K. ,. K. R., 2014. A Survey on Decision Tree Algorithm For Classification. *IJEDR.*

Muhammad Yaseen Khan, M. S. N., 2020. *Urdu Sentiment Corpus (v1.0): Linguistic Exploration and Visualization of Labeled Dataset for Urdu Sentiment Analysis.* s.l., s.n.

Narkhede, S., 2018. *Towards Data Science.* [Online]
Available at: https://towardsdatascience.com/understanding-logistic-regression-9b02c2aec102
[Accessed 2021].

Neelam Mukhtara, M. A. K. C., 2018. Lexicon-based approach outperforms Supervised Machine Learning. *elsevier.*

Nicholson, C., n.d. *pathmind.* [Online].

Reddy, V., 2018. *Medium.* [Online]
Available at: https://medium.com/@vasista/sentiment-analysis-using-svm-338d418e3ff1
[Accessed 2021].

S. Abbas Ali, M. D. N. M. A. J. M. M. A. O. A. K., 2016. Salience Analysis of NEWS Corpus using Heuristic Approach. *IJCSNS International Journal of Computer Science and Network Security.*

Sehra, C., 2018. *Medium.* [Online]
Available at: https://chiragsehra42.medium.com/decision-trees-explained-easily-28f23241248
[Accessed 2021].

Stecanella, B., 2019. *MonkeyLearn.* [Online]
Available at: https://monkeylearn.com/blog/what-is-tf-idf/
[Accessed 2021].

survey, U. l. p. a., 2017. Ali Daud, · Wahab Khan, Dunren Che. *Artif Intell Rev.*

Verma, N., 2018. *Kaggle.* [Online]
Available at: https://www.kaggle.com/nirajvermafcb/comparing-various-ml-models-roc-curve-comparison
[Accessed 2020].

Wisam A. Qader, M. M. B. I. A., 2019. *An Overview of Bag of Words;Importance,.* Erbil - IRAQ, Fifth International Engineering Conference on Developments in Civil & Computer Engineering Applications 2019 .

Zobia Rehman, W. A. U. I. B., 2011. Challenges in Urdu Text Tokenization and Sentence Boundary.

Appendix A

Implementation source code for feature extraction (TF-IDF, BoW), Algorithms (SVM, DT, LR), Confusion matrix and ROC Curve

TF-IDF – Support Vector Machine

```
import pandas as pd
import numpy as np
import re
import nltk

# Import Plotting Libararies
import seaborn as sns
import matplotlib.pyplot as plt

# Import Data Preprocessing Libraries
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

# Machine Learning Models
from sklearn import svm

# Model Evaluation Libraries
from sklearn.metrics import classification_report, confusion_matrix

train =  pd.read_csv('C:/Users/EMAAZ/Desktop/Dataset/urdureviews/urdu_reviews_train.csv')
test =   pd.read_csv('C:/Users/EMAAZ/Desktop/Dataset/urdureviews/urdu_reviews_test.csv')
print('Shape of Training Set ' , train.shape,'\nShape of Testing Set ', test.shape)

data =  pd.concat([train, test]).reset_index(drop=True)
print(data.shape)
df =  data.copy()
df.head()
le = LabelEncoder()
le.fit(df['sentiment'])
df['encoded_sentiments'] = le.transform(df['sentiment'])
def removing_unwanted_data(text):

    #format words and remove unwanted characters
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
    text = re.sub(r'\<a href', ' ',text)
    text = re.sub(r'&amp;', '', text)
    text = re.sub(r'[_"\-;()|+&=*%.,!?:#$@\[\]/]', ' ', text)
    text = re.sub(r'<br />', ' ', text)
    text = re.sub(r'\'', ' ', text)
    text = re.sub(r"[:          ]+", " ", text)
    text = re.sub(r"[۰۱۲۵٤٥٦٧٨٩  ]+", " ", text)
    text = re.sub(r"[a-zA-z0-9]+", " ", text)

    text = nltk.WordPunctTokenizer().tokenize(text)

    return text
df['text_cleaned']= list(map(removing_unwanted_data,df.review)) #map -> name and data send in func
df.head()
```

```
# Remove stop words from text
from typing import FrozenSet

# Urdu Language Stop words list
STOP_WORDS: FrozenSet[str] = frozenset("""
آ آئی آئیں آئے آتا آتی آتے آس آمدید آنا آنسہ آنی آنے آپ آگے آہ آہا آیا اب ابھی ابے
ارے اس اسکا اسکی اسکے اسی اسے اف افوہ البتہ الف ان اندر انکا انکی انکے انہوں انہی انہیں اوئے اور اوپر
اوہو اپ اپنا اپنوں اپنی اپنے اپنےآپ اکثر اگر اگرچہ اہاہا ایسا ایسی ایسے ایک بائیں بار بارے بالکل باوجود باہر
بج بجے بخیر بشرطیکہ بعد بعض بغیر بلکہ بن بنا بناؤ بند بڑی بھر بھریں بھی بہت بہتر تاکہ تاہم تب تجھ
تجھی تجھے ترا تری تلک تم تمام تمہارا تمہاروں تمہاری تمہارے تمہیں تو تک تھا تھی تھیں تھے تیرا تیری تیرے
جا جاؤ جائیں جائے جاتا جاتی جاتے جانی جانے جب جبکہ جدھر جس جسے جن جناب جنہوں جنہیں جو جہاں جی جیسا
جیسوں جیسی جیسے حالانکہ حالاں حصہ حضرت خاطر خالی خواہ خوب خود دائیں درمیان دریں دو دوران دوسرا دوسروں دوسری دوں
دکھائیں دی دیئے دیا دیتا دیتی دیتے دیر دینا دینی دینے دیکھو دیں دیے دے ذریعے رکھا رکھتا رکھتی رکھتے رکھنا رکھنی
رکھنے رکھو رکھی رکھے رہ رہا رہتا رہتی رہتے رہنا رہنی رہنے رہو رہی رہیں رہے ساتھ سامنے ساڑھے سب سبھی
سراسر سمیت سوا سوائے سکا سکتا سکتے سہ سہی سی سے شاید شکریہ صاحب صاحبہ صرف ضرور طرح طرف طور علاوہ عین
فقط فلاں فی قبل قطا لئے لائی لائے لاتا لاتی لاتے لانا لانی لانے لایا لو لوجی لوگوں لگ لگا لگتا
لگتی لگی لگیں لگے لہذا لی لیا لیتا لیتی لیتے لیکن لیں لیے لے ماسوا مت مجھ مجھی مجھے محترم محترمہ محض
مرا مرحبا مری مرے مزید مس مسز مسٹر مطابق مل مکرمی مگر مگھر مہربانی میرا میروں میری میرے میں نا نزدیک
نما نہ نہیں نیز نیچے نے و وار واسطے واقعی والا والوں والی والے واہ وجہ ورنہ وغیرہ ولے وگرنہ وہ وہاں
وہی وہیں ویسا ویسے ویں پاس پایا پر پس پلیز پون پونی پونے پھر پہ پہلا پہلی پہلے پیر پیچھے چاہئے
چاہتے چاہیئے چاہے چلا چلو چلیں چلے چناچہ چند چونکہ چکی چکیں چکے ڈالنا ڈالنی ڈالنے ڈالے کئے کا کاش کب کبھی
کدھر کر کرتا کرتی کرتے کرم کرنا کرنے کرو کریں کرے کس کسی کسے کم کن کنہیں کو کوئی کون کونسا
کونسے کچھ کہ کہا کہاں کہہ کہی کہیں کہے کی کیا کیسا کیسے کیونکر کیونکہ کیوں کیے کے گئی گئے گا گنا
گو گویا گی گیا ہائیں ہائے ہاں ہر ہرچند ہرگز ہم ہمارا ہماری ہمارے ہمی ہمیں ہو ہوئی ہوئیں ہوئے ہوا
ہوبہو ہوتا ہوتی ہوتیں ہوتے ہونا ہونگے ہونی ہونے ہوں ہی ہیلو ہیں ہے یا یات یعنی یک یہ یہاں یہی یہیں
""".split())

def remove_stopwords(text: str):
    return " ".join(word for word in text.split() if word not in STOP_WORDS)
from urduhack.models.lemmatizer import lemmatizer
def lemitizeStr(str):
    lemme_str = ""
    temp = lemmatizer.lemma_lookup(str)
    for t in temp:
        lemme_str += t[0] + " "

    return lemme_str
df['review'] =  df['review'].apply(remove_stopwords)
df['lemmatized_text'] = df['review'].apply(lemitizeStr)
df.head()
print("Samples per class: {}".format(np.bincount(df.encoded_sentiments)))
X_train, X_test, Y_train, Y_test = train_test_split(df['lemmatized_text'], df['encoded_sentiments'], test_size
= 0.30, random_state = 7)
print('Shape of X_train', X_train.shape)
print('Shape of X_test', X_test.shape)
print('Shape of Y_train', Y_train.shape)
print('Shape of Y_test', Y_test.shape)
max_feature_num = 300
vectorizer = TfidfVectorizer(max_features=max_feature_num)
train_vecs = vectorizer.fit_transform(X_train)
test_vecs = TfidfVectorizer(max_features=max_feature_num,
vocabulary=vectorizer.vocabulary_).fit_transform(X_test)
```

```
def SVM_classifier(train_vecs, Y_train, test_vecs, Y_test):
    # Training
    SVM = svm.LinearSVC(max_iter=10)
    SVM.fit(train_vecs, Y_train)

    # Testing
    test_predictionSVM = SVM.predict(test_vecs)
    return classification_report(test_predictionSVM, Y_test), confusion_matrix(test_predictionSVM, Y_test)
class_report , conf_matrix = SVM_classifier(train_vecs, Y_train, test_vecs, Y_test)
print('Results of SVM Classifier on TF-IDF Vectorizer')
print(class_report)
print(conf_matrix)
from sklearn.svm import SVC
# train the model on train set
model = SVC()
model.fit(train_vecs, Y_train)

# print prediction results
predictions = model.predict(test_vecs)
print(classification_report(Y_test, predictions))
print(confusion_matrix(Y_test, predictions))
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'C': [1],
              'gamma': [1],
              'kernel': ['linear']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3, cv = 10)

# fitting the model for grid search
grid.fit(train_vecs, Y_train)
# print best parameter after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)
grid_predictions = grid.predict(test_vecs)

# print classification report
print(classification_report(Y_test, grid_predictions))
print(confusion_matrix(Y_test, grid_predictions))
```

