# Urdu Text Sentiment Analysis Using Recurrent Neural Networks

Assignment 3

Syed Usman Ali Shah
*386543*
*BSCS11-A*

Ali Awais Safdar
*369475*
*BSCS11-A*

***Abstract*— This paper presents a sentiment analysis approach applied to Urdu movie reviews using bidirectional Gated Recurrent Unit (GRU) architectures. The dataset was acquired from Kaggle, comprising 50k movie reviews translated into Urdu. Preprocessing involved stop words removal, normalization, lemmatization, and tokenization using UrduHack and Spacy libraries. We experimented with various neural network architectures, including unidirectional and bidirectional LSTMs and GRUs, exploring different configurations for optimal performance. Our findings indicate that bidirectional GRU models outperform other architectures in terms of accuracy, F1 score, and training efficiency.**

## I. Introduction

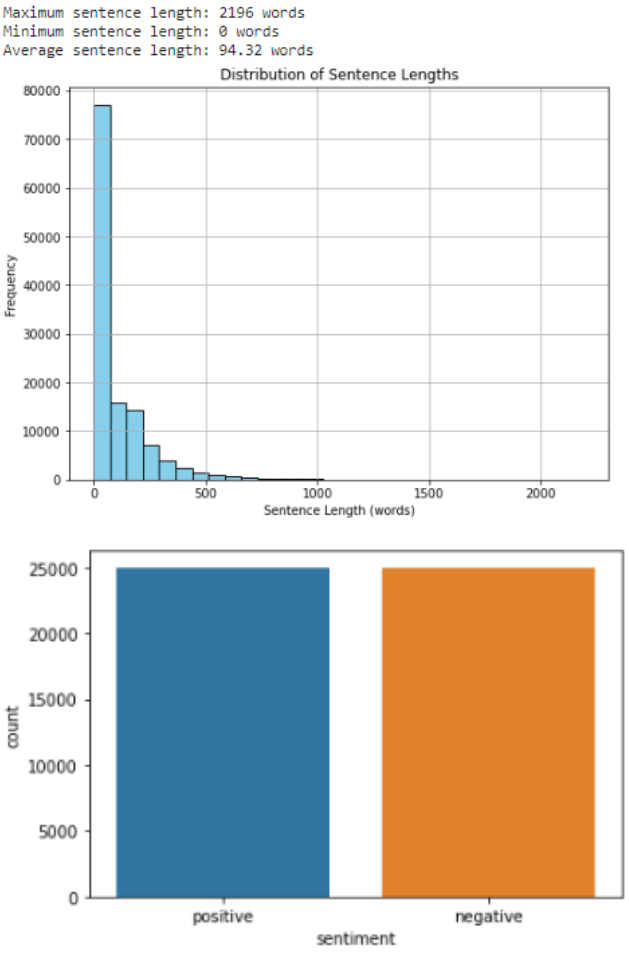
Sentiment analysis, a subfield of natural language processing (NLP), aims to identify and classify sentiments expressed in text data. With the increasing availability of digital content in multiple languages, sentiment analysis in non-English languages has gained importance. Urdu, being a widely spoken language, presents challenges and opportunities for sentiment analysis tasks. In this work, we focus on sentiment analysis of Urdu movie reviews, leveraging deep learning techniques.

## II. DATASET ACQUISITION

The dataset used in our study was sourced from Kaggle, a popular platform for data science and machine learning datasets. Specifically, we accessed the "IMDB Dataset of 50k Movie Translated Urdu Reviews" [1]. This dataset is valuable for sentiment analysis tasks as it contains a large collection of movie reviews in Urdu, a language widely spoken in Pakistan and other regions.

The dataset comprises 50,000 rows, each representing a movie review. These reviews are labelled with sentiment polarity, indicating whether the review expresses positive or negative sentiment. The dataset is evenly balanced, with 50% of the reviews labelled as positive sentiment and the other 50% as negative sentiment. This balanced distribution is essential for training machine learning models effectively, ensuring that the model learns from a diverse set of examples representing different sentiments.

By leveraging this dataset, we were able to create a robust framework for sentiment analysis on Urdu text. The richness of the dataset, combined with its balanced sentiment labels, provided us with a solid foundation for training and evaluating our sentiment analysis models accurately.



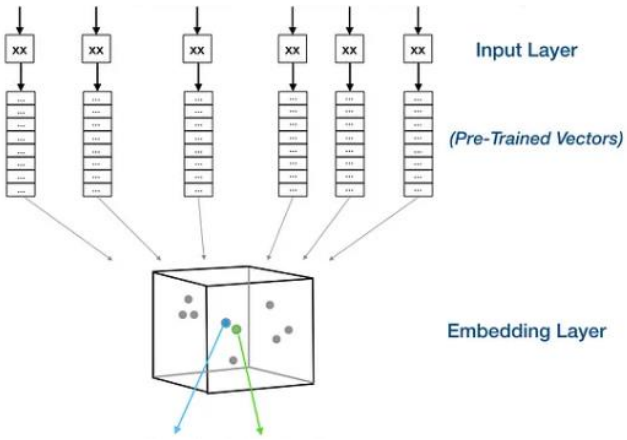

## III. Preprocessing

Preprocessing is a crucial step in natural language processing (NLP) tasks as it helps clean and transform raw text data into a format suitable for machine learning models. In our study, we utilized the UrduHack library, which is specifically designed for processing Urdu text, for the following preprocessing tasks:

1. **Stop Words Removal:** Stop words are common words in a language (like "تھے " "اپ " "آئے " etc.) that often do not carry significant meaning for analysis. By removing stop words, we reduce noise in the data and focus on

more meaningful content.

