Twitter Sentiment Analysis-NLP

CS156 - Spring 2020

Ahmed Abdelrahman

Table of Content:

Sentiment Analysis:

Motivation:

With the huge development in social media, people get more freedom to express their feelings, experience, and even political opinions. Twitter is one of the biggest social media platforms with over 330 million monthly active users. This paper performs a detailed sentiment analysis using NLP to understand patterns and behaviors in users' tweets. I am building a model to classify and predict negative speech in tweets. I am training a model to identify and classify tweets to be a negative or a positive text based on their characteristics. I utilized the NLTK (Natural Language Toolkit) library and other classification libraries to achieve the discussed goal.

Data Set Description:

I have chosen a big data set with 1.6 million entries to have a better chance of getting a high training score and be able to generalize the results with more tweets and external data sets. I have chosen "Sentiment140" data set, which is originated from Stanford University: http://help.sentiment140.com/for-students/ . The data consists of 6-columns:

Sentiment: the polarity of the tweet (0=negative, 4=positive), where there are 800000 entries for each category

ID: the id of the tweet

Date: the date of the tweet

Query_string: the query (lyx). If there is no query, then the column value is No_query

User: the tweet owner

Text: the text of the tweet itself

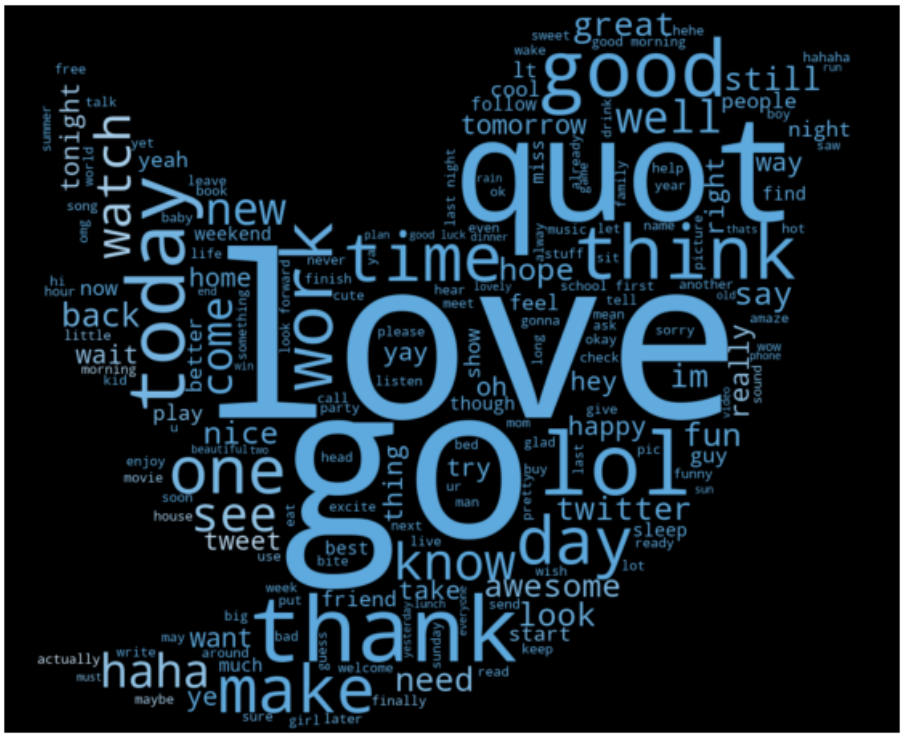
Data Cleaning:

Since I am only interested in the text of the tweet and its sentiment, I have dropped the rest of the columns to end up with a dataset_shape(16000000, 2). I have mapped all the positive values to be 1 to make in the classification part. Then, I have passed each tweet in the data frame through a cleaning function to remove punctuation, numbers, links, hashtags '#', mentions '@', and only leaves words with more than one character. To avoid noise and irrelevant words, I have dropped all the Stop_words using NLTK stop words list. Stop words are the most common words in the English language that doesn't add a lot of meaning to the tweet such as "the", "a", "and", etc. Finally, I have added all the clean tweets to the data frame in an additional column "Clean_Tweets".

Data Visualization:

I have generated wordcloud for tweets with positive and negative sentiments to visualize the most frequent words in the data set. We can see that the most frequent words in positive tweets are love, thank, and good which all have positive connotations. In negative tweets, we can see words like think, quote, miss, hate, and work. Most of these words don't have a strong negative connotation, but they can represent negative feelings. E.g, work, and miss can have a negative meaning based on the rest of the sentence. As we are only visualizing the most frequent words, I am sure there are more strong negative words but they are not as frequent.

Positive Tweets:



Negative Tweets:

Word Tokenization and Text Lemmatization:

I have tokenized every tweet into its component sentences, and each sentence into its component words. I have used text lemmatization to reduce words' inflectional forms and derivations to their unified roots. Thus, words like "nation, national, nationality" will go to their original base as "nation". I have used text lemmatization instead of text stemming to be able to capture the correct base for each word using vocabulary and morphological analysis. Stemmers usually clear the ends of words in the hope of returning to their base, which can cause multiple errors in different contexts

Feature Extraction:

Machine learning techniques and algorithms can't work with words and text directly, they need to be converted into a vector of numbers. In this section, I am exploring two feature extraction techniques (bag-of-words) and (TF-IDF) and explaining their effect on different machine learning classification models.