BoW – Support Vector Machine

```
import pandas as pd
import numpy as np
import re
import nltk

# Import Plotting Libararies
import seaborn as sns
import matplotlib.pyplot as plt

# Import Data Preprocessing Libraries
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

# Machine Learning Models
from sklearn import svm

# Model Evaluation Libraries
from sklearn.metrics import classification_report, confusion_matrix
train =  pd.read_csv('C:/Users/EMAAZ/Desktop/Dataset/urdureviews/urdu_reviews_train.csv')
test =   pd.read_csv('C:/Users/EMAAZ/Desktop/Dataset/urdureviews/urdu_reviews_test.csv')
print('Shape of Training Set ' , train.shape,'\nShape of Testing Set ', test.shape)
data =  pd.concat([train, test]).reset_index(drop=True)
print(data.shape)
df =  data.copy()
df.head()
le = LabelEncoder()
le.fit(df['sentiment'])
df['encoded_sentiments'] = le.transform(df['sentiment'])
def removing_unwanted_data(text):

    #format words and remove unwanted characters
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
    text = re.sub(r'\<a href', ' ',text)
    text = re.sub(r'&amp;', '', text)
    text = re.sub(r'[_"\-;()|+&=*%.,!?:#$@\[\]/]', ' ', text)
    text = re.sub(r'<br />', ' ', text)
    text = re.sub(r'\'', ' ', text)
    text = re.sub(r"[:          ]+", " ", text)
    text = re.sub(r"[۰۱۲۵¿❀۹۹۸  ]+", " ", text)
    text = re.sub(r"[a-zA-z0-9]+", " ", text)

    text = nltk.WordPunctTokenizer().tokenize(text)

    return text
df['text_cleaned']= list(map(removing_unwanted_data,df.review)) #map -> name and data send in func
df.head()
```

```
# Remove stop words from text
from typing import FrozenSet

# Urdu Language Stop words list
STOP_WORDS: FrozenSet[str] = frozenset("""
آ آئی آئیں آئے آتا آتی آتے آس آمدید آنا آنسہ آنی آنے آپ آگے آہ آہا آیا اب ابھی ابے
ارے اس اسکا اسکی اسکے اسی اسے اف افوہ البتہ الف ان اندر انکا انکی انکے انہوں انہی انہیں اوئے اور اوپر
اوہو اپ اپنا اپنوں اپنی اپنے اپنےآپ اکثر اگر اگرچہ اہاہا ایسا ایسی ایسے ایک بائیں بار بارے بالکل باوجود باہر
بج بجے بخیر بشرطیکہ بعد بعض بغیر بلکہ بن بنا بناؤ بند بڑی بھر بھریں بھی بہت بہتر تاکہ تاہم تب تجھ
تجھی تجھے ترا تری تلک تم تمام تمہارا تمہاروں تمہاری تمہارے تمہیں تو تک تھا تھی تھیں تھے تیرا تیری تیرے
جا جاؤ جائیں جائے جاتا جاتی جاتے جانی جانے جب جبکہ جدھر جس جسے جن جناب جنہوں جنہیں جو جہاں جی جیسا
جیسوں جیسی جیسے حالانکہ حالاں حصہ حضرت خاطر خالی خواہ خوب خود دائیں درمیان دریں دو دوران دوسرا دوسروں دوسری دوں
دکھائیں دی دیئے دیا دیتا دیتی دیتے دیر دینا دینی دینے دیکھو دیں دیے دے ذریعے رکھا رکھتا رکھتی رکھتے رکھنا رکھنی
رکھنے رکھو رکھی رکھے رہ رہا رہتا رہتی رہتے رہنا رہنی رہنے رہو رہی رہیں رہے ساتھ سامنے ساڑھے سب سبھی
سراسر سمیت سوا سوائے سکا سکتا سکتے سہ سہی سی سے شاید شکریہ صاحب صاحبہ صرف ضرور طرح طرف طور علاوہ عین
فقط فلاں فی قبل قطا لئے لائی لائے لاتا لاتی لاتے لانا لانی لانے لایا لو لوجی لوگوں لگ لگا لگتا
لگتی لگی لگیں لگے لہذا لی لیا لیتا لیتی لیتے لیکن لیں لیے لے ماسوا مت مجھ مجھی مجھے محترم محترمہ محض
مرا مرحبا مری مرے مزید مس مسز مسٹر مطابق مل مکرمی مگر مگھر مہربانی میرا میروں میری میرے میں نا نزدیک
نما نہ نہیں نیز نیچے نے و وار واسطے واقعی والا والوں والی والے واہ وجہ ورنہ وغیرہ ولے وگرنہ وہ وہاں
وہی وہیں ویسا ویسے وہیں پاس پایا پر پس پلیز پون پونی پونے پھر پہ پہلا پہلی پہلے پیر پیچھے چاہئے
چاہتے چاہیئے چاہے چلا چلو چلیں چلے چناچہ چند چونکہ چکی چکیں چکے ڈالنا ڈالنی ڈالنے ڈالے کئے کا کاش کب کبھی
کدھر کر کرتا کرتی کرتے کرم کرنا کرنے کرو کریں کرے کس کسی کسے کم کن کنہیں کو کوئی کون کونسا
کونسے کچھ کہ کہا کہاں کہہ کہی کہیں کہے کی کیا کیسا کیسے کیونکر کیونکہ کیوں کیے کے گئی گئے گا گنا
گو گویا گی گیا ہائیں ہائے ہاں ہر ہرچند ہرگز ہم ہمارا ہماری ہمارے ہمی ہمیں ہو ہوئی ہوئیں ہوئے ہوا
ہوبہو ہوتا ہوتی ہوتیں ہوتے ہونا ہونگے ہونی ہونے ہوں ہی ہیلو ہیں ہے یا یات یعنی یک یہ یہاں یہی یہیں
""".split())


def remove_stopwords(text: str):
    return " ".join(word for word in text.split() if word not in STOP_WORDS)
from urduhack.models.lemmatizer import lemmatizer
def lemitizeStr(str):
    lemme_str = ""
    temp = lemmatizer.lemma_lookup(str)
    for t in temp:
        lemme_str += t[0] + " "

    return lemme_str
df['review'] =  df['review'].apply(remove_stopwords)
df['lemmatized_text'] = df['review'].apply(lemitizeStr)
df.head()
print("Samples per class: {}".format(np.bincount(df.encoded_sentiments)))
X_train, X_test, Y_train, Y_test = train_test_split(df['lemmatized_text'], df['encoded_sentiments'], test_size
= 0.30, random_state = 7)
BOW_convert = CountVectorizer(tokenizer=lambda doc: doc, lowercase=False)
x = BOW_convert.fit_transform(df['lemmatized_text'])

words = BOW_convert.get_feature_names()
print("The total number of columns are =",len(words))
print('Shape of arrays =',x.toarray().shape)
```

```
import sklearn.model_selection

train,test= sklearn.model_selection.train_test_split(df, train_size = 0.3, random_state=4) #Split and
Shuffle

print('Training data set shape =',train.shape)
print('Testing data set shape =', test.shape)
BOW_convert = CountVectorizer(tokenizer=lambda doc: doc, lowercase=False)

X_train = BOW_convert.fit_transform(train['lemmatized_text'])

X_test = BOW_convert.transform(test['lemmatized_text'])

Y_train=train['encoded_sentiments']
Y_test=test['encoded_sentiments']
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
from sklearn.svm import SVC
# train the model on train set
model = SVC()
model.fit(X_train, Y_train)

# print prediction results
predictions = model.predict(X_test)
print(classification_report(Y_test, predictions))
print(confusion_matrix(Y_test, predictions))
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'C': [1],
              'gamma': [0.1],
              'kernel': ['linear']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3, cv = 10)

# fitting the model for grid search
grid.fit(X_train, Y_train)
# print best parameter after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)
grid_predictions = grid.predict(X_test)

# print classification report
print(classification_report(Y_test, grid_predictions))
print(confusion_matrix(Y_test, grid_predictions))
```

TF-IDF – Decision Tree

```
import nltk
import urduhack
import re
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction import text
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn import metrics
import warnings
warnings.filterwarnings("ignore")
urdu_doc = pd.read_csv('C:/Users/EMAAZ/Desktop/Dataset/UrduReview.csv' , encoding ='utf8')
urdu_doc = urdu_doc.dropna()
urdu_doc.shape
urdu_doc.head()
urdu_doc['sentiment'].value_counts()
# Encode the labels
df = urdu_doc
le = LabelEncoder()
le.fit(df['sentiment'])
df['encoded_sentiments'] = le.transform(df['sentiment'])
def removing_unwanted_data(text):
    #format words and remove unwanted characters
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
    text = re.sub(r'\<a href', ' ',text)
    text = re.sub(r'&amp;', '', text)
    text = re.sub(r'[_"\-;()|+&=*%.,!?:#$@\[\]/]', ' ', text)
    text = re.sub(r'<br />', ' ', text)
    text = re.sub(r'\'', ' ', text)
    text = re.sub(r"[:          ]+", " ", text)
    text = re.sub(r"[۰۱۲۳۴۵۶۷۸۹۸  ]+", " ", text)
    text = re.sub(r"[a-zA-z0-9]+", " ", text)
    text = nltk.WordPunctTokenizer().tokenize(text)
    return text
```

```
urdu_doc['text_cleaned']= list(map(removing_unwanted_data,urdu_doc.review)) #map -> name and data send in func
urdu_doc.head()
# Remove stop words from text
from typing import FrozenSet

# Urdu Language Stop words list
STOP_WORDS: FrozenSet[str] = frozenset("""
آ آئی آئیں آئے آتا آتی آتے آس آمدی آنا آنسہ آنی آنے آپ آگے آہ آہا آیا اب ابھی ابے
ارے اس اسکا اسکی اسکے اسی اسے اف افوہ البتہ الف ان اندر انکا انکی انکے انہوں انہی انہیں اوئے اور اوپر
اوہو اپ اپنا اپنوں اپنی اپنے اپنےآپ اکثر اگر اگرچہ اہاہا ایسا ایسی ایسے ایک بائیں بار بارے بالکل باوجود باہر
بج بجے بخیر بشرطیکہ بعد بعض بغیر بلکہ بن بنا بناؤ بند بڑی بڑے بھر بھریں بھی بہت بہتر بہتیرا تاکہ تاہم تب تجھ
تجھی تجھے ترا تری تلک تم تمام تمہارا تمہاروں تمہاری تمہارے تمہیں تو تک تھا تھی تھیں تھے تیئں تیرا تیری تیرے
جا جاؤ جائیں جائے جاتا جاتی جاتے جاری جانے جب جبکہ جدھر جس جسے جن جناب جنہوں جنہیں جو جہاں جی جیسا
جیسوں جیسی جیسے حالانکہ حالاں حصہ حضرت خاطر خالی خواہ خوب خود دائیں درمیان دریں دو دوران دوسرا دوسروں دوسری دوں
دکھائیں دی دیئے دیا دیتا دیتی دیتے دیں دینا دینی دینے دیکھو دیں دیئے دے ذریعے رکھا رکھتا رکھتی رکھتے رکھنا رکھنی
رکھنے رکھو رکھی رکھے رہ رہا رہتا رہتی رہتے رہنا رہنی رہنے رہو رہی رہیں رہے ساتھ سامنے ساڑھے سب سبھی
سراسر سمیت سوا سوائے سکا سکتا سکتے سہ سہی سی سے شائی شکریہ صاحب صاحبہ صرف ضرور طرف طور طرح علاوہ عین
فقط فلاں فی قبل قطا لئے لائی لائے لاتا لاتی لاتے لانا لانی لانے لایا لو لوجی لوگوں لگ لگا لگتا
لگتی لگی لگے لہذا لی لیا لیتا لیتی لیتے لیکن لیں لیے لے ماسوا مت مجھ مجھی مجھے محترم محترمہ محض
مرا مرحبا مری مرے مزید مس مسز مسٹر مطابق مل مکرمی مگر مگھر مہربانی میرا میروں میری میرے میں نا نزدیک
نما نہ نہی نہیں نیچے نے و وار واسطے واقعی والا والوں والی والے واہ وجہ ورنہ وغیرہ ولے وگرنہ وہ وہاں
وہی وہیں ویسا ویسے وی پاس پایا پر پس پلیز پون پوری پونے پھر پہ پہلا پہلی پہلے پہی پیچھے چاہئے
چاہتے چاہئے چاہے چلا چلو چلی چلے چناچہ چند چونکہ چکی چکیں چکے ڈالنا ڈالنی ڈالنے ڈالے کئے کا کاش کب کبھی
کدھر کر کرتا کرتی کرتے کرم کرنا کرنے کرو کری کرے کس کسی کسے کم کن کنہی کو کوئی کون کونسا
کونسے کچھ کہ کہا کہاں کہہ کہی کہیں کہے کی کیا کیسا کیسے کیونکر کیونکہ کیوں کیے کے گئی گئے گا گنا
گو گویا گی گیا ہائے ہاں ہر ہرچند ہرگز ہم ہمارا ہماری ہمارے ہمی ہمیں ہو ہوئی ہوئیں ہوئے ہوا
ہوبہو ہوتا ہوتی ہوتیں ہوتے ہونا ہونگے ہوئی ہونے ہوں ہی ہیں ہے یا یات یعنی یک یہ یہاں یہی یہیں
""".split())

def remove_stopwords(text: str):
    return " ".join(word for word in text.split() if word not in STOP_WORDS)
from urduhack.models.lemmatizer import lemmatizer
def lemitizeStr(str):
    lemme_str = " "
    temp = lemmatizer.lemma_lookup(str)
    for t in temp:
        lemme_str += t[0] + " "
    return lemme_str
df['review'] =  df['review'].apply(remove_stopwords)
df['lemmatized_text'] = df['review'].apply(lemitizeStr)
urdu_doc.head()
print("Samples per class: {}".format(np.bincount(urdu_doc.encoded_sentiments)))
train_review, test_review, train_sentiments, test_sentiments = train_test_split(df['lemmatized_text'],
df['encoded_sentiments'], test_size=0.3, random_state = 6, shuffle = True)
# training: tf-idf
max_feature_num = 300
train_vectorizer = TfidfVectorizer(max_features=max_feature_num)
train_vecs = train_vectorizer.fit_transform(train_review)
test_vecs =
TfidfVectorizer(max_features=max_feature_num,vocabulary=train_vectorizer.vocabulary_).fit_transform(t
est_review)
```

```
from sklearn.tree import DecisionTreeClassifier

model_tree = DecisionTreeClassifier()
model_tree.fit(train_vecs, train_sentiments)
y_prob = model_tree.predict_proba(test_vecs)[:,1] # This will give you positive class prediction
probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to give class predictions.
model_tree.score(test_vecs, y_pred)
from sklearn.tree import DecisionTreeClassifier

model_DD = DecisionTreeClassifier()


tuned_parameters= {'criterion': ['gini','entropy'], 'max_features': ["auto","sqrt","log2"],
                   'min_samples_leaf': range(1,100,1) , 'max_depth': range(1,50,1)
                  }
DD_model= RandomizedSearchCV(model_DD,
tuned_parameters,cv=10,scoring='accuracy',n_iter=1,n_jobs= -1,random_state=5)
DD_model.fit(train_vecs, train_sentiments)
print(DD_model.best_score_)
print(DD_model.best_params_)
y_prob = DD_model.predict_proba(test_vecs)[:,1] # This will give you positive class prediction
probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to give class predictions.
DD_model.score(test_vecs, y_pred)
```