2. **Normalization:** Normalization involves standardizing text by converting it to a consistent format. This may include converting numbers to words, replacing abbreviations with their full forms, and handling special characters or symbols.
3. **Lemmatization:** Lemmatization is the process of reducing words to their base or root form, known as the lemma. This helps in reducing the dimensionality of the data by grouping together different forms of the same word.
4. **Tokenization using Spacy:** Tokenization is the process of splitting text into smaller units, typically words or tokens. We used the Spacy library for tokenization, which provides robust support for various languages, including Urdu. This step is essential for breaking down text into manageable units for analysis.
5. **Word2Vec Embeddings:** Word2Vec is a popular technique for generating word embeddings, which are dense vector representations of words in a continuous vector space. These embeddings capture semantic relationships between words and are crucial for training deep learning models like neural networks. We generated word2vec embeddings to represent words in our text data, enabling the model to learn semantic similarities and patterns during training.




By performing these preprocessing steps, we ensure that the input data for our sentiment analysis model is clean, standardized, and structured in a way that facilitates effective learning and pattern recognition. This preprocessing pipeline lays the foundation for building accurate and robust sentiment analysis models on Urdu text.

## IV. MODEL CREATION

**Embedding Layer:**

The model starts with an embedding layer, which is responsible for converting the input text data (word indices) into dense vectors of fixed dimensions. These vectors represent semantic meanings of words in a continuous vector space.

The embedding layer helps the model understand relationships and similarities between words based on their context within the dataset.

**Bidirectional GRU Layer:**

Following the embedding layer is a bidirectional GRU (Gated Recurrent Unit) layer. GRU is a type of recurrent neural network (RNN) that can capture long-term dependencies in sequential data.

The bidirectional aspect means that this GRU layer processes input sequences in both forward and backward directions. This bidirectional processing helps the model capture context from both past and future time steps, improving its understanding of the text's context.

The GRU layer has 256 units (neurons) and uses a hyperbolic tangent (tanh) activation function, which introduces non-linearity into the model's computations.

**Dense Layers with Dropout:**

After the bidirectional GRU layer, the model includes dense layers. Dense layers are fully connected layers where each neuron is connected to every neuron in the previous and subsequent layers.

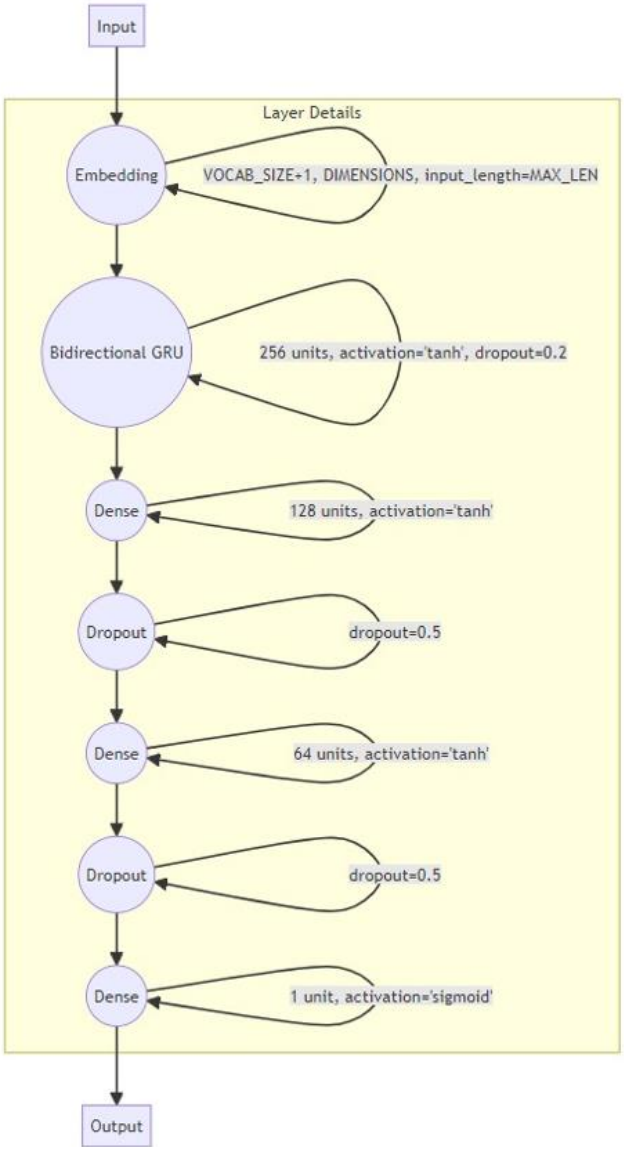
The first dense layer has 128 units with a hyperbolic tangent (tanh) activation function. This layer helps the model learn complex patterns and features from the output of the GRU layer.

Dropout layers are inserted after each dense layer with a dropout rate of 0.5 (50%). Dropout is a regularization technique that randomly "drops out" a fraction of units during training, preventing overfitting and improving model generalization.

**Output Layer:**

The final layer in the architecture is the output layer. It consists of a single neuron with a sigmoid activation function.

The sigmoid activation function outputs a value between 0 and 1, representing the predicted probability that the input text belongs to the positive sentiment class (1) or negative sentiment class (0). A threshold can be applied to this probability to make binary sentiment predictions.




| Layer (type) | Output Shape | Param # |
|---|---|---|
| embedding_10 (Embedding) | (None, 1398, 128) | 13039616 |
| bidirectional_8 (Bidirection | (None, 512) | 592896 |
| dense_30 (Dense) | (None, 128) | 65664 |
| dropout_20 (Dropout) | (None, 128) | 0 |
| dense_31 (Dense) | (None, 64) | 8256 |
| dropout_21 (Dropout) | (None, 64) | 0 |
| dense_32 (Dense) | (None, 1) | 65 |

Total params: 13,706,497
Trainable params: 666,881
Non-trainable params: 13,039,616

## V. Experimentation

The experimentation phase of our study involved systematically testing and evaluating different neural network architectures to determine the most effective model for sentiment analysis on Urdu movie reviews. We explored the following architectures:

1. **Unidirectional LSTM (Long Short-Term Memory):** LSTM networks are a type of recurrent neural network (RNN) designed to capture long-term dependencies in sequential data. In unidirectional LSTMs, information flows only in one direction, making them suitable for tasks where past information is crucial for prediction.
2. **Unidirectional GRU (Gated Recurrent Unit):** GRU networks are like LSTMs but have a simpler architecture with fewer parameters. They also address the vanishing gradient problem and are faster to train compared to LSTMs. Unidirectional GRUs were included in our experimentation to compare their performance with LSTMs.
3. **Bidirectional LSTM:** Bidirectional LSTMs incorporate information from both past and future time steps by processing sequences in both forward and backward directions. This allows the model to capture context from both directions, potentially improving performance in tasks that require understanding context from the entire sequence.
4. **Bidirectional GRU:** Like bidirectional LSTMs, bidirectional GRUs process information from both directions. GRUs have fewer parameters than LSTMs and can be computationally more efficient while still capturing long-term dependencies.

During experimentation, we varied several parameters to optimize model performance:

- **Layer Sizes:** We tested different sizes (512, 256, 128, 64 etc.) for the LSTM and GRU layers to find the optimal balance between model complexity and learning capacity. Larger layers can capture more complex patterns but may also increase training time and risk overfitting.
- **Dropout Rates:** Dropout is a regularization technique used to prevent overfitting by randomly dropping neurons during training. We experimented with different dropout rates to control model complexity and generalization. We tried using 0.3 and 0.5 dropout rates and then settling on the rate that gave the best performance according to the various performance metrics.
- **Activation Functions:** Activation functions play a crucial role in neural network architectures by introducing non-linearity. We tested different activation functions such as tanh, sigmoid, and relu to assess their impact on model performance.

By systematically varying these parameters and evaluating the performance metrics (such as accuracy, F1 score, and training efficiency) for each architecture, we aimed to identify the most effective model configuration for sentiment analysis on Urdu movie reviews. This rigorous experimentation process ensures that our final model is well-tuned and optimized for the task at hand.

## VI. Results and Analysis

$$\text{Precision} = \frac{TP}{TP + FP},$$

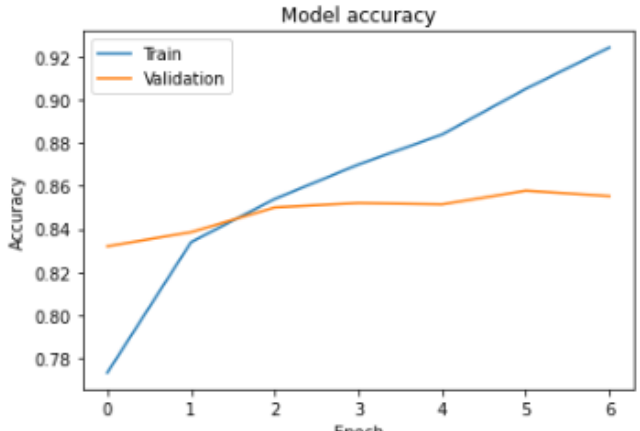
$$\text{Recall} = \frac{TP}{TP + FN},$$

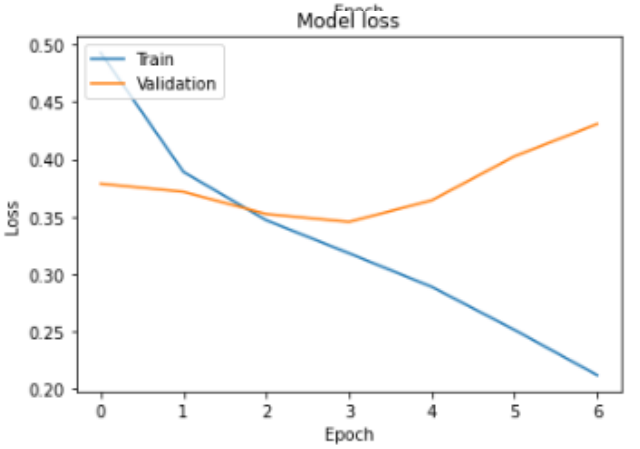
$$\text{F}_1 = \frac{2 \times P \times R}{P + R},$$

- In our study, we evaluated various neural network architectures for sentiment analysis on Urdu movie reviews. Among these architectures, the bidirectional GRU (Gated Recurrent Unit) model emerged as the top performer, demonstrating superior performance compared to other architectures. Here are the key findings from our analysis:
- **Superior Performance:** The bidirectional GRU model consistently outperformed other architectures in terms of multiple performance metrics, including validation accuracy, precision, recall and F1 score. These metrics are crucial for evaluating the model's ability to correctly classify sentiments in the reviews.
- **High Accuracy:** The bidirectional GRU model achieved high accuracy, 90% training accuracy and 85% validation accuracy. Bidirectional LSTM had 92% training accuracy, but the validation accuracy was lower, 84%.
- **F1 Score:** The F1 score is a metric that balances precision and recall, providing a comprehensive evaluation of the model's performance, especially in binary classification tasks like sentiment analysis. The bidirectional GRU model achieved a higher F1 score of 0.86 indicating its ability to achieve a good balance between identifying positive and negative sentiments while minimizing misclassifications. Compared to the bidirectional LSTM which had an F1 score of 0.85.
- **Efficient Training:** In addition to its superior performance, the bidirectional GRU model exhibited efficient training compared to both unidirectional models (such as unidirectional LSTM and unidirectional GRU) and bidirectional LSTM. Bidirectional GRU was able to achieve the best results whilst training for 20 minutes. On the other hand, bidirectional LSTM gave nearly similar results but had a higher training time of around 26 minutes.
- **Effectiveness for Urdu Text:** The success of the bidirectional GRU model underscores its effectiveness specifically for sentiment analysis on Urdu text. Urdu, being a complex language with unique linguistic characteristics, benefits from architectures like bidirectional GRU that can capture contextual dependencies effectively from both past and future sequences.