Bad-of-words:

This technique is built on the intuition of similar documents have similar content. After cleaning the tweets (corpus), I have built a list of vocabulary based on all vocabularies in the data. I have used CountVectorizer to count the frequency of a word in each tweet. E.g.,
If we have two documents D1:" I love Minerva", D2: " I love dogs but I hate cats", the count vectors matrix will be represented as:

	I	Love	Minerva	Dogs	But	Hate	Cats
D1	1	1	1	0	0	0	0
D2	2	1	0	1	1	1	1

Using big data, the size of the count vector-matrix can be so big to process. Thus, I have experimented with different top max_features in countvectorizer to illustrate the effect of max_features values on machine learning classification models' performances; which will be illustrated in the model comparison section.

TF-IDF:

Depending only on words frequency can raise a couple of errors as most frequent words are not guaranteed to be the most useful ones "Contains Significant Information". Term frequency-inverse document frequency, TF-IDF, is a statistical quantity to measure words weights/importance to a document in the corpus. The weight of the word increases by its appearance frequency in the document but also it is offset by the number of documents that contain the word.

$$W_{x,y} = tf_{x,y} * log(\frac{N}{df_x})$$

$W_{x,y}$: $TF - IDF$ *term x within document y*, $tf_{x,y}$: *frequency of x in y*

df_x : *number of docyments containing x*, N : *total number of documents*

Term Frequency (TF): Frequency of the word in a document.

$$TF(term) = \frac{Number\ of\ times\ term\ appears\ in\ a\ document}{Total\ number\ of\ items\ in\ the\ document}$$

Inverse Term Frequency (IDF): a measure of how rare the word across documents

$$IDF(term) = log(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ the\ word\ in\ it})$$

$$TFIDF(term) = TF(term) * IDF(term)$$

I have experimented with different max_features values in TfidfVectorizer to illustrate the effect of max_features values on machine learning classification models' performances; which will be illustrated in the model comparison section.

Models Comparision:

Coming back to the main goal of the section of classifying positive and negative tweets and predicting non-labeled data, I have tested the performance of Logistic Regression, XGBoost and Decision Trees to pick the best one for the data.

Logistic Regression: logistic regression is a supervised linear model for classification, rather than regression. Logistic regression is suitable for binary, One-vs-Rest, and multinomial logistic regression classification problems. I have chosen "liblinear" to be the solver of my model as it's suitable for large linear classification by using coordinate algorithms.

XGBoost: XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions.

Decision Trees: I am using DTs as a classification non-parametric supervised learning method for predicting tweets labels (positive-negative). The main advantage of decision trees is that they are easy to implement and understand and requires little data preparation.

Linear SVM: Support Vector Machine is a linear model for classification that solved linear and non-linear problems. The algorithm tries to fit a line between the data to separate them into classes, where it tries to maximize the separation between the two classes.

Ridge Classifier: This classifier first converts binary targets to {-1, 1}, and then treats the problem as a regression task. In practice, the ridge classifier can have a similar performance to logistic regression and traditional classifiers. With a high number of classes, Ridge Classifier can show a faster performance than logistic regression, because it is able to compute the projection matrix $(X^tX)^{-1}X^T$ only once (scikit-learn,2020).

Whiling trying to fit my data to these models, I have faced computational limitations with Colab RAM due to the data large size. Thus, I have decided to train my models with random 100,000 data points instead of 1.6 million. I believe that 100,000 data points will still enable me to capture most of the data set variations and providing me with a good prediction score.

Models Results:

Bag of Words Feature extraction

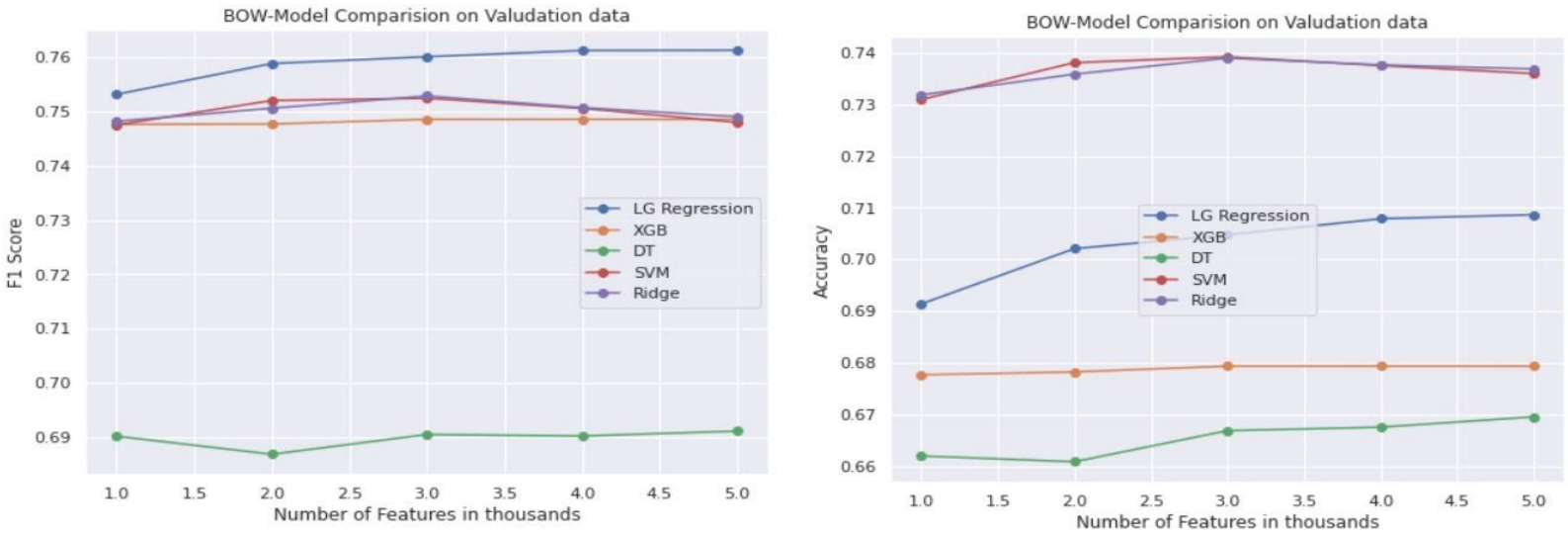


Figure-1. A comparison of different classification models using 5000 max_features. The figure is showing model performance using F1 Score and Accuracy.

As shown in the figures, the logistic regression model has a slightly higher F1-Score than SVM and Ridge classification models. On the other hand, linear SVM and Ridge classification models outperform the logistic regression model in the accuracy metric.

TF-IDF Feature extraction

Figure-2. A comparison of different classification models using 5000 max_features. The figure is showing model performance using F1 Score and Accuracy.

As shown in the figures, the logistic regression model has a slightly higher F1-Score than SVM, Ridge cand XGB classification models. On the other hand, SVM and Ridge classification models outperform the rest of the models in the accuracy metric.

Also, as we increase the max_features in the feature extraction stage, we notice a general improvement in the model performances. Finally, I have summarized the highest F1 and accuracy scores for all models (Figure-3).

	LG Regression	SVM	Ridge	XGB	Decision Trees
Bag-of-words F-1 Score	0.761	0.752	0.753	0.749	0.691
TF-IDF F-1 Score	0.765	0.752	0.750	0.752	0.692

	LG Regression	SVM	Ridge	XGB	Decision Trees
Bag-of-words Accuracy	0.709	0.739	0.739	0.679	0.670
TF-IDF Accuracy	0.710	0.741	0.739	0.687	0.679

Figure-3. A Table showing a comparison of different model accuracy.

The table shows the important information:

1- TF-IDF extraction models provide better results than bag of words method. As we increase the number of extracted features, as we obtain better performance in both metrics.

2- Logistic regression has a slightly higher F-1 score than SVM and Ridge models. However, SVM has the highest accuracy score among other models.

Considering the results from the above tables, I have chosen to apply SVM model on the test data set for its high accuracy rate and its ability to have a relatively high F-1 score as well. Also, the TF-IDF feature extraction method will be applied with max_features=4000 to ensure the

highest performance for the model. Finally, the number of features experiment was determined to be no higher than 5000 as I have faced RAM computational challenges for higher values of max_features.

Model Metric:

There are different metrics that can be used to evaluate the performance of binary classification models. The performance of binary classification problems can be well represented using a Confusion matrix (Figure-4). The confusion matrix enables us to calculate important metrics such as Recall, Precision, Specificity, and Accuracy.

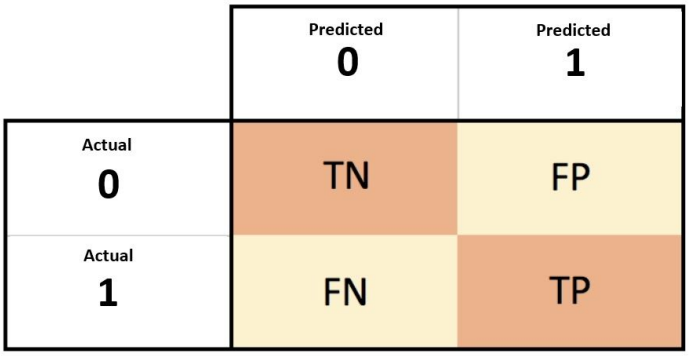


Figure-4. A confusion metric for binary classification problems

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$\text{Accuracy} = \frac{True\ Positive + True\ Negative}{Total}$$

Accuracy is a good metric to use as it describes the number of times the model predicted correctly over the total number of sets. Accuracy is best used when the distribution of the classes

is well balanced in the data, which means the number of positive tweets is equal to the number of negative ones. And, this is the case for our data as we have half of the data points as positive and the other half as negative. However, when one of the classes is dominating the dataset, we can encounter a high number of false-positive of a false negative. In order to generalize my model to any dataset, I have considered F1 score to be my second evaluation metric with the confusion matrix as well. F1 score considers both precision and recall, which means maximizing F1 is maximizing the ratio between precision and recall.

$$\text{F1 Score} = 2 * \frac{Precision * Recall}{Percision + Recall}$$

Final Model Results (SVM):

```
Accuracy Score on Test Data: 0.7422
F-1 Score on Test Data: 0.7503389502227389
-------------------------------------
The model classification report:
              precision    recall  f1-score   support

           0       0.75      0.72      0.73      4931
           1       0.74      0.76      0.75      5069

    accuracy                           0.74     10000
   macro avg       0.74      0.74      0.74     10000
weighted avg       0.74      0.74      0.74     10000

-------------------------------------
The model confusion matrix:
[[3548 1383]
 [1195 3874]]
```

In the obtained results, we can see that we almost have the same number of data points in each category (Positive, Negative), which makes accuracy a good metric to measure. From the classification report, we can see that the model has similar precision and recall values in both classes (positive, negative), and it was averaged out by calculating the f-score. By considering

accuracy and F-1 Scores, I am evaluating the performance of the model in a better way. Finally, I have implemented a grid search to find the best hyperparameters for LinearSVC and I have obtained a result of 0.75 accuracy score for `{'C': 0.1}`.