BoW – Decision Tree

```
import nltk
import urduhack
import re
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction import text
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn import metrics
import warnings
warnings.filterwarnings("ignore")
urdu_doc = pd.read_csv('C:/Users/EMAAZ/Desktop/Dataset/UrduReview.csv' , encoding ='utf8')
urdu_doc = urdu_doc.dropna()
urdu_doc.shape
urdu_doc.head()
urdu_doc['sentiment'].value_counts()
# Encode the labels
df = urdu_doc
le = LabelEncoder()
le.fit(df['sentiment'])
df['encoded_sentiments'] = le.transform(df['sentiment'])
def removing_unwanted_data(text):

    #format words and remove unwanted characters
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
    text = re.sub(r'\<a href', ' ',text)
    text = re.sub(r'&amp;', '', text)
    text = re.sub(r'[_"\-;()|+&=*%.,!?:#$@\[\]/]', ' ', text)
    text = re.sub(r'<br />', ' ', text)
    text = re.sub(r'\'', ' ', text)
    text = re.sub(r"[:          ]+", " ", text)
    text = re.sub(r"[۰۱۲۳۴۵۶۷۸۹۸  ]+", " ", text)
    text = re.sub(r"[a-zA-z0-9]+", " ", text)

    text = nltk.WordPunctTokenizer().tokenize(text)

    return text
urdu_doc['text_cleaned']= list(map(removing_unwanted_data,urdu_doc.review)) #map -> name and data
send in func
urdu_doc.head()
```

```
# Remove stop words from text
from typing import FrozenSet

# Urdu Language Stop words list
STOP_WORDS: FrozenSet[str] = frozenset("""
آ آئی آئیں آئے آتا آتی آتے آس آمدنی آنا آنسہ آنی آنے آپ آگے آہ آہا آیا اب ابھی ابے
ارے اس اسکا اسکی اسکے اسی اسے اف افوہ البتہ الف ان اندر انکا انکی انکے انہوں انہی انہیں اوئے اور اوپر
اوہو اپ اپنا اپنوں اپنی اپنے اپنےآپ اکثر اگر اگرچہ اہاہا ایسا ایسی ایسے ایک باتیں بار بارے بالکل باوجود باہر
بج بجے بخیر بشرطیکہ بعد بعض بغیر بلکہ بن بنا بناؤ بند بڑی بھر بھریں بھی بہت بہتر تاکہ تاہم تب تجھ
تجھی تجھے ترا تری تلک تم تمام تمہارا تمہاروں تمہاری تمہارے تمہیں تو تک تھا تھی تھیں تھے تیرا تیری تیرے
جا جاؤ جاتا جاتے جاتی جاتیں جائے جانا جانب جانے جب جبکہ جدھر جس جسے جن جناب جنہوں جنہیں جو جہاں جی جیسا
جیسوں جیسی جیسے حالانکہ حالاں حصہ حضرت خاطر خالی خواہ خوب خود داتا دراں دریں دو دوران دوسرا دوسروں دوسری دوں
دکھائیں دی دیئے دیا دیتا دیتی دیتے دیں دینا دینی دینے دیکھو دیں دیے دے ذریعے رکھا رکھتا رکھتی رکھتے رکھنا رکھنی
رکھنے رکھو رکھی رکھے رہ رہا رہتا رہتی رہتے رہنا رہنی رہنے رہو رہی رہیں رہے ساتھ سامنے ساڑھے سب سبھی
سراسر سمیت سوا سوائے سکا سکتا سکتے سہ سہی سی سے شاید شکریہ صاحب صاحبہ صرف ضرور طرح طرف طور علاوہ عین
فقط فلاں فی قبل قطا لئے لاتا لاتی لائے لاتے لانا لانی لانے لاکھ لو لوجی لوگوں لگ لگا لگتا
لگتی لگتی لگے لہذا لی لیا لیتا لیتی لیتے لیکن لیں لیے لے ماسوا مت مجھ مجھی مجھے محترم محترمہ محض
مرا مرحبا مری مرے مزید مس مسز مسٹر مطابق مل مکرمی مگر مہربانی میرا میروں میری میرے میں نا نزدیک
نما نہ نہیں نیچے نے و وار واسطے واقعی والا والوں والی والے واہ وجہ ورنہ وغیرہ ولے وگرنہ وہ وہاں
وہی وہیں ویسا ویسے وی پاس پاکھ پر پس پلیز پون پونی پونے پھر پہ پہلا پہلی پہلے پہں پیچھے چاہئے
چاہتے چاہیئے چاہے چلا چلو چلی چلے چناچہ چند چونکہ چکی چکیں چکے ڈالا ڈالنا ڈالنے ڈالے کئے کا کاش کب کبھی
کدھر کر کرتا کرتی کرتے کرم کرنا کرنے کرو کری کرے کس کسی کسے کم کن کنہیں کو کوئی کون کونسا
کونسے کچھ کہ کہا کہاں کہہ کہی کہیں کہے کی کیا کیسا کیسے کیونکر کیونکہ کیوں کیے کے گئی گئے گا گنا
گو گویا گیا گیاہائے ہاں ہر ہرچند ہرگز ہم ہمارا ہماری ہمارے ہمی ہمیں ہو ہوئی ہوئیں ہوئے ہوا
ہوبہو ہوتا ہوتی ہوتیں ہوتے ہونا ہونگے ہوں ہی ہیں ہے یا یات یعنی یک یہ یہاں یہی یہیں
""".split())


def remove_stopwords(text: str):
    return " ".join(word for word in text.split() if word not in STOP_WORDS)
from urduhack.models.lemmatizer import lemmatizer
def lemitizeStr(str):
    lemme_str = " "
    temp = lemmatizer.lemma_lookup(str)
    for t in temp:
        lemme_str += t[0] + " "

    return lemme_str
df['review'] =  df['review'].apply(remove_stopwords)
df['lemmatized_text'] = df['review'].apply(lemitizeStr)
urdu_doc.head()
print("Samples per class: {}".format(np.bincount(urdu_doc.encoded_sentiments)))
X_train, X_test, Y_train, Y_test = train_test_split(df['lemmatized_text'], df['encoded_sentiments'], test_size
= 0.30, random_state = 7)
print('Shape of X_train', X_train.shape)
print('Shape of X_test', X_test.shape)
print('Shape of Y_train', Y_train.shape)
print('Shape of Y_test', Y_test.shape)
```

```
BOW_convert = CountVectorizer(tokenizer=lambda doc: doc, lowercase=False)
x = BOW_convert.fit_transform(df['lemmatized_text'])

words = BOW_convert.get_feature_names()
print("The total number of columns are =",len(words))
print('Shape of arrays =',x.toarray().shape)
import sklearn.model_selection

train,test= sklearn.model_selection.train_test_split(df, train_size = 0.7, random_state=4) #Split and
Shuffle

print('Training data set shape =',train.shape)
print('Testing data set shape =', test.shape)
BOW_convert = CountVectorizer(tokenizer=lambda doc: doc, lowercase=False)

X_train = BOW_convert.fit_transform(train['lemmatized_text'])

X_test = BOW_convert.transform(test['lemmatized_text'])

Y_train=train['encoded_sentiments']
Y_test=test['encoded_sentiments']
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
from sklearn.tree import DecisionTreeClassifier

model_tree = DecisionTreeClassifier()
model_tree.fit(X_train, Y_train)
y_prob = model_tree.predict_proba(X_test)[:,1] # This will give you positive class prediction probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to give class predictions.
model_tree.score(X_test, y_pred)

from sklearn.tree import DecisionTreeClassifier

model_DD = DecisionTreeClassifier()


tuned_parameters= {'criterion': ['gini','entropy'], 'max_features': ["auto","sqrt","log2"],
                   'min_samples_leaf': range(1,100,1) , 'max_depth': range(1,50,1)
                  }
DD_model= RandomizedSearchCV(model_DD,
tuned_parameters,cv=45,scoring='accuracy',n_iter=10,n_jobs= -1,random_state=5)
DD_model.fit(X_train, Y_train)
print(DD_model.best_score_)
print(DD_model.best_params_)
y_prob = DD_model.predict_proba(X_test)[:,1] # This will give you positive class prediction probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to give class predictions.
DD_model.score(X_test, y_pred)
```

TF-IDF – Logistic Regression

```
import nltk
import urduhack
import re
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction import text
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn import metrics
import warnings
warnings.filterwarnings("ignore")
urdu_doc = pd.read_csv('C:/Users/EMAAZ/Desktop/Dataset/UrduReview.csv' , encoding ='utf8')
urdu_doc = urdu_doc.dropna()
urdu_doc.shape
urdu_doc.head()
urdu_doc['sentiment'].value_counts()
# Encode the labels
df = urdu_doc
le = LabelEncoder()
le.fit(df['sentiment'])
df['encoded_sentiments'] = le.transform(df['sentiment'])
def removing_unwanted_data(text):

    #format words and remove unwanted characters
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
    text = re.sub(r'\<a href', ' ',text)
    text = re.sub(r'&amp;', '', text)
    text = re.sub(r'[_"\-;()|+&=*%.,!?:#$@\[\]/]', ' ', text)
    text = re.sub(r'<br />', ' ', text)
    text = re.sub(r'\'', ' ', text)
    text = re.sub(r"[:          ]+", " ", text)
    text = re.sub(r"[۰۱۲۳۴۵۶۷۸۹  ]+", " ", text)
    text = re.sub(r"[a-zA-z0-9]+", " ", text)

    text = nltk.WordPunctTokenizer().tokenize(text)

    return text
urdu_doc['text_cleaned']= list(map(removing_unwanted_data,urdu_doc.review)) #map -> name and data
send in func
urdu_doc.head()
```

```
# Remove stop words from text
from typing import FrozenSet

# Urdu Language Stop words list
STOP_WORDS: FrozenSet[str] = frozenset("""
آ آئی آئیں آئے آتا آتی آتے آس آمدید آنا آنسہ آنی آنے آپ آگے آہ آہا آیا اب ابھی ابے
ارے اس اسکا اسکی اسکے اسی اسے اف افوہ البتہ الف ان اندر انکا انکی انکے انہوں انہی انہیں اوئے اور اوپر
اوہو اپ اپنا اپنوں اپنی اپنے اپنےآپ اکثر اگر اگرچہ اہاہا ایسا ایسی ایسے ایک بائیں بار بارے بالکل باوجود باہر
بج بجے بخیر بشرطیکہ بعد بعض بغیر بلکہ بن بنا بناؤ بند بڑی بھر بھریں بھی بہت بہتر تاکہ تاہم تب تجھ
تجھی تجھے ترا تری تلک تم تمام تمہارا تمہاروں تمہاری تمہارے تمہیں تو تک تھا تھی تھیں تھے تیرا تیری تیرے
جا جاؤ جائیں جائے جاتا جاتی جاتے جانی جانے جب جبکہ جدھر جس جسے جن جناب جنہوں جنہیں جو جہاں جی جیسا
جیسوں جیسی جیسے حالانکہ حالاں حصہ حضرت خاطر خالی خواہ خوب خود دائیں درمیان دریں دو دوران دوسرا دوسروں دوسری دوں
دکھائیں دی دیئے دیا دیتا دیتی دیتے دیر دینا دینی دینے دیکھو دیں دیے دے ذریعے رکھا رکھتا رکھتی رکھتے رکھنا رکھنی
رکھنے رکھو رکھی رکھے رہ رہا رہتا رہتی رہتے رہنا رہنی رہنے رہو رہی رہیں رہے ساتھ سامنے ساڑھے سب سبھی
سراسر سمیت سوا سوائے سکا سکتا سکتے سہ سہی سی سے شاید شکریہ صاحب صاحبہ صرف ضرور طرح طرف طور علاوہ عین
فقط فلاں فی قبل قطا لئے لائی لائے لاتا لاتی لاتے لانا لانی لانے لایا لو لوجی لوگوں لگ لگا لگتا
لگتی لگی لگیں لگے لہذا لی لیا لیتا لیتی لیتے لیکن لیں لیے لے ماسوا مت مجھ مجھی مجھے محترم محترمہ محض
مرا مرحبا مری مرے مزید مس مسز مسٹر مطابق مل مکرمی مگر مگھر مہربانی میرا میروں میری میرے میں نا نزدیک
نما نہ نہیں نیز نیچے نے و وار واسطے واقعی والا والوں والی والے واہ وجہ ورنہ وغیرہ ولے وگرنہ وہ وہاں
وہی وہیں ویسا ویسے ویں پاس پایا پر پس پلیز پون پونی پونے پھر پہ پہلا پہلی پہلے پیر پیچھے چاہئے
چاہتے چاہیئے چاہے چلا چلو چلیں چلے چناچہ چند چونکہ چکی چکیں چکے ڈالنا ڈالنی ڈالنے ڈالے کئے کا کاش کب کبھی
کدھر کر کرتا کرتی کرتے کرم کرنا کرنی کرنے کرو کریں کرے کس کسی کسے کم کن کنہیں کو کوئی کون کونسا
کونسے کچھ کہ کہا کہاں کہہ کہی کہیں کہے کی کیا کیسا کیسے کیونکر کیونکہ کیوں کیے کے گئی گئے گا گنا
گو گویا گی گیا ہائیں ہائے ہاں ہر ہرچند ہرگز ہم ہمارا ہماری ہمارے ہمی ہمیں ہو ہوئی ہوئیں ہوئے ہوا
ہوبہو ہوتا ہوتی ہوتیں ہوتے ہونا ہونگے ہونی ہونے ہوں ہی ہیلو ہیں ہے یا یات یعنی یک یہ یہاں یہی یہیں
""".split())


def remove_stopwords(text: str):
    return " ".join(word for word in text.split() if word not in STOP_WORDS)
from urduhack.models.lemmatizer import lemmatizer
def lemitizeStr(str):
    lemme_str = " "
    temp = lemmatizer.lemma_lookup(str)
    for t in temp:
        lemme_str += t[0] + " "

    return lemme_str
df['review'] =  df['review'].apply(remove_stopwords)
df['lemmatized_text'] = df['review'].apply(lemitizeStr)
urdu_doc.head()
print("Samples per class: {}".format(np.bincount(urdu_doc.encoded_sentiments)))
train_review, test_review, train_sentiments, test_sentiments = train_test_split(df['lemmatized_text'],
df['encoded_sentiments'], test_size=0.3, random_state = 6, shuffle = True)
# training: tf-idf
max_feature_num = 300
train_vectorizer = TfidfVectorizer(max_features=max_feature_num)
train_vecs = train_vectorizer.fit_transform(train_review)
test_vecs =
TfidfVectorizer(max_features=max_feature_num,vocabulary=train_vectorizer.vocabulary_).fit_transform(t
est_review)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn import metrics

model_LR= LogisticRegression()
model_LR.fit(train_vecs,train_sentiments)
y_prob = model_LR.predict_proba(test_vecs)[:,1] # This will give you positive class prediction
probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to give class predictions.
model_LR.score(test_vecs, y_pred)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn import metrics

LR_model= LogisticRegression()

tuned_parameters = {'C': [0.1] ,
            'penalty':['l1']
                }
from sklearn.model_selection import GridSearchCV

LR= GridSearchCV(LR_model, tuned_parameters,cv=10)
LR.fit(train_vecs,train_sentiments)
print(LR.best_params_)
print(LR.best_score_)
y_prob = LR.predict_proba(test_vecs)[:,1] # This will give you positive class prediction probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to give class predictions.
LR.score(test_vecs, y_pred)
```