Overall, our results highlight the effectiveness of bidirectional GRU architectures for sentiment analysis on Urdu movie reviews. The combination of high accuracy, F1 score, and efficient training makes the bidirectional GRU model a promising choice for sentiment analysis tasks in languages like Urdu, where capturing nuanced sentiment nuances is essential. Below are the results of

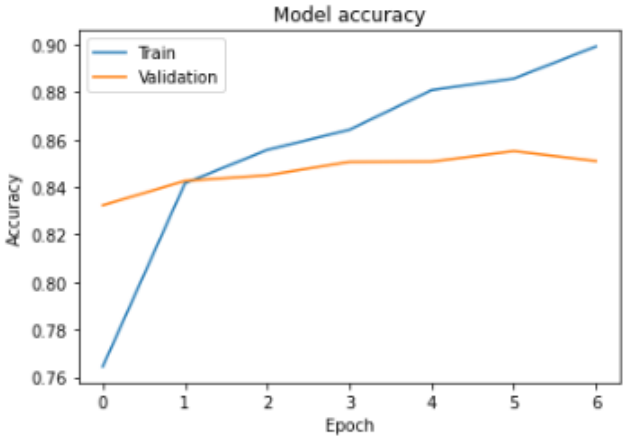
### 1. Bidirectional LSTM

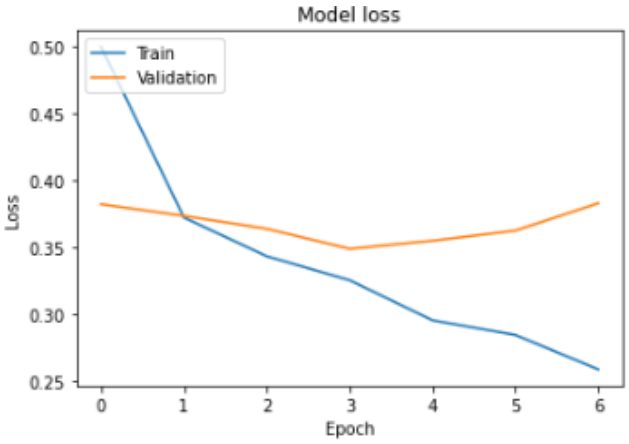



| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0.0 | 0.80 | 0.89 | 0.84 | 4564 |
| 1.0 | 0.90 | 0.82 | 0.85 | 5436 |
| accuracy | | | 0.85 | 10000 |
| macro avg | 0.85 | 0.85 | 0.85 | 10000 |
| weighted avg | 0.85 | 0.85 | 0.85 | 10000 |

### 2. Bidirectional GRU






| | precision | recall | f1-score | suppor |
|---|---|---|---|---|
| 0.0 | 0.85 | 0.86 | 0.86 | 479 |
| 1.0 | 0.87 | 0.86 | 0.87 | 520 |
| accuracy | | | 0.86 | 1000 |
| macro avg | 0.86 | 0.86 | 0.86 | 1000 |
| weighted avg | 0.86 | 0.86 | 0.86 | 1000 |

## VII. LIMITATIONS

While the "IMDB Dataset of 50k Movie Translated Urdu Reviews" from Kaggle provides a substantial amount of data, there are a few considerations that might impact the results and analysis:

1. **Translation Accuracy:** The dataset comprises translated Urdu movie reviews. However, the accuracy of the translations could vary, leading to potential discrepancies or loss of nuanced sentiment expressions from the original Urdu text. Inaccurate translations may introduce noise or bias into the sentiment analysis results.
2. **Limited Domain Coverage:** The dataset focuses specifically on movie reviews, which may limit the model's ability to generalize to other domains or topics. Sentiment analysis models trained primarily on movie-related content may not perform as effectively when applied to diverse text genres or industries.
3. **Bias in Labeling:** The sentiment labels (positive and negative) assigned to the reviews may reflect subjective judgments and individual annotator biases. This could introduce inconsistencies or misinterpretations in the labeled data, affecting the model's training and evaluation.
4. **Data Imbalance:** Although the dataset is described as having an equal number of positive and negative sentiment reviews, there could be instances of data imbalance within subcategories or specific themes. Imbalanced data distributions may impact the model's ability to generalize across sentiment classes and could result in biased predictions.
5. **Limited Sentiment Nuances:** Sentiment analysis typically categorizes text into positive, negative, or neutral sentiments. However, human emotions and opinions often exhibit nuanced variations that may not be fully captured by binary sentiment classification. The model's ability to detect and interpret subtle sentiment nuances may be limited by the dataset's granularity.

## REFERENCES

[1] Kaggle. IMDB Dataset of 50k Movie Translated Urdu Reviews. Available online: https://www.kaggle.com/datasets/akkefa/imdb-dataset-of-50k-movie-translated-urdu-reviews