Model Extension (Neural Network):

After experimenting with different models, I noticed that SVM provides the best accuracy score over the test data. Due to the similarities of SVM and NN, I thought that utilizing a Neural Network model can be helpful to solve the classifying problem. I have used ADAM optimizer for optimizing the parameters and minimizing the cost function of the neural network. Considering NN to be an optimization problem, I have chosen binary_crossentropy to be my loss function. A loss function is a function that we want to minimize in order to get the lowest error. We can think of Crossentropy as the error function for predicting the probability of the input to belong to one of the classes. “Therefore, under maximum likelihood estimation, we would seek a set of model weights that minimize the difference between the model’s predicted probability distribution given the dataset and the distribution of probabilities in the training dataset. This is called the cross-entropy”. I have chosen binary cross-entropy to match the problem objective of binary classifying inputs (0,1).

Moreover, I have varied a dropout technique with 50% ratio to reduce model overfitting. According to the research paper "Improving neural networks by preventing co-adaptation of feature detectors" by Hinton et al. (2012), "A good way to reduce the error on the test set is to average the predictions produced by a very large number of different networks. The standard way to do this is to train many separate networks and then to apply each of these networks to the test data, but this is computationally expensive during both training and testing. Random dropout makes it possible to train a huge number of different networks in a reasonable time."
The obtained results can be summarized in the following figures:

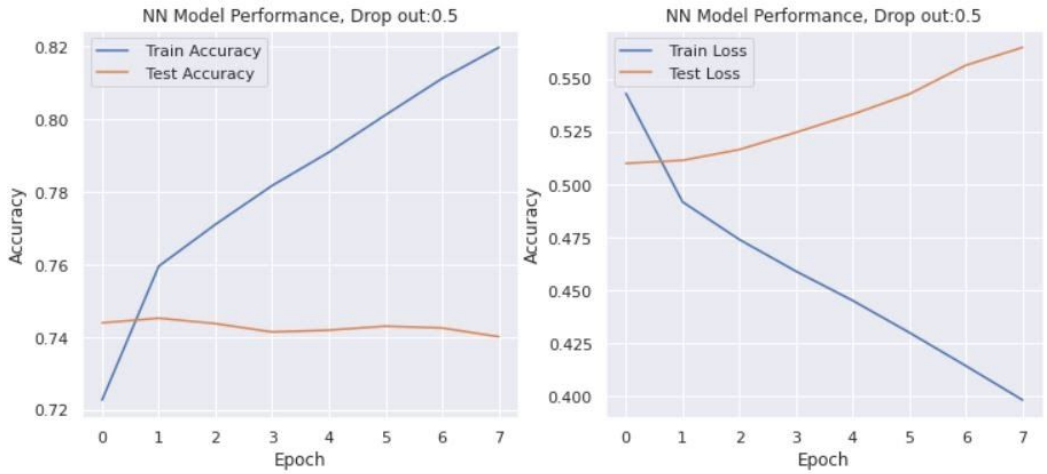


Figure-5. NN model performance with different epochs.

```
Accuracy Score on Test Data: 0.718
F-1 Score on Test Data: 0.7626662178084498
-------------------------------------
The model classification report:
              precision    recall  f1-score   support

           0       0.83      0.54      0.65      4931
           1       0.67      0.89      0.76      5069

    accuracy                           0.72     10000
   macro avg       0.75      0.72      0.71     10000
weighted avg       0.75      0.72      0.71     10000

-------------------------------------
The model confusion matrix:
[[2649 2282]
 [ 538 4531]]
```

Figure-6. The figure represents the NN model performance on test data.

As shown in Figure-6, the model has accuracy score of 0.72, which is lower than the obtained results from the SVM model. I have tried increasing the Epochs iterations but reached similar accuracy scores as well. Also, I have varied the dropout rate to reduce the model overfitting behavior and reached to the best results when dropout=0.5. However, I believe that overfitting

can be even reduced more by adding more layers or changing the structure of the NN. I have examined different variations and the model above provided the best accuracy/time balance.

Appendix:

This is assignment was a learning opportunity for me to showcase most of the skills that I have learned during CS-156 course. I have chosen a real-life problem and illustrated my approach to solve it and represented my findings in a professional way. The LOs that I used includes, but not limited to:

#Classification: I have utilized the skills and the concept of classification to solve the sentiment analysis. I have used various models and chose the most suitable model with the highest performance.

#regressionalgorithm: I have worked with different regression algorithms and evaluated their performance based on the nature of the data set. To be specific, I provided a clear, detailed, well-justified and accurate explanation supporting the choice, or provides extensive evidence that other competing alternatives would not solve the problem as effectively

#modelmetrics: I provided a clear, detailed, well-justified, and accurate explanation behind my model metrics (F1-score, accuracy) and why these metrics can provide a better evaluation of the model performance than others.

#neuralnetwork: I have correctly used neural network model and provided a detailed and provided a well-justified and accurate explanation supporting my choice. I was able to explain my parameters choice and correctly critique my model.