BoW – Logistic Regression

```
import nltk
import urduhack
import re
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction import text
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn import metrics
import warnings
warnings.filterwarnings("ignore")
urdu_doc = pd.read_csv('C:/Users/EMAAZ/Desktop/Dataset/UrduReview.csv' , encoding ='utf8')
urdu_doc = urdu_doc.dropna()
urdu_doc.shape
urdu_doc.head()
urdu_doc['sentiment'].value_counts()
# Encode the labels
df = urdu_doc
le = LabelEncoder()
le.fit(df['sentiment'])
df['encoded_sentiments'] = le.transform(df['sentiment'])
def removing_unwanted_data(text):

    #format words and remove unwanted characters
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
    text = re.sub(r'\<a href', ' ',text)
    text = re.sub(r'&amp;', '', text)
    text = re.sub(r'[_"\-;()|+&=*%.,!?:#$@\[\]/]', ' ', text)
    text = re.sub(r'<br />', ' ', text)
    text = re.sub(r'\'', ' ', text)
    text = re.sub(r"[:          ]+", " ", text)
    text = re.sub(r"[۰۱۲۵٤٥٩٦٨  ]+", " ", text)
    text = re.sub(r"[a-zA-z0-9]+", " ", text)

    text = nltk.WordPunctTokenizer().tokenize(text)

    return text
urdu_doc['text_cleaned']= list(map(removing_unwanted_data,urdu_doc.review)) #map -> name and data
send in func
urdu_doc.head()
```

```
# Remove stop words from text
from typing import FrozenSet

# Urdu Language Stop words list
STOP_WORDS: FrozenSet[str] = frozenset("""
آ آئی آئیں آئے آتا آتی آتے آس آمدنی آنا آنسہ آنی آنے آپ آگے آہ آہا آیا اب ابھی ابے
ارے اس اسکا اسکی اسکے اسی اسے اف افوہ البتہ الف ان اندر انکا انکی انکے انہوں انہی انہیں اوئے اور اوپر
اوہو اپ اپنا اپنوں اپنی اپنے اپنےآپ اکثر اگر اگرچہ اہاہا ایسا ایسی ایسے ایک بائیں بار بارے بالکل باوجود باہر
بج بجے بخیر بشرطیکہ بعد بعض بغیر بلکہ بن بنا بناؤ بند بڑی بھر بھریں بھی بہت بہتر تاکہ تاہم تب تجھ
تجھی تجھے ترا تری تلک تم تمام تمہارا تمہاروں تمہاری تمہارے تمہیں تو تک تھا تھی تھیں تھے تیئں تیئں ٹھے
جا جاؤ جائیں جائے جاتا جاتی جاتے جاری جانے جب جبکہ جدھر جس جسے جن جناب جنہوں جنہیں جو جہاں جی جیسا
جیسوں جیسی جیسے حالانکہ حالاں حصہ حضرت خاطر خالی خواہ خوب خود دائیں درمیان دریں دو دوران دوسرا دوسروں دوسری دوں
دکھائیں دی دیئے دیا دیتا دیتی دیتے دیں دینا دینی دینے دیکھو دیں دیے دے ذریعے رکھا رکھتا رکھتی رکھتے رکھنا رکھنی
رکھنے رکھو رکھی رکھے رہ رہا رہتا رہتی رہتے رہنا رہنی رہنے رہو رہی رہیں رہے ساتھ سامنے ساڑھے سب سبھی
سراسر سمیت سوا سوائے سکا سکتا سکتے سہ سہی سی سے شائیں شکریہ صاحب صاحبہ صرف ضرور طرف طرح طور علاوہ عین
فقط فلاں فی قبل قطا لئے لائیں لائے لاتا لاتی لاتے لانا لانی لانے لایا لو لوجی لوگوں لگ لگا لگتا
لگتی لگی لگیں لگے لہذا لی لیا لیتا لیتی لیتے لیکن لیں لیے لے ماسوا مت مجھ مجھی مجھے محترم محترمہ محض
مرا مرحبا مری مرے مزید مس مسز مسٹر مطابق مل مکرمی مگر مگھر مہربانی میرا میروں میری میرے میں نا نزدیک
نما نہ نہیں نیچے نے و وار واسطے واقعی والا والوں والی والے واہ وجہ ورنہ وغیرہ ولے وگرنہ وہ وہاں
وہی وہیں وہیا وہیے وی پاس پایا پر پس پلیں پون پونی پونے پھر پہ پہلا پہلی پہلے پیں پیچھے چاہئے
چاہتے چاہئے چاہے چلا چلو چلی چلے چناچہ چند چونکہ چکی چکیں چکے ڈالنا ڈالنی ڈالنے ڈالے کئے کا کاش کب کبھی
کدھر کر کرتا کرتی کرتے کرم کرنا کرنے کرو کری کرے کس کسی کسے کم کن کنہیں کو کوئی کون کونسا
کونسے کچھ کہ کہا کہاں کہہ کہی کہیں کہے کئی کئیں کیسا کیسے کیونکر کیونکہ کیوں کیے کے گئی گئے گا گنا
گو گویا گیا گیاہائیں ہائے ہاں ہر ہرچند ہرگز ہم ہمارا ہماری ہمارے ہمی ہمیں ہو ہوئی ہوئے ہوا
ہوبہو ہوتا ہوتی ہوتیں ہوتے ہونا ہونگے ہوئی ہونے ہوں ہی ہیں ہے یا یات یعنی یک یہ یہاں یہی یہیں
""".split())


def remove_stopwords(text: str):
    return " ".join(word for word in text.split() if word not in STOP_WORDS)
from urduhack.models.lemmatizer import lemmatizer
def lemitizeStr(str):
    lemme_str = " "
    temp = lemmatizer.lemma_lookup(str)
    for t in temp:
        lemme_str += t[0] + " "

    return lemme_str
df['review'] =  df['review'].apply(remove_stopwords)
df['lemmatized_text'] = df['review'].apply(lemitizeStr)
urdu_doc.head()
print("Samples per class: {}".format(np.bincount(urdu_doc.encoded_sentiments)))
X_train, X_test, Y_train, Y_test = train_test_split(df['lemmatized_text'], df['encoded_sentiments'], test_size
= 0.30, random_state = 7)
print('Shape of X_train', X_train.shape)
print('Shape of X_test', X_test.shape)
print('Shape of Y_train', Y_train.shape)
print('Shape of Y_test', Y_test.shape)
```

```
BOW_convert = CountVectorizer(tokenizer=lambda doc: doc, lowercase=False)
x = BOW_convert.fit_transform(df['lemmatized_text'])

words = BOW_convert.get_feature_names()
print("The total number of columns are =",len(words))
print('Shape of arrays =',x.toarray().shape)
import sklearn.model_selection

train,test= sklearn.model_selection.train_test_split(df, train_size = 0.7, random_state=4) #Split and
Shuffle

print('Training data set shape =',train.shape)
print('Testing data set shape =', test.shape)
BOW_convert = CountVectorizer(tokenizer=lambda doc: doc, lowercase=False)

X_train = BOW_convert.fit_transform(train['lemmatized_text'])

X_test = BOW_convert.transform(test['lemmatized_text'])

Y_train=train['encoded_sentiments']
Y_test=test['encoded_sentiments']
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn import metrics

model_LR= LogisticRegression()
model_LR.fit(X_train,Y_train)
y_prob = model_LR.predict_proba(X_test)[:,1] # This will give you positive class prediction probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to give class predictions.
model_LR.score(X_test, y_pred)
print(LR.best_score_)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn import metrics

LR_model= LogisticRegression()

tuned_parameters = {'C': [0.1] ,
            'penalty':['11']
                }
from sklearn.model_selection import GridSearchCV
LR= GridSearchCV(LR_model, tuned_parameters,cv=10)
LR.fit(X_train,Y_train)
print(LR.best_params_)
print(LR.best_score_)
y_prob = LR.predict_proba(X_test)[:,1] # This will give you positive class prediction probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to give class predictions.
LR.score(X_test, y_pred)
```

Confusion Matrix and Roc Curve

```
confusion_matrix=metrics.confusion_matrix(Y_test,y_pred)

confusion_matrix

auc_roc=metrics.classification_report(Y_test,y_pred)
auc_roc
auc_roc=metrics.roc_auc_score(Y_test,y_pred)
auc_roc
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(Y_test, y_prob)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate,true_positive_rate, color='red',label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

Appendix B

Implementation source code for Word2Vec using LSTM

LSTM – Word 2 Vector

```
import pandas as pd
import numpy as np
import re
import nltk

# Import Plotting Libararies
import seaborn as sns
import matplotlib.pyplot as plt

# Import Data Preprocessing Libraries
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import GridSearchCV, StratifiedKFold, learning_curve
from sklearn.svm import LinearSVC

# Machine Learning Models
from sklearn import svm

# Model Evaluation Libraries
from sklearn.metrics import classification_report, confusion_matrix
train =  pd.read_csv('C:/Users/EMAAZ/Desktop/Dataset/urdureviews/urdu_reviews_train.csv')
test =   pd.read_csv('C:/Users/EMAAZ/Desktop/Dataset/urdureviews/urdu_reviews_test.csv')
print('Shape of Training Set ' , train.shape,'\nShape of Testing Set ', test.shape)
data =  pd.concat([train, test]).reset_index(drop=True)
print(data.shape)
df =  data.copy()
df.head()
sns.countplot( x = 'sentiment', data = df );
le = LabelEncoder()
le.fit(df['sentiment'])
df['encoded_sentiments'] = le.transform(df['sentiment'])
def removing_unwanted_data(text):

    #format words and remove unwanted characters
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
    text = re.sub(r'\<a href', ' ',text)
    text = re.sub(r'&amp;', '', text)
    text = re.sub(r'[_"\-;()|+&=*%.,!?:#$@\[\]/]', ' ', text)
    text = re.sub(r'<br />', ' ', text)
    text = re.sub(r'\'', ' ', text)
    text = re.sub(r"[:-ء*'؟؛]+", " ", text)
    text = re.sub(r"[آ-ے ]+", " ", text)
    text = re.sub(r"[a-zA-z0-9]+", " ", text)

    text = nltk.WordPunctTokenizer().tokenize(text)

    return text
df['text_cleaned']= list(map(removing_unwanted_data,df.review)) #map -> name and data send in func
df.head()
# Remove stop words from text
from typing import FrozenSet
```

```
# Remove stop words from text
from typing import FrozenSet

# Urdu Language Stop words list
STOP_WORDS: FrozenSet[str] = frozenset("""
آ آئی آئیں آئے آتا آتی آتے آس آمدید آنا آنسہ آنی آنے آپ آگے آہ آہا آیا اب ابھی ابے
ارے اس اسکا اسکی اسکے اسی اسے اف افوہ البتہ الف ان اندر انکا انکی انکے انہوں انہی انہیں اوئے اور اوپر
اوہو اپ اپنا اپنوں اپنی اپنے اپنےآپ اکثر اگر اگرچہ اہاہا ایسا ایسی ایسے ایک بائیں بار بارے بالکل باوجود باہر
بج بجے بخیر بشرطیکہ بعد بعض بغیر بلکہ بن بنا بناؤ بند بڑی بھر بھریں بھی بہت بہتر تاکہ تاہم تب تجھ
تجھی تجھے ترا تری تلک تم تمام تمہارا تمہاروں تمہاری تمہارے تمہیں تو تک تھا تھی تھیں تھے تیرا تیری تیرے
جا جاؤ جائیں جائے جاتا جاتی جاتے جانی جانے جب جبکہ جدھر جس جسے جن جناب جنہوں جنہیں جو جہاں جی جیسا
جیسوں جیسی جیسے حالانکہ حالاں حصہ حضرت خاطر خالی خواہ خوب خود دائیں درمیان دریں دو دوران دوسرا دوسروں
دوسری دوں
دکھائیں دی دیئے دیا دیتا دیتی دیتے دیر دینا دینی دینے دیکھو دیں دیے دے ذریعے رکھا رکھتا رکھتی رکھتے رکھنا رکھنی
رکھنے رکھو رکھی رکھے رہ رہا رہتا رہتی رہتے رہنا رہنی رہنے رہو رہی رہیں رہے ساتھ سامنے ساڑھے سب سبھی
سراسر سمیت سوا سوائے سکا سکتا سکتے سہ سہی سی سے شاید شکریہ صاحب صاحبہ صرف ضرور طرح طرف طور
علاوہ عین
فقط فلاں فی قبل قطا لئے لائی لائے لاتا لاتی لاتے لانا لانی لانے لایا لو لوجی لوگوں لگ لگا لگتا
لگتی لگی لگیں لگے لہذا لی لیا لیتا لیتی لیتے لیکن لیں لیے لے ماسوا مت مجھ مجھی مجھے محترم محترمہ محض
مرا مرحبا مری مرے مزید مس مسز مسٹر مطابق مل مکرمی مگر مگھر مہربانی میرا میروں میری میرے میں نا نزدیک
نما نہ نہیں نیز نیچے نے و وار واسطے واقعی والا والوں والی والے واہ وجہ ورنہ وغیرہ ولے وگرنہ وہ وہاں
وہی وہیں ویسا ویسے ویں پاس پایا پر پس پلیز پون پونی پونے پھر پہ پہلا پہلی پہلے پیر پیچھے چاہئے
چاہتے چاہیئے چاہے چلا چلو چلیں چلے چناچہ چند چونکہ چکی چکیں چکے ڈالنا ڈالنی ڈالنے ڈالے کئے کا کاش کب کبھی
کدھر کر کرتا کرتی کرتے کرم کرنا کرنے کرو کریں کرے کس کسی کسے کم کن کنہیں کو کوئی کون کونسا
کونسے کچھ کہ کہا کہاں کہہ کہی کہیں کہے کی کیا کیسا کیسے کیونکر کیونکہ کیوں کیے کے گئی گئے گا گنا
گو گویا گی گیا ہائیں ہائے ہاں ہر ہرچند ہرگز ہم ہمارا ہماری ہمارے ہمی ہمیں ہو ہوئی ہوئیں ہوئے ہوا
ہوبہو ہوتا ہوتی ہوتیں ہوتے ہونا ہونگے ہونی ہونے ہوں ہی ہیلو ہیں ہے یا یات یعنی یک یہ یہاں یہی یہیں
""".split())


def remove_stopwords(text: str):
    return " ".join(word for word in text.split() if word not in STOP_WORDS)

from urduhack.models.lemmatizer import lemmatizer
def lemitizeStr(str):
    lemme_str = ""
    temp = lemmatizer.lemma_lookup(str)
    for t in temp:
        lemme_str += t[0] + " "

    return lemme_str
df['review'] =  df['review'].apply(remove_stopwords)
df['lemmatized_text'] = df['review'].apply(lemitizeStr)
df.head()
```