## SOURCE CODE

```
[2] import numpy as np # linear algebra
[3] import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
[4]
[5] # Input data files are available in the read-only "../input/" directory
[6] # For example, running this (by clicking run or pressing Shift+Enter) will list all
    files under the input directory
[7]
[8] import os
[9] for dirname, _, filenames in os.walk('/kaggle/input'):
[10]     for filename in filenames:
[11]         print(os.path.join(dirname, filename))
[12]
[13]# You can write up to 20GB to the current directory (/kaggle/working/) that gets
    preserved as output when you create a version using "Save & Run All"
[14]# You can also write temporary files to /kaggle/temp/, but they won't be saved outside
    of the current session
[15]

[16]# ## 1. Install  Required Libraries
[17]
[18]get_ipython().system('pip install urduhack')
[19]

[20]# ## 2. Import Libraries
[21]
[22]import pandas as pd
[23]import numpy as np
[24]
[25]# Import Plotting Libararies
[26]import seaborn as sns
[27]import matplotlib.pyplot as plt
[28]
[29]# Import Data Preprocessing Libraries
[30]from sklearn.preprocessing import LabelEncoder
[31]from sklearn.model_selection import train_test_split
[32]from sklearn.feature_extraction.text import TfidfVectorizer
[33]
[34]# Machine Learning Models
[35]from sklearn import svm
[36]from sklearn.naive_bayes import GaussianNB
[37]from sklearn.linear_model import LogisticRegression
[38]from sklearn.tree import DecisionTreeClassifier
[39]from sklearn.ensemble import RandomForestClassifier
[40]from sklearn.ensemble import GradientBoostingClassifier
[41]from sklearn.ensemble import AdaBoostClassifier
[42]import xgboost as xgb
[43]
[44]# Model Evaluation Libraries
[45]from sklearn.metrics import classification_report, confusion_matrix
[46]
```

```
[47]# ### 2.1 Urduhack
[48]#
[49]# Urduhack is a NLP library for urdu language. It comes with a lot of battery included features to help you process Urdu data in the easiest way possible.
[50]#
[51]# https://docs.urduhack.com/en/stable/
[52]

[53]import urduhack
[54]urduhack.download()
[55]from urduhack.normalization import normalize
[56]from urduhack.preprocessing import normalize_whitespace, remove_punctuation, remove_accents, replace_urls, replace_emails, replace_numbers, replace_currency_symbols, remove_english_alphabets
[57]

[58]# ## 3. Load Dataset
[59]# Dataset is available in 2 sets:
[60]# * Training
[61]# * Testing
[62]
[63]train_data = pd.read_csv("/kaggle/input/imdb-dataset-of-50k-movie-translated-urdu-reviews/imdb_urdu_reviews_train.csv")
[64]test_data = pd.read_csv("/kaggle/input/imdb-dataset-of-50k-movie-translated-urdu-reviews/imdb_urdu_reviews_test.csv")
[65]
[66]train_data.head(), test_data.head()
[67]

[68]# Combine Both Files to Preprocess
[69]data = pd.concat([train_data, test_data]).reset_index(drop=True)
[70]print(data.shape)
[71]

[72]# Make Copy of dataset so we dont have to load again and again
[73]
[74]# Make copy of a dataset
[75]df = data.copy()
[76]df
[77]

[78]# We have now 50,000 records available in our dataset. The size of dataset is good and we can build very good predictive model using this data.
[79]
[80]# Lets see the distribution of label column which is sentiment.
[81]

[82]sns.countplot( x = 'sentiment', data = df );
[83]

[84]# We can see that there are only two classes in our dataset:
[85]#
```

```
# * Positive means the review holds positive sentiment.
# * Negative means the review holds negative sentiment.
#
# Also the class is very balanced. So, it will be easy for us to build any model.

# ## 4. Data Preprocessing

# ### 4.1 Label Encoding of Target Variable
#
# Encode the target label sentiment.

# Encode the labels
le = LabelEncoder()
le.fit(df['sentiment'])
 df['encoded_sentiments'] = le.transform(df['sentiment'])


 # ### 4.2 Apply urduhack preprocessing
 # Now we will apply text cleaning modules from Urdu Hack Library

 df['review'] = df['review'].apply(normalize) # To normalize some text, all you need to do pass unicode text. It will return a str with normalized characters both single and combined, proper spaces after digits and punctuations and diacritics(Zabar - Paish) removed.
 df['review'] = df['review'].apply(remove_punctuation) # Remove punctuation from text by removing all instances of marks. marks=',;:'
 df['review'] = df['review'].apply(remove_accents) # Remove accents from any accented unicode characters in text str, either by transforming them into ascii equivalents or removing them entirely.
 df['review'] = df['review'].apply(replace_urls) # Replace all URLs in text str with replace_with str.
 df['review'] = df['review'].apply(replace_emails) # Replace all emails in text str with replace_with str.
 df['review'] = df['review'].apply(replace_numbers) # Replace all numbers in text str with replace_with str.
 df['review'] = df['review'].apply(replace_currency_symbols) # Replace all currency symbols in text str with string specified by replace_with str.