HCS: #optimization, #dataviz, #organization

References:

Brownlee, J. (2019, October 22). Loss and Loss Functions for Training Deep Learning Neural Networks. Retrieved from https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

Das, D. (2019, October 18). Social Media Sentiment Analysis using Machine Learning : Part - II. Retrieved from https://towardsdatascience.com/social-media-sentiment-analysis-part-ii-bcacca5aaa39

Kim, R. (2018, January 31). Another Twitter sentiment analysis with Python - Part 9 (Neural Networks with Tfidf vectors using... Retrieved from https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-9-neural-networks-with-tfidf-vectors-using-d0b4af6be6d7

Kim, R. (2018, January 12). Another Twitter sentiment analysis with Python - Part 4 (Count vectorizer, confusion matrix). Retrieved from https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-4-count-vectorizer-b3f4944e51b5

Kim, R. (2018, January 13). Another Twitter sentiment analysis with Python - Part 5 (Tfidf vectorizer, model comparison... Retrieved from https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-5-50b4e87d9bdd

Yordanov, V. (2019, August 13). Introduction to Natural Language Processing for Text. Retrieved from https://towardsdatascience.com/introduction-to-natural-language-processing-for-text-df845750fb63

Sentiment Analysis

April 29, 2020

Cleaning and Pre-Proccessing the Data

Ahmed Abdelrahman- Final Project

[1]:
```
#importing libraries

import re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import string
import nltk
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
sns.set()
%matplotlib inline
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

[2]:
```
#Importing the data from my drive
#zero for negative tweets, 1 for positive ones

all_data = pd.read_csv("drive/Shared drives/CS166-FP-DATA/Part-1/training.
↪1600000.processed.noemoticon.csv",header=None,
                 usecols=[0,5],names=['sentiment','text'],encoding =␣
↪"ISO-8859-1")
all_data['sentiment'] = all_data['sentiment'].map({0: 0, 4: 1})
all_data.head()
```

[2]:

	sentiment	text
0	0	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	is upset that he can't update his Facebook by ...
2	0	@Kenichan I dived many times for the ball. Man...
3	0	my whole body feels itchy and like its on fire

```
4          0  @nationwideclass no, it's not behaving at all...
```

[3]:
```
nltk.download('stopwords')
stopwords = nltk.corpus.stopwords.words("english")

# Clean tweets for sentiment analysis
def get_clean_words(words):
    def _isnum(w):
        try:
            int(w)
            return True
        except ValueError:
            return False

    # Set words to lowercase and remove them if they are stop words
    words = [w.lower() for w in words if w.lower() not in stopwords]

    # Removing punctuation
    words = [w.replace('(', '') for w in words]
    words = [w.replace(')', '') for w in words]
    words = [w.replace('?', '') for w in words]
    words = [w.replace(',', '') for w in words]
    words = [w.replace('.', '') for w in words]
    words = [w.replace('"', '') for w in words]
    words = [w.replace('!', '') for w in words]
    words = [w.replace(':', '') for w in words]
    words = [w.replace('&amp;', '') for w in words]
    words = [w.replace('/', '') for w in words]
    words = [w.replace('[', '') for w in words]
    words = [w.replace(']', '') for w in words]

    # Removing numbers
    words = [w for w in words if not _isnum(w)]

    # Removing links
    words = [w for w in words if 'http' not in w]

    # Removing hashtags
    words = [w for w in words if not w.startswith("#")]

    # Removing mentions
    words = [w for w in words if not w.startswith("@")]

    # Keeping words with more than 1 character
    words = [w for w in words if len(w) > 1]
    w = " ".join(words)
    return w
```

```
#This Function was inspired from Mudium Post:https://towardsdatascience.com
#/analyzing-twitter-spheres-through-nlp-techniques-748b0df10b6c
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[0]:
```
# Funtion to pass each tweet into the clean words function and create a new list
def clean_tweets(df):
    new_text = []
    for i in range(len(df)):
        new_text.append(get_clean_words(df['text'].values[i].split()))
    all_data['Clean_Tweets'] = new_text
```

[5]:
```
#visualizing the data after being cleaned
clean_tweets(all_data)
all_data.head()
```

[5]:

	sentiment	...	Clean_Tweets
0	0	...	awww that's bummer shoulda got david carr thir...
1	0	...	upset can't update facebook texting it might c...
2	0	...	dived many times ball managed save 50% rest go...
3	0	...	whole body feels itchy like fire
4	0	...	no behaving all i'm mad here can't see there

[5 rows x 3 columns]

[6]:
```
#Lemmatization

nltk.download('wordnet')
from nltk.corpus import wordnet
from nltk import WordNetLemmatizer
tokenized_tweet = all_data['Clean_Tweets'].apply(lambda x: x.split())
#tokenized_tweet.head()

wl = WordNetLemmatizer()
tokenized_tweet = tokenized_tweet.apply(lambda x: [wl.lemmatize(word=i, pos =␣
 ↪wordnet.VERB) for i in x])
tokenized_tweet.head()
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

[6]:
```
0    [awww, that's, bummer, shoulda, get, david, ca...
1    [upset, can't, update, facebook, texting, it, ...
2    [dive, many, time, ball, manage, save, 50%, re...
3                [whole, body, feel, itchy, like, fire]
```