```
print("Samples per class: {}".format(np.bincount(df.encoded_sentiments)))
X_train, X_test, Y_train, Y_test = train_test_split(df['lemmatized_text'], df['encoded_sentiments'], test_size
= 0.30, random_state = 7)
print('Shape of X_train', X_train.shape)
print('Shape of X_test', X_test.shape)
print('Shape of Y_train', Y_train.shape)
print('Shape of Y_test', Y_test.shape)
import spacy
def tokenizer(str):
    nlp = spacy.blank('ur')
    doc = nlp.tokenizer(str)
    return [i.text for i in doc]
df["tokens"] = df["lemmatized_text"].apply(tokenizer)
import gensim

model_word2vec = gensim.models.Word2Vec(sentences=df["tokens"], size=128, window=5, workers=10,
min_count = 1)
model_word2vec.wv.most_similar("مرد")
VOCAB_SIZE = len(model_word2vec.wv.vocab)
DIMENSIONS = 128
MAX_LEN = max([len(x) for x in df["tokens"]])
VOCAB_SIZE, DIMENSIONS, MAX_LEN
from keras.preprocessing.text import Tokenizer
token = Tokenizer()
token.fit_on_texts(df["tokens"])
encoded = token.texts_to_sequences(df["tokens"])
words2vec_matrix = np.zeros((VOCAB_SIZE+1,DIMENSIONS))
for word, index in token.word_index.items():
    try:
        words2vec_matrix[index] = model_word2vec.wv[word]
    except:
        print(index, word)
import tensorflow as tf
train_vectors = tf.keras.preprocessing.sequence.pad_sequences(encoded,padding='post',dtype=int)
train_label = df.encoded_sentiments
type(train_label[0])
(train_sentences,test_sentences, train_tags, test_tags) = train_test_split(train_vectors, train_label,
test_size=0.2, shuffle = True)
train_sentences
import tensorflow.keras.layers as Layers
import tensorflow.keras.activations as Actications
import tensorflow.keras.models as Models
from tensorflow.keras.optimizers import Adam, Optimizer, SGD
import tensorflow.keras.initializers as Init
from tensorflow.keras import regularizers
lstm = Models.Sequential()
```

```
lstm = Models.Sequential()

lstm.add(Layers.Embedding(VOCAB_SIZE+1,DIMENSIONS,
                    embeddings_initializer = Init.Constant(words2vec_matrix),
                    input_length=MAX_LEN, trainable=False ))

lstm.add(Layers.Bidirectional(Layers.LSTM(256, activation='tanh')))

lstm.add(Layers.Dense(128, activation='tanh'))
lstm.add(Layers.Dropout(0.3))

lstm.add(Layers.Dense(64, activation='tanh'))
lstm.add(Layers.Dropout(0.3))

lstm.add(Layers.Dense(1, activation='sigmoid'))

lstm.summary()

from keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import EarlyStopping


lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
es_callback = EarlyStopping(monitor='val_loss', patience=3)
LSTM_NET = lstm.fit(train_sentences, train_tags, epochs=10, validation_split=0.2,
callbacks=[es_callback], shuffle=False)

plt.plot(LSTM_NET.history['accuracy'])
plt.plot(LSTM_NET.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

plt.plot(LSTM_NET.history['loss'])
plt.plot(LSTM_NET.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

Appendix C

Experiment results for Sentiment Analysis using TF-IDF and BoW on SVM, Decision Tree and Logistic Regression

	Iteration	1					2					3					4					5				
	CV = 10	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM
TFIDF - SVM	1	0.812	0.81	0.81	0.81	[5976 1494] [1366 6164]	0.811	0.81	0.81	0.81	[5992 1478] [1360 6170]	0.813	0.81	0.81	0.81	[5990 1480] [1356 6174]	0.813	0.81	0.81	0.81	[5989 1481] [1364 6166]	0.81	0.81	0.81	0.81	[5984 1486] [1360 6170]
	2	0.804	0.81	0.81	0.81	[5976 1494] [1366 6164]	0.805	0.81	0.81	0.81	[5992 1478] [1360 6170]	0.805	0.81	0.81	0.81	[5990 1480] [1356 6174]	0.804	0.81	0.81	0.81	[5989 1481] [1364 6166]	0.805	0.81	0.81	0.81	[5984 1486] [1360 6170]
	3	0.813	0.81	0.81	0.81	[5976 1494] [1366 6164]	0.813	0.81	0.81	0.81	[5992 1478] [1360 6170]	0.813	0.81	0.81	0.81	[5990 1480] [1356 6174]	0.813	0.81	0.81	0.81	[5989 1481] [1364 6166]	0.814	0.81	0.81	0.81	[5984 1486] [1360 6170]
	4	0.813	0.81	0.81	0.81	[5976 1494] [1366 6164]	0.813	0.81	0.81	0.81	[5992 1478] [1360 6170]	0.811	0.81	0.81	0.81	[5990 1480] [1356 6174]	0.809	0.81	0.81	0.81	[5989 1481] [1364 6166]	0.809	0.81	0.81	0.81	[5984 1486] [1360 6170]
	5	0.799	0.81	0.81	0.81	[5976 1494] [1366 6164]	0.8	0.81	0.81	0.81	[5992 1478] [1360 6170]	0.802	0.81	0.81	0.81	[5990 1480] [1356 6174]	0.804	0.81	0.81	0.81	[5989 1481] [1364 6166]	0.804	0.81	0.81	0.81	[5984 1486] [1360 6170]
	6	0.807	0.81	0.81	0.81	[5976 1494] [1366 6164]	0.808	0.81	0.81	0.81	[5992 1478] [1360 6170]	0.808	0.81	0.81	0.81	[5990 1480] [1356 6174]	0.808	0.81	0.81	0.81	[5989 1481] [1364 6166]	0.808	0.81	0.81	0.81	[5984 1486] [1360 6170]
	7	0.814	0.81	0.81	0.81	[5976 1494] [1366 6164]	0.817	0.81	0.81	0.81	[5992 1478] [1360 6170]	0.818	0.81	0.81	0.81	[5990 1480] [1356 6174]	0.817	0.81	0.81	0.81	[5989 1481] [1364 6166]	0.817	0.81	0.81	0.81	[5984 1486] [1360 6170]
	8	0.804	0.81	0.81	0.81	[5976 1494] [1366 6164]	0.804	0.81	0.81	0.81	[5992 1478] [1360 6170]	0.804	0.81	0.81	0.81	[5990 1480] [1356 6174]	0.803	0.81	0.81	0.81	[5989 1481] [1364 6166]	0.803	0.81	0.81	0.81	[5984 1486] [1360 6170]
	9	0.804	0.81	0.81	0.81	[5976 1494] [1366 6164]	0.801	0.81	0.81	0.81	[5992 1478] [1360 6170]	0.801	0.81	0.81	0.81	[5990 1480] [1356 6174]	0.8	0.81	0.81	0.81	[5989 1481] [1364 6166]	0.8	0.81	0.81	0.81	[5984 1486] [1360 6170]
	10	0.804	0.81	0.81	0.81	[5976 1494] [1366 6164]	0.805	0.81	0.81	0.81	[5992 1478] [1360 6170]	0.806	0.81	0.81	0.81	[5990 1480] [1356 6174]	0.807	0.81	0.81	0.81	[5989 1481] [1364 6166]	0.807	0.81	0.81	0.81	[5984 1486] [1360 6170]
	Average	0.81	0.81	0.81	0.81	[5976 1494] [1366 6164]	0.81	0.81	0.81	0.81	[5992 1478] [1360 6170]	0.81	0.81	0.81	0.81	[5990 1480] [1356 6174]	0.81	0.81	0.81	0.81	[5989 1481] [1364 6166]	0.81	0.81	0.81	0.81	[5984 1486] [1360 6170]

	Iteration	6					7					8					9					10				
	CV = 10	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM
TFIDF - SVM	1	0.81	0.81	0.81	0.81	[5990 1480] [1360 6170]	0.811	0.81	0.81	0.81	[5994 1476] [1364 6166]	0.81	0.81	0.81	0.81	[5991 1479] [1359 6171]	0.81	0.81	0.81	0.81	[5991 1479] [1360 6170]	0.81	0.81	0.81	0.81	[5991 1479] [1350 6180]
	2	0.805	0.81	0.81	0.81	[5990 1480] [1360 6170]	0.806	0.81	0.81	0.81	[5994 1476] [1364 6166]	0.807	0.81	0.81	0.81	[5991 1479] [1359 6171]	0.807	0.81	0.81	0.81	[5991 1479] [1360 6170]	0.806	0.81	0.81	0.81	[5991 1479] [1350 6180]
	3	0.813	0.81	0.81	0.81	[5990 1480] [1360 6170]	0.813	0.81	0.81	0.81	[5994 1476] [1364 6166]	0.813	0.81	0.81	0.81	[5991 1479] [1359 6171]	0.812	0.81	0.81	0.81	[5991 1479] [1360 6170]	0.812	0.81	0.81	0.81	[5991 1479] [1350 6180]
	4	0.81	0.81	0.81	0.81	[5990 1480] [1360 6170]	0.809	0.81	0.81	0.81	[5994 1476] [1364 6166]	0.809	0.81	0.81	0.81	[5991 1479] [1359 6171]	0.808	0.81	0.81	0.81	[5991 1479] [1360 6170]	0.809	0.81	0.81	0.81	[5991 1479] [1350 6180]
	5	0.805	0.81	0.81	0.81	[5990 1480] [1360 6170]	0.804	0.81	0.81	0.81	[5994 1476] [1364 6166]	0.804	0.81	0.81	0.81	[5991 1479] [1359 6171]	0.804	0.81	0.81	0.81	[5991 1479] [1360 6170]	0.803	0.81	0.81	0.81	[5991 1479] [1350 6180]
	6	0.807	0.81	0.81	0.81	[5990 1480] [1360 6170]	0.807	0.81	0.81	0.81	[5994 1476] [1364 6166]	0.807	0.81	0.81	0.81	[5991 1479] [1359 6171]	0.807	0.81	0.81	0.81	[5991 1479] [1360 6170]	0.805	0.81	0.81	0.81	[5991 1479] [1350 6180]
	7	0.818	0.81	0.81	0.81	[5990 1480] [1360 6170]	0.818	0.81	0.81	0.81	[5994 1476] [1364 6166]	0.817	0.81	0.81	0.81	[5991 1479] [1359 6171]	0.817	0.81	0.81	0.81	[5991 1479] [1360 6170]	0.817	0.81	0.81	0.81	[5991 1479] [1350 6180]
	8	0.804	0.81	0.81	0.81	[5990 1480] [1360 6170]	0.801	0.81	0.81	0.81	[5994 1476] [1364 6166]	0.801	0.81	0.81	0.81	[5991 1479] [1359 6171]	0.801	0.81	0.81	0.81	[5991 1479] [1360 6170]	0.802	0.81	0.81	0.81	[5991 1479] [1350 6180]
	9	0.8	0.81	0.81	0.81	[5990 1480] [1360 6170]	0.8	0.81	0.81	0.81	[5994 1476] [1364 6166]	0.801	0.81	0.81	0.81	[5991 1479] [1359 6171]	0.801	0.81	0.81	0.81	[5991 1479] [1360 6170]	0.801	0.81	0.81	0.81	[5991 1479] [1350 6180]
	10	0.806	0.81	0.81	0.81	[5990 1480] [1360 6170]	0.806	0.81	0.81	0.81	[5994 1476] [1364 6166]	0.806	0.81	0.81	0.81	[5991 1479] [1359 6171]	0.806	0.81	0.81	0.81	[5991 1479] [1360 6170]	0.807	0.81	0.81	0.81	[5991 1479] [1350 6180]
	Average	0.81	0.81	0.81	0.81	[5990 1480] [1360 6170]	0.81	0.81	0.81	0.81	[5994 1476] [1364 6166]	0.81	0.81	0.81	0.81	[5991 1479] [1359 6171]	0.81	0.81	0.81	0.81	[5991 1479] [1360 6170]	0.81	0.81	0.81	0.81	[5991 1479] [1350 6180]

	Iteration	1					2					3					4					5				
	CV = 10	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM
BOW - SVM	1	0.663	0.67	0.67	0.67	[11040 6530] [5050 12380]	0.66	0.67	0.67	0.67	[11062 6508] [5068 12362]	0.662	0.67	0.67	0.67	[11066 6504] [5068 12362]	0.66	0.67	0.67	0.67	[11074 6496] [5079 12351]	0.661	0.67	0.67	0.67	[11086 6484] [5075 12355]
	2	0.656	0.67	0.67	0.67	[11040 6530] [5050 12380]	0.656	0.67	0.67	0.67	[11062 6508] [5068 12362]	0.658	0.67	0.67	0.67	[11066 6504] [5068 12362]	0.657	0.67	0.67	0.67	[11074 6496] [5079 12351]	0.659	0.67	0.67	0.67	[11086 6484] [5075 12355]
	3	0.659	0.67	0.67	0.67	[11040 6530] [5050 12380]	0.66	0.67	0.67	0.67	[11062 6508] [5068 12362]	0.662	0.67	0.67	0.67	[11066 6504] [5068 12362]	0.659	0.67	0.67	0.67	[11074 6496] [5079 12351]	0.658	0.67	0.67	0.67	[11086 6484] [5075 12355]
	4	0.64	0.67	0.67	0.67	[11040 6530] [5050 12380]	0.643	0.67	0.67	0.67	[11062 6508] [5068 12362]	0.644	0.67	0.67	0.67	[11066 6504] [5068 12362]	0.641	0.67	0.67	0.67	[11074 6496] [5079 12351]	0.64	0.67	0.67	0.67	[11086 6484] [5075 12355]
	5	0.68	0.67	0.67	0.67	[11040 6530] [5050 12380]	0.679	0.67	0.67	0.67	[11062 6508] [5068 12362]	0.678	0.67	0.67	0.67	[11066 6504] [5068 12362]	0.677	0.67	0.67	0.67	[11074 6496] [5079 12351]	0.675	0.67	0.67	0.67	[11086 6484] [5075 12355]
	6	0.672	0.67	0.67	0.67	[11040 6530] [5050 12380]	0.669	0.67	0.67	0.67	[11062 6508] [5068 12362]	0.672	0.67	0.67	0.67	[11066 6504] [5068 12362]	0.672	0.67	0.67	0.67	[11074 6496] [5079 12351]	0.671	0.67	0.67	0.67	[11086 6484] [5075 12355]
	7	0.66	0.67	0.67	0.67	[11040 6530] [5050 12380]	0.663	0.67	0.67	0.67	[11062 6508] [5068 12362]	0.666	0.67	0.67	0.67	[11066 6504] [5068 12362]	0.666	0.67	0.67	0.67	[11074 6496] [5079 12351]	0.665	0.67	0.67	0.67	[11086 6484] [5075 12355]
	8	0.67	0.67	0.67	0.67	[11040 6530] [5050 12380]	0.668	0.67	0.67	0.67	[11062 6508] [5068 12362]	0.667	0.67	0.67	0.67	[11066 6504] [5068 12362]	0.669	0.67	0.67	0.67	[11074 6496] [5079 12351]	0.67	0.67	0.67	0.67	[11086 6484] [5075 12355]
	9	0.658	0.67	0.67	0.67	[11040 6530] [5050 12380]	0.659	0.67	0.67	0.67	[11062 6508] [5068 12362]	0.66	0.67	0.67	0.67	[11066 6504] [5068 12362]	0.661	0.67	0.67	0.67	[11074 6496] [5079 12351]	0.662	0.67	0.67	0.67	[11086 6484] [5075 12355]
	10	0.687	0.67	0.67	0.67	[11040 6530] [5050 12380]	0.687	0.67	0.67	0.67	[11062 6508] [5068 12362]	0.687	0.67	0.67	0.67	[11066 6504] [5068 12362]	0.685	0.67	0.67	0.67	[11074 6496] [5079 12351]	0.685	0.67	0.67	0.67	[11086 6484] [5075 12355]
	Average	0.67	0.67	0.67	0.67	[11040 6530] [5050 12380]	0.67	0.67	0.67	0.67	[11062 6508] [5068 12362]	0.67	0.67	0.67	0.67	[11066 6504] [5068 12362]	0.67	0.67	0.67	0.67	[11074 6496] [5079 12351]	0.67	0.67	0.67	0.67	[11086 6484] [5075 12355]