 df['review'] = df['review'].apply(remove_english_alphabets) # Removes English words and digits from a text
 df['review'] = df['review'].apply(normalize_whitespace) ## Given text str, replace one or more spacings with a single space, and one or more linebreaks with a single newline. Also strip leading/trailing whitespace.


 # Using publically available set of Urdu Text Stopwords we will remove stop words from our text.

 # Remove stop words from text
 from typing import FrozenSet

 # Urdu Language Stop words list
```

```
[121]  STOP_WORDS: FrozenSet[str] = frozenset("""
[122]  آ آئی آئیں آئے آتا آتی آتے آس آمدید آنا آنسہ آنی آنے آپ آگے آہ آہا آیا اب ابھی
       ابے
[123]  ارے اس اسکا اسکی اسکے اسی اسے اف افوہ البتہ الف ان اندر انکا انکی انکے انہوں
       انہی انہیں اوئے اور اوپر
[124]  اوہو اپ اپنا اپنوں اپنی اپنے اپنےآپ اکثر اگر اگرچہ اہاہا ایسا ایسی ایسے ایک
       بائیں بار بارے بالکل باوجود باہر
[125]  بج بجے بخیر بشرطیکہ بعد بعض بغیر بلکہ بن بنا بناؤ بند بڑی بھر بھریں بھی بہت
       بہتر تاکہ تاہم تب تجھ
[126]  تجھی تجھے ترا تری تلک تم تمام تمہارا تمہاروں تمہاری تمہارے تمہیں تو تک تھا
       تھی تھیں تھے تیرا تیری تیرے
[127]  جا جاؤ جائیں جائے جاتا جاتی جاتے جانی جانے جب جبکہ جدھر جس جسے جن جناب جنہوں
       جنہیں جو جہاں جی جیسا
[128]  جیسوں جیسی جیسے حالانکہ حالاں حصہ حضرت خاطر خالی خواہ خوب خود دائیں درمیان دریں
       دو دوران دوسرا دوسروں دوسری دوں
[129]  دکھائیں دی دیئے دیا دیتا دیتی دیتے دیر دینا دینی دینے دیکھو دیں دیے دے ذریعے
       رکھا رکھتا رکھتی رکھتے رکھنا رکھنی
[130]  رکھنے رکھو رکھی رکھے رہ رہا رہتا رہتی رہتے رہنا رہنی رہنے رہو رہی رہیں رہے
       ساتھ سامنے ساڑھے سب سبھی
[131]  سراسر سمیت سوا سوائے سکا سکتا سکتے سہ سہی سی سے شاید شکریہ صاحب صاحبہ صرف ضرور
       طرح طرف طور علاوہ عین
[132]  فقط فلاں فی قبل قطا لئے لائی لائے لاتا لاتی لاتے لانا لانی لانے لایا لو لوجی لوگوں لگ
       لگا لگتا
[133]  لگتی لگی لگیں لگے لہذا لی لیا لیتا لیتی لیتے لیکن لیں لیے لے ماسوا مت مجھ مجھی
       مجھے محترم محترمہ محض
[134]  مرا مرحبا مری مرے مزید مس مسز مسٹر مطابق مل مکرمی مگر مگھر مہربانی میرا میروں
       میری میرے میں نا نزدیک
[135]  نما نہ نہیں نیز نیچے نے و وار واسطے واقعی والا والوں والی والے واہ وجہ ورنہ
       وغیرہ ولے وگرنہ وہ وہاں
[136]  وہی وہیں ویسا ویسے ویں پاس پایا پر پس پلیز پون پونی پونے پھر پہ پہلا پہلی پہلے
       پیر پیچھے چاہئے
[137]  چاہتے چاہیئے چاہے چلا چلو چلیں چلے چناچہ چند چونکہ چکی چکیں چکے ڈالنا ڈالنی
       ڈالنے ڈالے کئے کا کاش کب کبھی
[138]  کدھر کر کرتا کرتی کرتے کرم کرنا کرنے کرو کریں کرے کس کسی کسے کم کن کنہیں کو
       کوئی کون کونسا
[139]  کونسے کچھ کہ کہا کہاں کہہ کہی کہیں کہے کی کیا کیسا کیسے کیونکر کیونکہ کیوں
       کیے کے گئی گئے گا گنا
[140]  گو گویا گی گیا ہائیں ہائے ہاں ہر ہرچند ہرگز ہم ہمارا ہماری ہمارے ہمی ہمیں ہو
       ہوئی ہوئیں ہوئے ہوا
[141]  ہوبہو ہوتا ہوتی ہوتیں ہوتے ہونا ہونگے ہونی ہونے ہوں ہی ہیلو ہیں ہے یا یات یعنی
       یک یہ یہاں یہی یہیں
[142]  """.split())
[143]

[144]  def remove_stopwords(text: str):
[145]      return " ".join(word for word in text.split() if word not in STOP_WORDS)
[146]



[147]  from urduhack.models.lemmatizer import lemmatizer
[148]  def lemitizeStr(str):
```

```
[149]       lemme_str = ""
[150]       temp = lemmatizer.lemma_lookup(str)
[151]       for t in temp:
[152]           lemme_str += t[0] + " "
[153]
[154]       return lemme_str
[155]

[156]  df['review'] =  df['review'].apply(remove_stopwords)
[157]


[158]  df['lemmatized_text'] = df['review'].apply(lemitizeStr)
[159]

[160]  df.head()
[161]

[162]  # Now we have cleansed text in lemmatized_text and encoded version of sentiment column
       as encoded_sentiments.
[163]  #
[164]  # Data is prepared for the Modeling.
[165]
[166]  # ### 4.3 Train Test Split
[167]

[168]  X_train,   X_test,   Y_train,   Y_test   =   train_test_split(df['lemmatized_text'],
       df['encoded_sentiments'], test_size = 0.30, random_state = 7, shuffle = True)
[169]


[170]  print('Shape of X_train', X_train.shape)
[171]  print('Shape of X_test', X_test.shape)
[172]  print('Shape of Y_train', Y_train.shape)
[173]  print('Shape of Y_test', Y_test.shape)
[174]

[175]  # ### 4.4 TF - IDF Vectorization
[176]  #
[177]  # TF-IDF is a statistical measure that evaluates how relevant a word is to a document
       in a collection of documents. This is done by multiplying two metrics: how many times a
       word appears in a document, and the inverse document frequency of the word across a set
       of documents.
[178]
[179]  max_feature_num = 50000
[180]  vectorizer = TfidfVectorizer(max_features=max_feature_num)
[181]  train_vecs = vectorizer.fit_transform(X_train)
[182]
[183]  test_vecs               =               TfidfVectorizer(max_features=max_feature_num,
       vocabulary=vectorizer.vocabulary_).fit_transform(X_test)
[184]
```

```
[185] # check the dimensions of feature vectors
[186] train_vecs.shape, test_vecs.shape
[187]

[188] #  ## 5. Sentiment Analysis using Word to Vector and Deep Learning