```
4    [no, behave, all, i'm, mad, here, can't, see, ...
Name: Clean_Tweets, dtype: object
```

[7]:
```
for i in range(len(tokenized_tweet)):
    tokenized_tweet[i] = ' '.join(tokenized_tweet[i])
all_data['Clean_Tweets'] = tokenized_tweet
all_data.head()
```

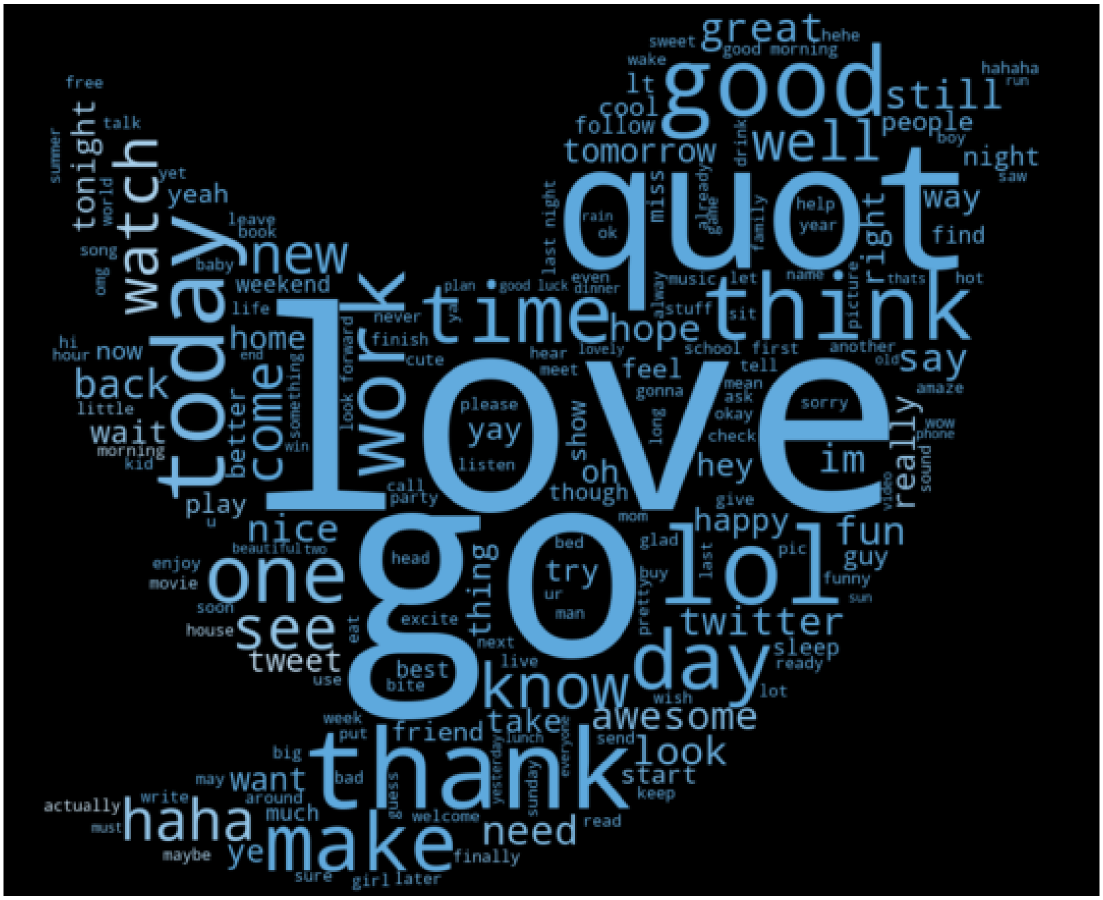
[7]:
```
   sentiment  ...                                       Clean_Tweets
0          0  ...  awww that's bummer shoulda get david carr thir...
1          0  ...  upset can't update facebook texting it might c...
2          0  ...  dive many time ball manage save 50% rest go bound
3          0  ...                    whole body feel itchy like fire
4          0  ...         no behave all i'm mad here can't see there

[5 rows x 3 columns]
```

Model Visualization

[0]:
```
from wordcloud import WordCloud,ImageColorGenerator
from PIL import Image
import urllib
import requests
```

[0]:
```
model_data=all_data.sample(n = 100000)
```

[78]:
```
#One is all the positive words
all_words_positive = ' '.join(text for text in␣
 ↪model_data['Clean_Tweets'][model_data['sentiment']==1])
# combining the image with the dataset
Mask = np.array(Image.open(requests.get('http://clipart-library.com/
 ↪image_gallery2/Twitter-PNG-Image.png', stream=True).raw))

image_colors = ImageColorGenerator(Mask)
wc = WordCloud(background_color='black', height=1500, width=4000,mask=Mask).
 ↪generate(all_words_positive)

# Size of the image generated
plt.figure(figsize=(10,20))

plt.imshow(wc.recolor(color_func=image_colors),interpolation="hamming")

plt.axis('off')
plt.show()
```




```
[80]: #Negative words:

      all_words_negative = ' '.join(text for text in␣
       ↪all_data['Clean_Tweets'][all_data['sentiment']==0])
      Mask = np.array(Image.open(requests.get('http://clipart-library.com/data_images/
       ↪362062.jpg', stream=True).raw))
      image_colors = ImageColorGenerator(Mask)

      # Now we use the WordCloud function from the wordcloud library
      wc = WordCloud(background_color='white', height=1500, width=4000,mask=Mask).
       ↪generate(all_words_negative)