	Iteration	6					7					8					9					10				
	CV = 10	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM
BOW - SVM	1	0.66	0.67	0.67	0.67	[11077 6493] [5081 12349]	0.66	0.67	0.67	0.67	[11076 6494] [5079 12351]	0.661	0.67	0.67	0.67	[11076 6494] [5073 12357]	0.659	0.67	0.67	0.67	[11070 6500] [5059 12371]	0.658	0.67	0.67	0.67	[11062 6508] [5064 12366]
	2	0.658	0.67	0.67	0.67	[11077 6493] [5081 12349]	0.658	0.67	0.67	0.67	[11076 6494] [5079 12351]	0.66	0.67	0.67	0.67	[11076 6494] [5073 12357]	0.66	0.67	0.67	0.67	[11070 6500] [5059 12371]	0.659	0.67	0.67	0.67	[11062 6508] [5064 12366]
	3	0.658	0.67	0.67	0.67	[11077 6493] [5081 12349]	0.659	0.67	0.67	0.67	[11076 6494] [5079 12351]	0.658	0.67	0.67	0.67	[11076 6494] [5073 12357]	0.659	0.67	0.67	0.67	[11070 6500] [5059 12371]	0.659	0.67	0.67	0.67	[11062 6508] [5064 12366]
	4	0.641	0.67	0.67	0.67	[11077 6493] [5081 12349]	0.641	0.67	0.67	0.67	[11076 6494] [5079 12351]	0.642	0.67	0.67	0.67	[11076 6494] [5073 12357]	0.64	0.67	0.67	0.67	[11070 6500] [5059 12371]	0.64	0.67	0.67	0.67	[11062 6508] [5064 12366]
	5	0.676	0.67	0.67	0.67	[11077 6493] [5081 12349]	0.676	0.67	0.67	0.67	[11076 6494] [5079 12351]	0.676	0.67	0.67	0.67	[11076 6494] [5073 12357]	0.677	0.67	0.67	0.67	[11070 6500] [5059 12371]	0.677	0.67	0.67	0.67	[11062 6508] [5064 12366]
	6	0.669	0.67	0.67	0.67	[11077 6493] [5081 12349]	0.667	0.67	0.67	0.67	[11076 6494] [5079 12351]	0.669	0.67	0.67	0.67	[11076 6494] [5073 12357]	0.669	0.67	0.67	0.67	[11070 6500] [5059 12371]	0.669	0.67	0.67	0.67	[11062 6508] [5064 12366]
	7	0.665	0.67	0.67	0.67	[11077 6493] [5081 12349]	0.666	0.67	0.67	0.67	[11076 6494] [5079 12351]	0.666	0.67	0.67	0.67	[11076 6494] [5073 12357]	0.665	0.67	0.67	0.67	[11070 6500] [5059 12371]	0.666	0.67	0.67	0.67	[11062 6508] [5064 12366]
	8	0.671	0.67	0.67	0.67	[11077 6493] [5081 12349]	0.671	0.67	0.67	0.67	[11076 6494] [5079 12351]	0.671	0.67	0.67	0.67	[11076 6494] [5073 12357]	0.671	0.67	0.67	0.67	[11070 6500] [5059 12371]	0.671	0.67	0.67	0.67	[11062 6508] [5064 12366]
	9	0.659	0.67	0.67	0.67	[11077 6493] [5081 12349]	0.659	0.67	0.67	0.67	[11076 6494] [5079 12351]	0.658	0.67	0.67	0.67	[11076 6494] [5073 12357]	0.659	0.67	0.67	0.67	[11070 6500] [5059 12371]	0.659	0.67	0.67	0.67	[11062 6508] [5064 12366]
	10	0.685	0.67	0.67	0.67	[11077 6493] [5081 12349]	0.683	0.67	0.67	0.67	[11076 6494] [5079 12351]	0.683	0.67	0.67	0.67	[11076 6494] [5073 12357]	0.683	0.67	0.67	0.67	[11070 6500] [5059 12371]	0.684	0.67	0.67	0.67	[11062 6508] [5064 12366]
	Average	0.67	0.67	0.67	0.67	[11077 6493] [5081 12349]	0.67	0.67	0.67	0.67	[11076 6494] [5079 12351]	0.67	0.67	0.67	0.67	[11076 6494] [5073 12357]	0.67	0.67	0.67	0.67	[11070 6500] [5059 12371]	0.67	0.67	0.67	0.67	[11062 6508] [5064 12366]

	Number of Iteration	1					Number of Iteration	2					Number of Iteration	3					Number of Iteration	4					Number of Iteration	5				
	Cross Validation - 10	A	P	R	f1-Score	CM	Cross Validation	A	P	R	f1-Score	CM	Cross Validation	A	P	R	f1-Score	CM	Cross Validation	A	P	R	f1-Score	CM	Cross Validation	A	P	R	f1-Score	CM
	1	0.64043	0.66	0.64	0.63	[3697, 3923] [1526, 5944]	1	0.65786	0.67	0.67	0.67	[4734, 2786] [2171, 5309]	1	0.66314	0.67	0.67	0.67	[5402, 2118] [2849, 4631]	1	0.69457	0.7	0.7	0.7	[5246, 2274] [2299, 5181]	1	0.68197	0.71	0.71	0.71	[5282, 2238] [2064, 5416]
	2	0.64623	0.62	0.62	0.62	[4512, 3008] [2641, 4839]	2	0.65374	0.69	0.68	0.67	[4229, 3291] [1522, 5958]	2	0.65903	0.68	0.67	0.67	[4394, 3126] [1786, 5694]	2	0.69674	0.68	0.68	0.68	[4493, 2607] [2207, 5273]	2	0.6994	0.7	0.7	0.7	[5176, 2344] [2168, 5312]
	3	0.63929	0.62	0.62	0.61	[5525, 1995] [3759, 3721]	3	0.64757	0.65	0.65	0.65	[4690, 2830] [2267, 5112]	3	0.65791	0.64	0.64	0.64	[4489, 3031] [2222, 5156]	3	0.69417	0.71	0.71	0.71	[5245, 2272] [2099, 5311]	3	0.69254	0.7	0.7	0.7	[5126, 2394] [2061, 5419]
	4	0.63857	0.66	0.65	0.65	[5028, 1692] [3525, 3955]	4	0.65557	0.7	0.7	0.7	[5112, 2408] [2164, 5326]	4	0.66114	0.62	0.62	0.61	[3914, 3606] [2121, 5359]	4	0.69731	0.7	0.7	0.7	[5335, 2185] [2250, 5220]	4	0.69063	0.72	0.72	0.72	[5268, 2252] [1991, 5489]
TFIDF - Decision Tree	5	0.6328	0.66	0.66	0.66	[4790, 2730] [2372, 5108]	5	0.6524	0.66	0.66	0.66	[5217, 2303] [2760, 4720]	5	0.66549	0.65	0.65	0.65	[5361, 2159] [3067, 4413]	5	0.69371	0.69	0.69	0.69	[5091, 2429] [2211, 5269]	5	0.69434	0.7	0.7	0.7	[5102, 2418] [2119, 5361]
	6	0.63137	0.66	0.65	0.65	[5502, 2018] [3161, 4319]	6	0.65086	0.65	0.65	0.65	[4526, 2994] [2239, 5241]	6	0.66363	0.68	0.68	0.67	[5229, 2291] [2582, 4898]	6	0.69391	0.71	0.71	0.71	[5274, 2246] [2141, 5339]	6	0.6942	0.71	0.71	0.71	[5096, 2424] [1961, 5499]
	7	0.63826	0.66	0.65	0.64	[3914, 3606] [1640, 5840]	7	0.64974	0.66	0.66	0.65	[5704, 1816] [3317, 4163]	7	0.66191	0.66	0.66	0.66	[4574, 2946] [2158, 5322]	7	0.69309	0.71	0.71	0.71	[5329, 2191] [2190, 5290]	7	0.69463	0.7	0.7	0.7	[5211, 2309] [2126, 5344]
	8	0.63069	0.67	0.67	0.67	[5524, 1996] [3000, 4480]	8	0.6556	0.66	0.65	0.65	[4049, 3471] [1713, 5767]	8	0.66509	0.67	0.67	0.67	[5299, 2221] [2471, 4809]	8	0.69809	0.69	0.69	0.69	[5230, 2290] [2312, 5168]	8	0.6942	0.7	0.7	0.7	[5171, 2349] [2142, 5337]
	9	0.63757	0.67	0.66	0.65	[5931, 1589] [3561, 2919]	9	0.64646	0.65	0.64	0.64	[4164, 3356] [1990, 5490]	9	0.66157	0.64	0.64	0.64	[5040, 2472] [2922, 4558]	9	0.69414	0.69	0.69	0.69	[5306, 2214] [2450, 5030]	9	0.69646	0.69	0.69	0.69	[4988, 2532] [2062, 5418]
	10	0.63826	0.63	0.62	0.61	[3700, 3812] [1094, 5586]	10	0.64989	0.65	0.65	0.64	[4100, 3412] [1860, 5612]	10	0.66757	0.65	0.65	0.65	[4927, 2593] [2468, 4812]	10	0.70097	0.7	0.7	0.7	[5102, 2417] [2072, 5408]	10	0.69709	0.72	0.72	0.72	[5329, 2191] [2023, 5447]
	Average	0.63735	0.65	0.64	0.64	[4512, 3008] [2641, 4839]	Average	0.65197	0.66	0.66	0.66	[4734, 2786] [2171, 5309]	Average	0.66201	0.656	0.655	0.653	[4927, 2593] [2468, 4812]	Average	0.69567	0.698	0.698	0.698	[5102, 2417] [2072, 5408]	Average	0.69445	0.705	0.71	0.705	[5176, 2344] [2168, 5312]

	Number of Iteration	6					Number of Iteration	7					Number of Iteration	8					Number of Iteration	4					Number of Iteration	5				
	Cross Validation	A	P	R	f1-Score	CM	Cross Validation	A	P	R	f1-Score	CM	Cross Validation	A	P	R	f1-Score	CM	Cross Validation	A	P	R	f1-Score	CM	Cross Validation	A	P	R	f1-Score	CM
	1	0.70249	0.7	0.7	0.7	[5020, 2492] [1990, 5490]	1	0.69954	0.7	0.7	0.7	[5176, 2344] [2187, 5293]	1	0.69537	0.7	0.7	0.7	[5184, 2336] [2181, 5299]	1	0.69066	0.69	0.69	0.69	[4885, 2635] [2069, 5411]	1	0.69011	0.7	0.7	0.7	[5424, 2096] [2361, 5119]
	2	0.70103	0.68	0.68	0.68	[5202, 2316] [2417, 5062]	2	0.69743	0.7	0.7	0.7	[5067, 2453] [2089, 5391]	2	0.69251	0.71	0.71	0.71	[5143, 2377] [2021, 5491]	2	0.70117	0.7	0.7	0.7	[5335, 2185] [2265, 5215]	2	0.69689	0.72	0.72	0.72	[5142, 2378] [1916, 5564]
	3	0.70246	0.7	0.7	0.7	[5040, 2472] [2028, 5462]	3	0.69263	0.7	0.7	0.7	[5163, 2357] [2089, 5391]	3	0.69631	0.71	0.71	0.71	[5070, 2450] [1978, 5602]	3	0.69889	0.7	0.7	0.7	[5127, 2393] [2127, 5403]	3	0.69206	0.71	0.71	0.71	[4934, 2586] [1820, 5560]
	4	0.70249	0.7	0.7	0.7	[5304, 2216] [2241, 5239]	4	0.69346	0.68	0.68	0.68	[4932, 2588] [2182, 5298]	4	0.69426	0.68	0.68	0.68	[4963, 2557] [2192, 5288]	4	0.69703	0.72	0.72	0.72	[5275, 2245] [1972, 5507]	4	0.6952	0.7	0.7	0.7	[5147, 2373] [2187, 5293]
TFIDF - Decision Tree	5	0.6962	0.69	0.69	0.69	[5054, 2464] [2172, 5207]	5	0.69537	0.7	0.7	0.7	[5115, 2405] [2094, 5286]	5	0.698	0.69	0.69	0.69	[5193, 2327] [2240, 5140]	5	0.69363	0.68	0.68	0.68	[4896, 2624] [2173, 5201]	5	0.69991	0.69	0.69	0.69	[5442, 2078] [2516, 4964]
	6	0.69989	0.7	0.7	0.7	[5206, 2314] [2252, 5228]	6	0.69431	0.72	0.72	0.72	[5033, 2167] [2067, 5412]	6	0.696	0.68	0.68	0.68	[5070, 2450] [2237, 5142]	6	0.69549	0.68	0.68	0.68	[4965, 2615] [2169, 5311]	6	0.69636	0.7	0.7	0.7	[5237, 2323] [2179, 5301]
	7	0.70034	0.7	0.7	0.7	[5250, 2270] [2223, 5257]	7	0.69491	0.71	0.71	0.71	[5224, 2294] [2121, 5359]	7	0.69449	0.7	0.7	0.7	[5357, 2163] [2367, 5112]	7	0.69669	0.71	0.71	0.71	[5243, 2237] [2167, 5313]	7	0.69706	0.69	0.69	0.69	[5220, 2300] [2302, 5178]
	8	0.69377	0.69	0.69	0.69	[5299, 2221] [2426, 5044]	8	0.69706	0.7	0.7	0.7	[5051, 2469] [2088, 5392]	8	0.69421	0.71	0.71	0.71	[5160, 2360] [2045, 5435]	8	0.69811	0.7	0.7	0.7	[5135, 2385] [2045, 5431]	8	0.69414	0.68	0.68	0.68	[4757, 2763] [2001, 5479]
	9	0.69026	0.69	0.69	0.69	[5146, 2374] [2220, 5240]	9	0.69511	0.71	0.71	0.71	[5242, 2178] [2106, 5374]	9	0.69529	0.7	0.7	0.7	[5164, 2356] [2140, 5340]	9	0.69751	0.71	0.71	0.71	[5231, 2289] [2044, 5436]	9	0.69706	0.71	0.71	0.71	[5206, 2314] [2072, 5408]
	10	0.69954	0.7	0.7	0.7	[5176, 2344] [2187, 5293]	10	0.69963	0.7	0.7	0.7	[5307, 2213] [2275, 5205]	10	0.69314	0.69	0.69	0.69	[5156, 2364] [2286, 5194]	10	0.69811	0.7	0.7	0.7	[5424, 2096] [2361, 5119]	10	0.69571	0.71	0.71	0.71	[5308, 2212] [2121, 5349]
	Average	0.69893	0.7	0.7	0.7	[5304, 2216] [2241, 5239]	Average	0.69595	0.7	0.7	0.7	[5307, 2213] [2275, 5205]	Average	0.69517	0.697	0.697	0.697	[5193, 2327] [2240, 5140]	Average	0.69673	0.699	0.699	0.699	[5335, 2185] [2265, 5215]	Average	0.6964	0.701	0.7	0.701	[5442, 2078] [2516, 4964]