[189]

[190] # Make copy of Dataset to prepare for Word2Vector
[191] df_w2v = df.copy()
[192]


[193] df_w2v.head()
[194]

[195] # ### 5.1 Tokenize the Text using Spacy Urdu Tokenizer
[196]

[197] import spacy
[198] def tokenizer(str):
[199]     nlp = spacy.blank('ur')
[200]     doc = nlp.tokenizer(str)
[201]     return [i.text for i in doc]
[202] df_w2v["tokens"] = df_w2v["lemmatized_text"].apply(tokenizer)
[203]

[204] # We have used spacy tokenizer for urdu text. Urduhack library also provides tokenizer
  but its bit slow so we used this.
[205]
[206] # ### 5.2 Word to Vector Model
[207] # Word2Vec consists of models for generating word embedding. These models are shallow
  two layer neural networks having one input layer, one hidden layer and one output layer.
[208]

[209] import gensim
[210]
[211] model_word2vec    =    gensim.models.Word2Vec(sentences=df_w2v["tokens"],    size=128,
  window=5, workers=10, min_count = 1)
[212]

[213] # Lets Check the quality of embeddings generated.
[214]

[215] model_word2vec.wv.most_similar("مرد")
[216]


[217] model_word2vec.wv.most_similar("عورت")
[218]

[219] model_word2vec.wv.most_similar("خوفناک")
[220]
```

```
[221] model_word2vec.wv.most_similar("فلم")
[222]

[223] model_word2vec.wv.most_similar("کارٹون")
[224]

[225] # ### 5.3 Embedding Layer Preparation
[226]
[227] VOCAB_SIZE = len(model_word2vec.wv.vocab)
[228] DIMENSIONS = 128
[229] MAX_LEN = max([len(x) for x in df_w2v["tokens"]])
[230]


[231] VOCAB_SIZE, DIMENSIONS, MAX_LEN
[232]

[233] # * The VOCAB_SIZE shows the size of vocabulary.
[234] # * We set the size of dimension to 128 to reduce the dimensions of data.
[235] # * The MAX_LEN represents the Maximum length of Sentence in dataset.
[236]

[237] from keras.preprocessing.text import Tokenizer
[238] token = Tokenizer()
[239] token.fit_on_texts(df_w2v["tokens"])
[240] encoded = token.texts_to_sequences(df_w2v["tokens"])
[241]

[242] # We used Keras Tokenizer to tokenize the data and made text sequences.
[243]

[244] words2vec_matrix = np.zeros((VOCAB_SIZE+1,DIMENSIONS))
[245] for word, index in token.word_index.items():
[246]     try:
[247]         words2vec_matrix[index] = model_word2vec.wv[word]
[248]     except:
[249]         print(index, word)
[250]


[251] import tensorflow as tf
[252] train_vectors                                                                    =
  tf.keras.preprocessing.sequence.pad_sequences(encoded,padding='post',dtype=int)
[253]


[254] train_label = df_w2v.encoded_sentiments
[255]

[256] type(train_label[0])
[257]

[258] # Split the dataset into train and test data.
```

```
[259]
[260] (train_sentences,test_sentences,          train_tags,          test_tags)          =
  train_test_split(train_vectors, train_label, test_size=0.2, shuffle = True)
[261]


[262] train_sentences
[263]

[264] # # 6. RNN Models
[265]
[266] # ## 6.1 Long Short Term Memory
[267] # As LSTM models works good on text data, as they considered the sequences of inputs
  to make predictions. We will try LSTM Model on this data.
[268]

[269] import tensorflow.keras.layers as Layers
[270] import tensorflow.keras.activations as Actications
[271] import tensorflow.keras.models as Models
[272] from tensorflow.keras.optimizers import Adam, Optimizer, SGD
[273] import tensorflow.keras.initializers as Init
[274] from tensorflow.keras import regularizers
[275] from keras.models import Sequential
[276] from keras.layers import Embedding, LSTM, Dense, Dropout
[277] from keras.callbacks import EarlyStopping
[278] import matplotlib.pyplot as plt
[279]

[280] #  ### Uni-Directional LSTM:
[281]

[282] lstm_uni = Sequential()
[283] lstm_uni.add(Embedding(VOCAB_SIZE+1, DIMENSIONS,
[284]                       embeddings_initializer=Init.Constant(words2vec_matrix),
[285]                       input_length=MAX_LEN, trainable=False))
[286] lstm_uni.add(LSTM(256, activation='tanh'))
[287] lstm_uni.add(Dense(128, activation='tanh'))
[288] lstm_uni.add(Dropout(0.5))
[289] lstm_uni.add(Dense(64, activation='tanh'))
[290] lstm_uni.add(Dropout(0.5))
[291] lstm_uni.add(Dense(1, activation='sigmoid'))
[292]
[293] lstm_uni.summary()
[294]


[295] lstm_uni.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
[296] es_callback = EarlyStopping(monitor='val_loss', patience=3)
[297] LSTM_NET_uni     =     lstm_uni.fit(train_sentences,     train_tags,     epochs=10,
  validation_split=0.2, callbacks=[es_callback], shuffle=False)
[298]
```

```
[299] plt.plot(LSTM_NET_uni.history['accuracy'])
[300] plt.plot(LSTM_NET_uni.history['val_accuracy'])
[301] plt.title('Model accuracy')
[302] plt.ylabel('Accuracy')
[303] plt.xlabel('Epoch')
[304] plt.legend(['Train', 'Validation'], loc='upper left')
[305] plt.show()
[306]

[307] plt.plot(LSTM_NET_uni.history['loss'])
[308] plt.plot(LSTM_NET_uni.history['val_loss'])
[309] plt.title('Model loss')
[310] plt.ylabel('Loss')
[311] plt.xlabel('Epoch')
[312] plt.legend(['Train', 'Validation'], loc='upper left')