      # Size of the image generated
      plt.figure(figsize=(10,20))

      plt.imshow(wc.recolor(color_func=image_colors),interpolation="gaussian")

      plt.axis('off')
```

```
plt.show()
```

Prediction and Model Comparision

[0]:
```
from sklearn.model_selection import train_test_split
rann = 2000
x_train,x_test,y_train,y_test=train_test_split(
    model_data['Clean_Tweets'],model_data['sentiment'],
    test_size=0.1,random_state=rann)
```

[0]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
```

```
# A function to provide the metrics scores for various classifiers
# The function uses bag-of-words vectorization feature extraction
def model_features_scores(classifier,x,y,linear=False):
  results=[]
  results_acc=[]
  features_range=np.arange(1000,5001,1000)
  for num in features_range:
    #building bag of words
    bow_vectorizer= CountVectorizer(max_df=0.90, min_df=2, max_features=num,
                                    stop_words='english')
    # bag-of-words feature matrix
    bow = bow_vectorizer.fit_transform(x)

    #getting the train part of the data
    train_bow = bow
    train_bow.todense()
    x_train_bow,x_valid_bow,y_train_bow,y_valid_bow = train_test_split(
        train_bow,y,test_size=0.1,random_state=200)

    classifier.fit(x_train_bow,y_train_bow)
    if linear:
      prediction_bow = classifier.predict(x_valid_bow)
      prediction_int = prediction_bow
    else:
      prediction_bow = classifier.predict_proba(x_valid_bow)
      prediction_int = prediction_bow[:,1]>=0.3


    # if prediction is greater than or equal to 0.3 then 1 else 0
    # Where 0 is for negative sentiment tweets and 1 for positive sentiment␣
 ↪tweets

    prediction_int = prediction_int.astype(np.int)

    # calculating f1 score and accuracy
    score = f1_score(y_valid_bow, prediction_int)
    results.append(score)
    accuracy = accuracy_score(y_valid_bow, prediction_int)
    results_acc.append(accuracy)

  return results,results_acc
```

[101]:
```
#Fitting different classification algorithms and getting accuracy and
#F-1 scores

from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
```

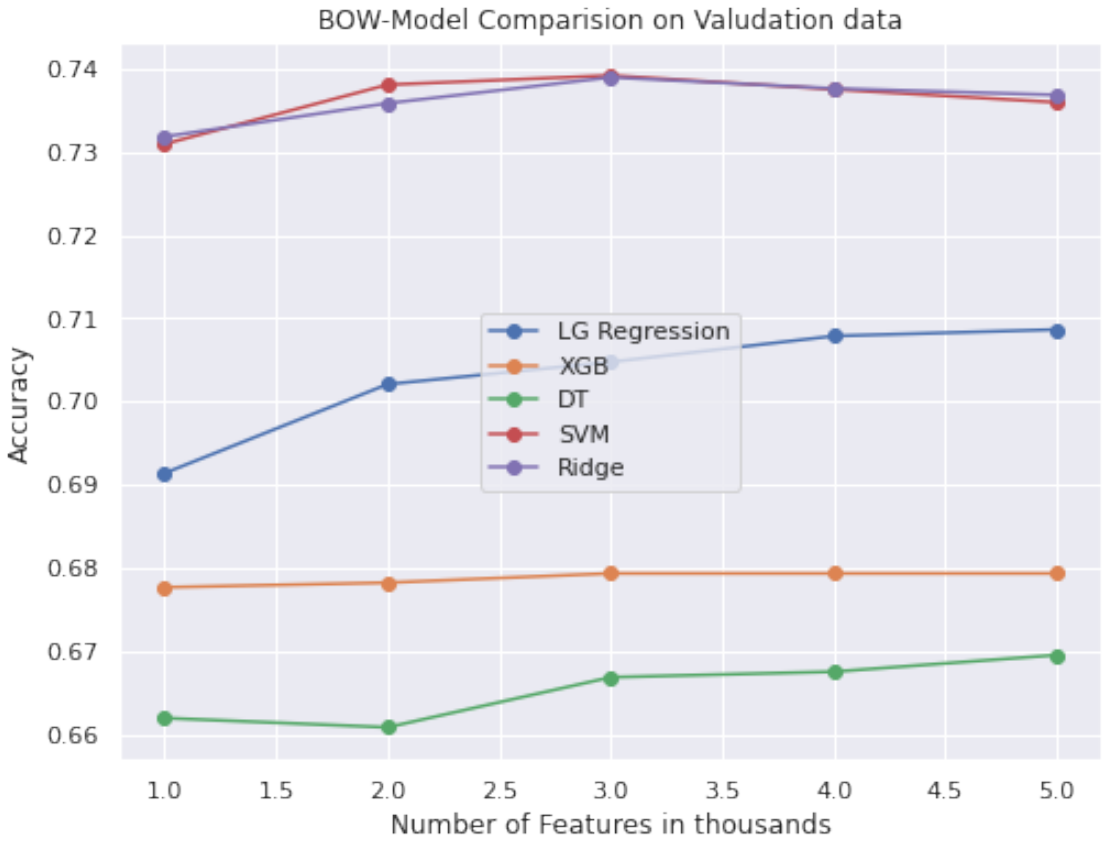
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC
from sklearn.linear_model import RidgeClassifier

Log_Reg = LogisticRegression(solver='liblinear',random_state=200)
Log_bow_scores,Log_bow_accuracy=model_features_scores(Log_Reg,x_train,y_train)