	Number of Iteration	1					2					3					4					5				
	Cross Validation = 10	A	P	R	F1-Score	CM	A	P	R	F1-Score	CM	A	P	R	F1-Score	CM	A	P	R	F1-Score	CM	A	P	R	F1-Score	CM
	1	0.5571	0.57	0.57	0.57	[4070, 3419] [3061, 4450]	0.563	0.56	0.56	0.56	[4166, 3323] [3318, 4193]	0.5715	0.57	0.57	0.57	[4318, 3171] [3329, 4182]	0.5731	0.58	0.58	0.58	[4379, 3110] [3260, 4251]	0.5778	0.58	0.58	0.58	[4278, 3211] [3050, 4461]
	2	0.5558	0.54	0.54	0.54	[3343, 4146] [2748, 4763]	0.5661	0.56	0.56	0.56	[4290, 3199] [3433, 4078]	0.5671	0.56	0.56	0.56	[3865, 3624] [2945, 4566]	0.5793	0.58	0.58	0.57	[3838, 3651] [2713, 4798]	0.5806	0.57	0.57	0.57	[4316, 3173] [3257, 4254]
	3	0.5626	0.56	0.56	0.56	[4395, 3094] [3537, 3974]	0.5689	0.57	0.57	0.56	[3922, 3567] [2955, 4556]	0.5674	0.57	0.57	0.57	[4223, 3266] [3199, 4312]	0.5817	0.58	0.58	0.58	[4283, 3206] [3142, 4369]	0.5766	0.57	0.57	0.57	[4220, 3269] [3150, 4361]
	4	0.5562	0.56	0.56	0.56	[4360, 2909] [3742, 3769]	0.5582	0.56	0.56	0.56	[3993, 3498] [3113, 4398]	0.5647	0.55	0.55	0.55	[3563, 3926] [2820, 4691]	0.5734	0.57	0.57	0.57	[3992, 3497] [2947, 4564]	0.5765	0.56	0.56	0.56	[4292, 3197] [3356, 4155]
	5	0.5553	0.55	0.55	0.54	[4736, 2753] [4035, 3476]	0.5669	0.56	0.56	0.56	[4125, 3364] [3186, 4325]	0.5655	0.56	0.56	0.56	[4209, 3280] [3297, 4214]	0.5802	0.58	0.58	0.58	[4432, 3057] [3205, 4306]	0.5779	0.57	0.57	0.57	[4220, 3269] [3240, 4271]
BOW - Decision Tree	6	0.5601	0.55	0.55	0.55	[3966, 3523] [3213, 4298]	0.5638	0.56	0.56	0.56	[4390, 3099] [3536, 3975]	0.5638	0.56	0.55	0.55	[4625, 2864] [3813, 3698]	0.5765	0.58	0.58	0.58	[4256, 3233] [3139, 4372]	0.5763	0.57	0.57	0.57	[4302, 3187] [3281, 4230]
	7	0.5595	0.58	0.58	0.58	[3935, 3554] [2764, 4747]	0.5665	0.58	0.58	0.58	[4403, 3086] [3229, 4282]	0.5626	0.56	0.56	0.56	[4150, 3339] [3194, 4317]	0.5771	0.58	0.58	0.58	[4439, 3050] [3244, 4267]	0.5803	0.57	0.57	0.57	[4017, 3472] [2971, 4540]
	8	0.563	0.57	0.57	0.57	[4635, 2854] [3572, 3939]	0.5629	0.56	0.56	0.56	[4173, 3316] [3336, 4175]	0.5622	0.56	0.56	0.56	[4456, 3033] [3558, 3953]	0.5726	0.58	0.58	0.58	[4324, 3165] [3148, 4363]	0.5826	0.57	0.57	0.57	[4371, 3118] [3292, 4219]
	9	0.5604	0.56	0.56	0.55	[4707, 2782] [3865, 3646]	0.5663	0.58	0.58	0.57	[3743, 3746] [2611, 4900]	0.5645	0.56	0.56	0.56	[4031, 3458] [3170, 4341]	0.5779	0.58	0.58	0.58	[4394, 3095] [3241, 4270]	0.5794	0.58	0.58	0.58	[4356, 3133] [3162, 4349]
	10	0.5594	0.55	0.55	0.55	[4487, 3002] [3796, 3715]	0.5631	0.56	0.56	0.55	[3651, 3838] [2811, 4700]	0.5625	0.58	0.58	0.58	[4105, 3384] [2859, 4652]	0.5787	0.57	0.57	0.57	[4333, 3156] [3344, 4167]	0.5765	0.57	0.57	0.57	[4175, 3314] [3102, 4409]
	Average	0.5589	0.559	0.559	0.557	[4635, 2854] [3572, 3939]	0.5646	0.565	0.565	0.562	[3922, 3567] [2955, 4556]	0.5652	0.563	0.562	0.562	[4318, 3171] [3329, 4182]	0.577	0.578	0.578	0.577	[4283, 3206] [3142, 4369]	0.5784	0.571	0.571	0.571	[4371, 3118] [3292, 4219]

	Number of Iteration	6					7					8					9					10				
	Cross Validation = 10	A	P	R	F1-Score	CM	A	P	R	F1-Score	CM	A	P	R	F1-Score	CM	A	P	R	F1-Score	CM	A	P	R	F1-Score	CM
	1	0.5796	0.57	0.57	0.57	[4308, 3181] [3333, 4178]	0.5814	0.58	0.58	0.58	[4382, 3107] [3248, 4263]	0.578	0.58	0.58	0.58	[4343, 3146] [3144, 4367]	0.5799	0.58	0.58	0.58	[4361, 3128] [3191, 4320]	0.5827	0.56	0.56	0.56	[4285, 3204] [3363, 4148]
	2	0.5765	0.57	0.57	0.57	[4334, 3155] [3290, 4221]	0.5805	0.58	0.58	0.58	[4313, 3176] [3128, 4383]	0.5834	0.59	0.59	0.59	[4282, 3207] [2930, 4581]	0.5765	0.57	0.57	0.57	[4423, 3066] [3373, 4138]	0.5778	0.57	0.57	0.57	[4252, 3237] [3276, 4235]
	3	0.5781	0.57	0.57	0.57	[4044, 3445] [2939, 4572]	0.5776	0.57	0.57	0.57	[4382, 3107] [3359, 4152]	0.5802	0.57	0.57	0.57	[4538, 2951] [3430, 4081]	0.5796	0.57	0.57	0.57	[4085, 3404] [3034, 4477]	0.5809	0.57	0.57	0.57	[4275, 3214] [3300, 4211]
	4	0.5774	0.57	0.57	0.57	[4360, 3129] [3367, 4144]	0.5826	0.6	0.6	0.6	[4212, 3277] [2757, 4754]	0.5773	0.59	0.59	0.59	[4387, 3102] [3079, 4432]	0.5789	0.57	0.57	0.57	[4291, 3198] [3311, 4200]	0.5796	0.58	0.58	0.58	[4380, 3109] [3227, 4284]
	5	0.5787	0.59	0.59	0.59	[4444, 3045] [3087, 4424]	0.5794	0.57	0.57	0.57	[4504, 2985] [3485, 4026]	0.5759	0.57	0.57	0.57	[4223, 3266] [3226, 4285]	0.5826	0.58	0.58	0.58	[4496, 2993] [3274, 4237]	0.5794	0.57	0.56	0.56	[3762, 3727] [2802, 4709]
	6	0.5761	0.58	0.58	0.58	[4188, 3301] [3007, 4504]	0.5807	0.6	0.6	0.6	[4410, 3079] [2935, 4576]	0.5801	0.6	0.6	0.6	[4421, 3068] [2985, 4526]	0.5805	0.57	0.57	0.57	[4302, 3187] [3227, 4284]	0.5782	0.57	0.57	0.57	[4358, 3131] [3347, 4164]
	7	0.5757	0.59	0.59	0.59	[4235, 3254] [2886, 4625]	0.5779	0.57	0.57	0.57	[4480, 3009] [3371, 4140]	0.5798	0.58	0.58	0.58	[4379, 3110] [3256, 4253]	0.5787	0.59	0.59	0.59	[4235, 3254] [2951, 4560]	0.5789	0.58	0.58	0.58	[4072, 3417] [2944, 4567]
	8	0.579	0.57	0.57	0.57	[4243, 3246] [3278, 4233]	0.5778	0.57	0.57	0.57	[4145, 3344] [3145, 4366]	0.5791	0.58	0.58	0.58	[4428, 3061] [3200, 4311]	0.5791	0.58	0.58	0.58	[4474, 3015] [3255, 4256]	0.5796	0.57	0.57	0.57	[4360, 3129] [3294, 4217]
	9	0.5808	0.57	0.57	0.57	[4366, 3123] [3382, 4129]	0.5791	0.58	0.58	0.58	[4345, 3144] [3150, 4361]	0.5746	0.58	0.58	0.58	[4489, 3000] [3333, 4178]	0.5793	0.57	0.57	0.57	[4244, 3245] [3204, 4307]	0.5835	0.59	0.59	0.59	[4344, 3145] [3075, 4436]
	10	0.5802	0.57	0.57	0.57	[4107, 3382] [3069, 4442]	0.5806	0.59	0.59	0.59	[4308, 3181] [3025, 4486]	0.5792	0.57	0.57	0.57	[3917, 3572] [2860, 4651]	0.5913	0.59	0.59	0.59	[4391, 3098] [3033, 4478]	0.5808	0.57	0.57	0.57	[4155, 3334] [3131, 4380]
	Average	0.5782	0.575	0.575	0.575	[4366, 3123] [3382, 4129]	0.5798	0.581	0.581	0.581	[4212, 3277] [2757, 4754]	0.5788	0.581	0.581	0.581	[4282, 3207] [2930, 4581]	0.5806	0.577	0.577	0.577	[4496, 2993] [3274, 4237]	0.5801	0.573	0.572	0.572	[4344, 3145] [3075, 4436]

	Iteration	1					2					3					4					5				
	CV = 10	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM
TFIDF - Logistic Regression	1	0.4994	0.25	0.5	0.33	[7520, 0] [7480, 0]	0.6893	0.74	0.73	0.73	[4858, 2662] [1386, 6094]	0.7997	0.81	0.81	0.81	[5981, 1539] [1344,	0.8071	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8064	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	2	0.4994	0.25	0.5	0.33	[7520, 0] [7480, 0]	0.7161	0.74	0.73	0.73	[4858, 2662] [1386, 6094]	0.8013	0.81	0.81	0.81	[5981, 1539]	0.8071	0.81	0.81	0.81	[6015, 1505] [1309, 6171]	0.8067	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	3	0.4994	0.25	0.5	0.33	[7520, 0] [7480, 0]	0.7241	0.74	0.73	0.73	[4858, 2662] [1386, 6094]	0.8019	0.81	0.81	0.81	[5981, 1539]	0.807	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8067	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	4	0.4994	0.25	0.5	0.33	[7520, 0] [7480, 0]	0.7247	0.74	0.73	0.73	[4858, 2662] [1386, 6094]	0.8025	0.81	0.81	0.81	[5981, 1539]	0.8069	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8065	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	5	0.4994	0.25	0.5	0.33	[7520, 0] [7480, 0]	0.7251	0.74	0.73	0.73	[4858, 2662] [1386, 6094]	0.8029	0.81	0.81	0.81	[5981, 1539]	0.8065	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8061	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	6	0.4994	0.25	0.5	0.33	[7520, 0] [7480, 0]	0.7253	0.74	0.73	0.73	[4858, 2662] [1386, 6094]	0.8032	0.81	0.81	0.81	[5981, 1539]	0.8072	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8066	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	7	0.4994	0.25	0.5	0.33	[7520, 0] [7480, 0]	0.7256	0.74	0.73	0.73	[4858, 2662] [1386, 6094]	0.8027	0.81	0.81	0.81	[5981, 1539]	0.807	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8069	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	8	0.4994	0.25	0.5	0.33	[7520, 0] [7480, 0]	0.7258	0.74	0.73	0.73	[4858, 2662] [1386, 6094]	0.8029	0.81	0.81	0.81	[5981, 1539]	0.8075	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8069	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	9	0.4994	0.25	0.5	0.33	[7520, 0] [7480, 0]	0.726	0.74	0.73	0.73	[4858, 2662] [1386, 6094]	0.803	0.81	0.81	0.81	[5981, 1539]	0.8073	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8069	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	10	0.4994	0.25	0.5	0.33	[7520, 0] [7480, 0]	0.7259	0.74	0.73	0.73	[4858, 2662] [1386, 6094]	0.8028	0.81	0.81	0.81	[5981, 1539]	0.8071	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8067	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	Average	0.4994	0.25	0.5	0.33	[7520, 0] [7480, 0]	0.7208	0.74	0.73	0.73	[4858, 2662] [1386, 6094]	0.8023	0.81	0.81	0.81	[5981, 1539]	0.8071	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8067	0.81	0.81	0.81	[6021, 1499] [1321, 6159]