[313] plt.show()
[314]


[315] print(classification_report(lstm_uni.predict(test_sentences).round(), test_tags))
[316]

[317] # ### Bi-Directional LSTM
[318]

[319] lstm = Models.Sequential()
[320]
[321] lstm.add(Layers.Embedding(VOCAB_SIZE+1,DIMENSIONS,
[322]                          embeddings_initializer = Init.Constant(words2vec_matrix),
[323]                          input_length=MAX_LEN, trainable=False ))
[324]
[325] lstm.add(Layers.Bidirectional(Layers.LSTM(256, activation='tanh')))
[326]
[327] lstm.add(Layers.Dense(128, activation='tanh'))
[328] lstm.add(Layers.Dropout(0.5))
[329]
[330] lstm.add(Layers.Dense(64, activation='tanh'))
[331] lstm.add(Layers.Dropout(0.5))
[332]
[333] lstm.add(Layers.Dense(1, activation='sigmoid'))
[334]
[335] lstm.summary()
[336]


[337] from keras.callbacks import ModelCheckpoint
[338] from tensorflow.keras.callbacks import EarlyStopping
[339]
[340]
[341] lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
[342] es_callback = EarlyStopping(monitor='val_loss', patience=3)
```

```
[343] LSTM_NET = lstm.fit(train_sentences, train_tags, epochs=10, validation_split=0.2,
  callbacks=[es_callback], shuffle=False)
[344]


[345] plt.plot(LSTM_NET.history['accuracy'])
[346] plt.plot(LSTM_NET.history['val_accuracy'])
[347] plt.title('Model accuracy')
[348] plt.ylabel('Accuracy')
[349] plt.xlabel('Epoch')
[350] plt.legend(['Train', 'Validation'], loc='upper left')
[351] plt.show()
[352]


[353] plt.plot(LSTM_NET.history['loss'])
[354] plt.plot(LSTM_NET.history['val_loss'])
[355] plt.title('Model loss')
[356] plt.ylabel('Loss')
[357] plt.xlabel('Epoch')
[358] plt.legend(['Train', 'Validation'], loc='upper left')
[359] plt.show()
[360]


[361] print(classification_report(lstm.predict(test_sentences).round(), test_tags))
[362]

[363] # ## 6.2 GRUs
[364]
[365] # ### Uni-Directional GRU
[366]

[367] from keras.layers import Embedding, GRU, Dense, Dropout
[368]
[369] gru_uni = Sequential()
[370] gru_uni.add(Embedding(VOCAB_SIZE+1, DIMENSIONS,
[371]                       embeddings_initializer=Init.Constant(words2vec_matrix),
[372]                       input_length=MAX_LEN, trainable=False))
[373] gru_uni.add(GRU(256, activation='tanh'))
[374] gru_uni.add(Dense(128, activation='tanh'))
[375] gru_uni.add(Dropout(0.5))
[376] gru_uni.add(Dense(64, activation='tanh'))
[377] gru_uni.add(Dropout(0.5))
[378] gru_uni.add(Dense(1, activation='sigmoid'))
[379]
[380] gru_uni.summary()
[381]


[382] gru_uni.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
[383] es_callback = EarlyStopping(monitor='val_loss', patience=3)
```

```
[384] GRU_NET_uni = gru_uni.fit(train_sentences, train_tags, epochs=10,
  validation_split=0.2, callbacks=[es_callback], shuffle=False)
[385]

[386] plt.plot(GRU_NET_uni.history['accuracy'])
[387] plt.plot(GRU_NET_uni.history['val_accuracy'])
[388] plt.title('Model accuracy')
[389] plt.ylabel('Accuracy')
[390] plt.xlabel('Epoch')
[391] plt.legend(['Train', 'Validation'], loc='upper left')
[392] plt.show()
[393]


[394] plt.plot(GRU_NET_uni.history['loss'])
[395] plt.plot(GRU_NET_uni.history['val_loss'])
[396] plt.title('Model loss')
[397] plt.ylabel('Loss')
[398] plt.xlabel('Epoch')
[399] plt.legend(['Train', 'Validation'], loc='upper left')
[400] plt.show()
[401]

[402] print(classification_report(gru_uni.predict(test_sentences).round(), test_tags))
[403]

[404] # ### Bi-Directional GRU
[405]
[406] from keras.layers import Embedding, Bidirectional, GRU, Dense, Dropout
[407]
[408] gru_bi = Sequential()
[409] gru_bi.add(Embedding(VOCAB_SIZE+1, DIMENSIONS,
[410]                     embeddings_initializer=Init.Constant(words2vec_matrix),
[411]                     input_length=MAX_LEN, trainable=False))
[412] gru_bi.add(Bidirectional(GRU(256, activation='tanh', dropout=0.2)))
[413] gru_bi.add(Dense(128, activation='tanh'))
[414] gru_bi.add(Dropout(0.5))
[415] gru_bi.add(Dense(64, activation='tanh'))
[416] gru_bi.add(Dropout(0.5))
[417] gru_bi.add(Dense(1, activation='sigmoid'))
[418]
[419] gru_bi.summary()
[420]


[421] gru_bi.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
[422] es_callback = EarlyStopping(monitor='val_loss', patience=3)
[423] GRU_NET_bi = gru_bi.fit(train_sentences, train_tags, epochs=10, validation_split=0.2,
  callbacks=[es_callback], shuffle=False)
[424]
```

```
[425] plt.plot(GRU_NET_bi.history['accuracy'])
[426] plt.plot(GRU_NET_bi.history['val_accuracy'])
[427] plt.title('Model accuracy')
[428] plt.ylabel('Accuracy')
[429] plt.xlabel('Epoch')
[430] plt.legend(['Train', 'Validation'], loc='upper left')
[431] plt.show()
[432]

[433] plt.plot(GRU_NET_bi.history['loss'])
[434] plt.plot(GRU_NET_bi.history['val_loss'])
[435] plt.title('Model loss')
[436] plt.ylabel('Loss')
[437] plt.xlabel('Epoch')
[438] plt.legend(['Train', 'Validation'], loc='upper left')
[439] plt.show()
[440]


[441] print(classification_report(gru_bi.predict(test_sentences).round(), test_tags))
[442]
```