	Iteration	6					7					8					9					10				
	CV = 10	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM
TFIDF - Logistic Regression	1	0.8064	0.81	0.81	0.81	[6021, 1499] [1321, 6159]	0.8071	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8064	0.81	0.81	0.81	[5981, 1539]	0.8071	0.81	0.81	0.81	[4858, 2662] [1386, 6094]	0.8064	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	2	0.8067	0.81	0.81	0.81	[6021, 1499] [1321, 6159]	0.8071	0.81	0.81	0.81	[6015, 1505] [1309, 6171]	0.8067	0.81	0.81	0.81	[5981, 1539]	0.8071	0.81	0.81	0.81	[4858, 2662] [1386, 6094]	0.8067	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	3	0.8067	0.81	0.81	0.81	[6021, 1499] [1321, 6159]	0.807	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8067	0.81	0.81	0.81	[5981, 1539]	0.807	0.81	0.81	0.81	[4858, 2662] [1386, 6094]	0.8067	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	4	0.8065	0.81	0.81	0.81	[6021, 1499] [1321, 6159]	0.8069	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8065	0.81	0.81	0.81	[5981, 1539]	0.8069	0.81	0.81	0.81	[4858, 2662] [1386, 6094]	0.8065	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	5	0.8061	0.81	0.81	0.81	[6021, 1499] [1321, 6159]	0.8065	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8061	0.81	0.81	0.81	[5981, 1539]	0.8065	0.81	0.81	0.81	[4858, 2662] [1386, 6094]	0.8061	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	6	0.8066	0.81	0.81	0.81	[6021, 1499] [1321, 6159]	0.8072	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8066	0.81	0.81	0.81	[5981, 1539]	0.8072	0.81	0.81	0.81	[4858, 2662] [1386, 6094]	0.8066	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	7	0.8069	0.81	0.81	0.81	[6021, 1499] [1321, 6159]	0.807	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8069	0.81	0.81	0.81	[5981, 1539]	0.807	0.81	0.81	0.81	[4858, 2662] [1386, 6094]	0.8069	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	8	0.8069	0.81	0.81	0.81	[6021, 1499] [1321, 6159]	0.8075	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8069	0.81	0.81	0.81	[5981, 1539]	0.8075	0.81	0.81	0.81	[4858, 2662] [1386, 6094]	0.8069	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	9	0.8069	0.81	0.81	0.81	[6021, 1499] [1321, 6159]	0.8073	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8069	0.81	0.81	0.81	[5981, 1539]	0.8073	0.81	0.81	0.81	[4858, 2662] [1386, 6094]	0.8069	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	10	0.8067	0.81	0.81	0.81	[6021, 1499] [1321, 6159]	0.8071	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8067	0.81	0.81	0.81	[5981, 1539]	0.8071	0.81	0.81	0.81	[4858, 2662] [1386, 6094]	0.8067	0.81	0.81	0.81	[6021, 1499] [1321, 6159]
	Average	0.8069	0.81	0.81	0.81	[6021, 1499] [1321, 6159]	0.8076	0.81	0.81	0.81	[6015, 1505] [1310, 6170]	0.8022	0.81	0.81	0.81	[5981, 1539]	0.8071	0.81	0.81	0.81	[4858, 2662] [1386, 6094]	0.8066	0.81	0.81	0.81	[6021, 1499] [1321, 6159]

	Iteration	1					2					3					4					5				
	CV = 10	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM
BOW - Logistic Regression	1	0.6523	0.66	0.66	0.66	[4924, 2565] [2528, 4983]	0.6652	0.67	0.67	0.67	[4951, 2538] [2411, 5100]	0.6683	0.67	0.67	0.67	[4910, 2579] [2332, 5179]	0.6692	0.67	0.67	0.67	[4913, 2576] [2330, 5181]	0.6681	0.67	0.67	0.67	[4914, 2575] [2326, 5185]
	2	0.655	0.66	0.66	0.66	[4924, 2565] [2528, 4983]	0.6664	0.67	0.67	0.67	[4951, 2538] [2411, 5100]	0.6674	0.67	0.67	0.67	[4910, 2579] [2332, 5179]	0.6681	0.67	0.67	0.67	[4913, 2576] [2330, 5181]	0.6677	0.67	0.67	0.67	[4914, 2575] [2326, 5185]
	3	0.6556	0.66	0.66	0.66	[4924, 2565] [2528, 4983]	0.6662	0.67	0.67	0.67	[4951, 2538] [2411, 5100]	0.6675	0.67	0.67	0.67	[4910, 2579] [2332, 5179]	0.6676	0.67	0.67	0.67	[4913, 2576] [2330, 5181]	0.6675	0.67	0.67	0.67	[4914, 2575] [2326, 5185]
	4	0.6556	0.66	0.66	0.66	[4924, 2565] [2528, 4983]	0.6665	0.67	0.67	0.67	[4951, 2538] [2411, 5100]	0.6677	0.67	0.67	0.67	[4910, 2579] [2332, 5179]	0.6684	0.67	0.67	0.67	[4913, 2576] [2330, 5181]	0.6678	0.67	0.67	0.67	[4914, 2575] [2326, 5185]
	5	0.656	0.66	0.66	0.66	[4924, 2565] [2528, 4983]	0.6666	0.67	0.67	0.67	[4951, 2538] [2411, 5100]	0.6683	0.67	0.67	0.67	[4910, 2579] [2332, 5179]	0.6686	0.67	0.67	0.67	[4913, 2576] [2330, 5181]	0.6682	0.67	0.67	0.67	[4914, 2575] [2326, 5185]
	6	0.656	0.66	0.66	0.66	[4924, 2565] [2528, 4983]	0.6667	0.67	0.67	0.67	[4951, 2538] [2411, 5100]	0.6687	0.67	0.67	0.67	[4910, 2579] [2332, 5179]	0.6681	0.67	0.67	0.67	[4913, 2576] [2330, 5181]	0.6734	0.67	0.67	0.67	[4914, 2575] [2326, 5185]
	7	0.6557	0.66	0.66	0.66	[4924, 2565] [2528, 4983]	0.6668	0.67	0.67	0.67	[4951, 2538] [2411, 5100]	0.6685	0.67	0.67	0.67	[4910, 2579] [2332, 5179]	0.6686	0.67	0.67	0.67	[4913, 2576] [2330, 5181]	0.6677	0.67	0.67	0.67	[4914, 2575] [2326, 5185]
	8	0.6564	0.66	0.66	0.66	[4924, 2565] [2528, 4983]	0.6669	0.67	0.67	0.67	[4951, 2538] [2411, 5100]	0.6686	0.67	0.67	0.67	[4910, 2579] [2333, 5178]	0.6682	0.67	0.67	0.67	[4913, 2576] [2330, 5181]	0.6679	0.67	0.67	0.67	[4914, 2575] [2326, 5185]
	9	0.6561	0.66	0.66	0.66	[4924, 2565] [2528, 4983]	0.6667	0.67	0.67	0.67	[4951, 2538] [2411, 5100]	0.6691	0.67	0.67	0.67	[4910, 2579] [2332, 5179]	0.668	0.67	0.67	0.67	[4913, 2576] [2330, 5181]	0.6679	0.67	0.67	0.67	[4914, 2575] [2326, 5185]
	10	0.6563	0.66	0.66	0.66	[4924, 2565] [2528, 4983]	0.667	0.67	0.67	0.67	[4951, 2538] [2411, 5100]	0.6691	0.67	0.67	0.67	[4910, 2579] [2332, 5179]	0.669	0.67	0.67	0.67	[4913, 2576] [2330, 5181]	0.6683	0.67	0.67	0.67	[4914, 2575] [2326, 5185]
	Average	0.6555	0.66	0.66	0.66	[4924, 2565] [2528, 4983]	0.6665	0.67	0.67	0.67	[4951, 2538] [2411, 5100]	0.6683	0.67	0.67	0.67	[4910, 2579] [2332, 5179]	0.6684	0.67	0.67	0.67	[4913, 2576] [2330, 5181]	6.6844	6.7	6.7	6.7	[4914, 2575] [2326, 5185]

	Iteration	1					2					3					4					5				
	CV = 10	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM	A	P	R	F1	CM
BOW - Logistic Regression	1	0.6652	0.67	0.67	0.67	[4926, 2563] [2548, 4984]	0.6692	0.67	0.67	0.67	[4955, 2539] [2413, 5350]	0.6683	0.67	0.67	0.67	[4915, 2560] [2322, 5192]	0.6681	0.67	0.67	0.67	[4918, 2584] [2358, 5185]	0.6623	0.67	0.67	0.67	[4918, 2585] [2329, 5188]
	2	0.6664	0.67	0.67	0.67	[4926, 2563] [2548, 4984]	0.6681	0.67	0.67	0.67	[4955, 2539] [2413, 5350]	0.6674	0.67	0.67	0.67	[4915, 2560] [2322, 5192]	0.6677	0.67	0.67	0.67	[4918, 2584] [2358, 5185]	0.665	0.67	0.67	0.67	[4918, 2585] [2329, 5188]
	3	0.6662	0.67	0.67	0.67	[4926, 2563] [2548, 4984]	0.6676	0.67	0.67	0.67	[4955, 2539] [2413, 5350]	0.6675	0.67	0.67	0.67	[4915, 2560] [2322, 5192]	0.6675	0.67	0.67	0.67	[4918, 2584] [2358, 5185]	0.6656	0.67	0.67	0.67	[4918, 2585] [2329, 5188]
	4	0.6665	0.67	0.67	0.67	[4926, 2563] [2548, 4984]	0.6684	0.67	0.67	0.67	[4955, 2539] [2413, 5350]	0.6677	0.67	0.67	0.67	[4915, 2560] [2322, 5192]	0.6678	0.67	0.67	0.67	[4918, 2584] [2358, 5185]	0.6656	0.67	0.67	0.67	[4918, 2585] [2329, 5188]
	5	0.6666	0.67	0.67	0.67	[4926, 2563] [2548, 4984]	0.6686	0.67	0.67	0.67	[4955, 2539] [2413, 5350]	0.6683	0.67	0.67	0.67	[4915, 2560] [2322, 5192]	0.6682	0.67	0.67	0.67	[4918, 2584] [2358, 5185]	0.666	0.67	0.67	0.67	[4918, 2585] [2329, 5188]
	6	0.6667	0.67	0.67	0.67	[4926, 2563] [2548, 4984]	0.6681	0.67	0.67	0.67	[4955, 2539] [2413, 5350]	0.6687	0.67	0.67	0.67	[4915, 2560] [2322, 5192]	0.6734	0.67	0.67	0.67	[4918, 2584] [2358, 5185]	0.664	0.67	0.67	0.67	[4918, 2585] [2329, 5188]
	7	0.6668	0.67	0.67	0.67	[4926, 2563] [2548, 4984]	0.6686	0.67	0.67	0.67	[4955, 2539] [2413, 5350]	0.6685	0.67	0.67	0.67	[4915, 2560] [2322, 5192]	0.6677	0.67	0.67	0.67	[4918, 2584] [2358, 5185]	0.6657	0.67	0.67	0.67	[4918, 2585] [2329, 5188]
	8	0.6669	0.67	0.67	0.67	[4926, 2563] [2548, 4984]	0.6682	0.67	0.67	0.67	[4955, 2539] [2413, 5350]	0.6686	0.67	0.67	0.67	[4915, 2560] [2322, 5192]	0.6679	0.67	0.67	0.67	[4918, 2584] [2358, 5185]	0.6664	0.67	0.67	0.67	[4918, 2585] [2329, 5188]
	9	0.6667	0.67	0.67	0.67	[4926, 2563] [2548, 4984]	0.668	0.67	0.67	0.67	[4955, 2539] [2413, 5350]	0.6691	0.67	0.67	0.67	[4915, 2560] [2322, 5192]	0.6679	0.67	0.67	0.67	[4918, 2584] [2358, 5185]	0.6631	0.67	0.67	0.67	[4918, 2585] [2329, 5188]
	10	0.667	0.67	0.67	0.67	[4926, 2563] [2548, 4984]	0.669	0.67	0.67	0.67	[4955, 2539] [2413, 5350]	0.6691	0.67	0.67	0.67	[4915, 2560] [2322, 5192]	0.6683	0.67	0.67	0.67	[4918, 2584] [2358, 5185]	0.6613	0.67	0.67	0.67	[4918, 2585] [2329, 5188]
	Average	0.6665	0.67	0.67	0.67	[4926, 2563] [2548, 4984]	0.6684	0.67	0.67	0.67	[4955, 2539] [2413, 5350]	0.6683	0.67	0.67	0.67	[4915, 2560] [2322, 5192]	0.6684	0.67	0.67	0.67	[4918, 2584] [2358, 5185]	0.6645	0.67	0.67	0.67	[4918, 2585] [2329, 5188]

Appendix D

External libraries used during the implementation phase

- NLTK
- UrduHack
- Stanza
- Pandas
- Seaborn
- NumPy
- Sklearn
- Tenserflow,Keras
- Genism

Appendix E

Project Timeline/Gantt Chart