XGB_model= XGBClassifier(learning_rate=0.9,random_state=200)
XGB_bow_scores,XGB_bow_accuracy=model_features_scores(XGB_model,x_train,y_train)

DT_model = DecisionTreeClassifier(criterion='entropy', random_state=200)
DT_bow_scores,DT_bow_accuracy=model_features_scores(DT_model,x_train,y_train)

SVC_Reg=LinearSVC(max_iter=2000,random_state=200)
SVC_bow_scores,SVC_bow_accuracy=model_features_scores(SVC_Reg,x_train,y_train,linear=True)

Ridge_Reg=RidgeClassifier(random_state=200)
Ridge_bow_scores,Ridge_bow_accuracy=model_features_scores(Ridge_Reg,x_train,y_train,linear=True
```

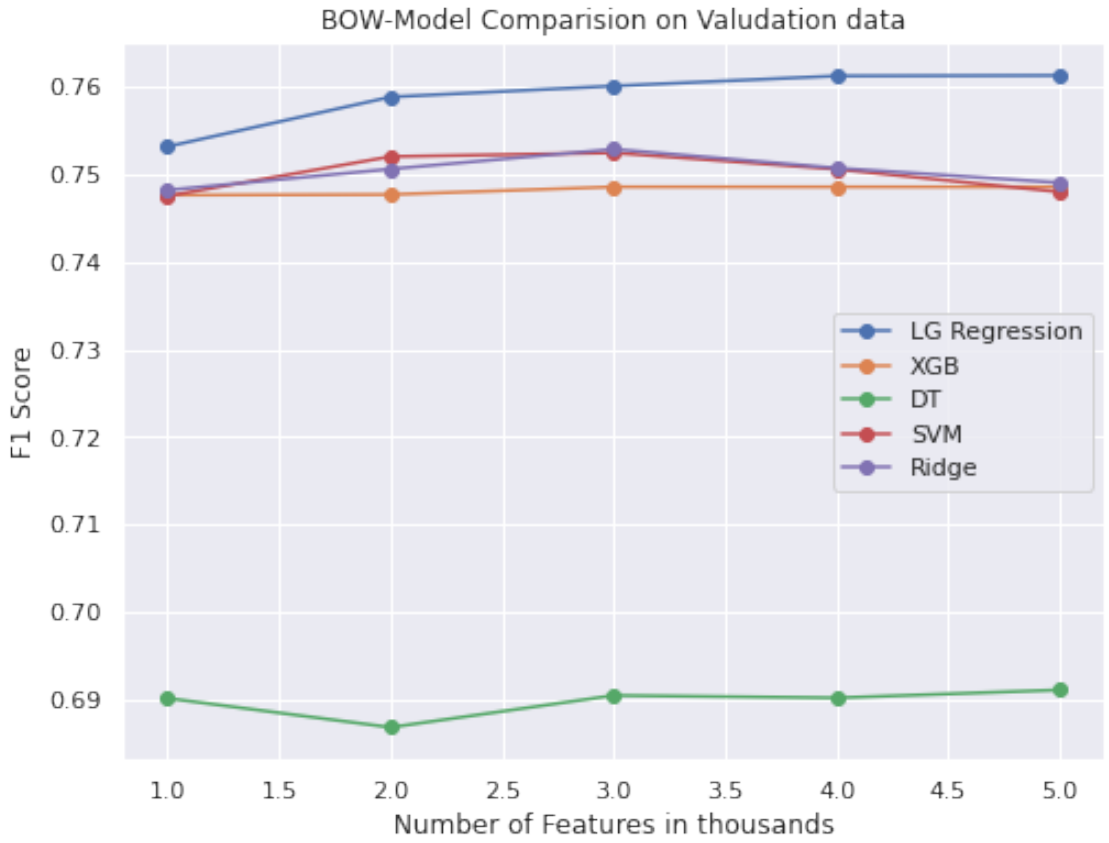
```
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:947:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
```

[102]:

```
#Plotting different classification models performances
#over the bag of words features

plt.figure(figsize=(8, 6))
plt.plot([*range(1,6,1)] ,Log_bow_accuracy,'o-',label='LG Regression')
plt.plot([*range(1,6,1)] ,XGB_bow_accuracy,'o-',label='XGB')
plt.plot([*range(1,6,1)] ,DT_bow_accuracy,'o-',label='DT')
```

```
plt.plot([*range(1,6,1)] ,SVC_bow_accuracy,'o-',label='SVM')
plt.plot([*range(1,6,1)] ,Ridge_bow_accuracy,'o-',label='Ridge')

plt.title("BOW-Model Comparision on Valudation data")
plt.xlabel("Number of Features in thousands")
plt.ylabel("Accuracy")
plt.legend()
plt.show

##################


plt.figure(figsize=(8, 6))

plt.plot([*range(1,6,1)] ,Log_bow_scores,'o-',label='LG Regression')
plt.plot([*range(1,6,1)] ,XGB_bow_scores,'o-',label='XGB')
plt.plot([*range(1,6,1)] ,DT_bow_scores,'o-',label='DT')
plt.plot([*range(1,6,1)] ,SVC_bow_scores,'o-',label='SVM')
plt.plot([*range(1,6,1)] ,Ridge_bow_scores,'o-',label='Ridge')

plt.title("BOW-Model Comparision on Valudation data")
plt.xlabel("Number of Features in thousands")
plt.ylabel("F1 Score")
plt.legend()
plt.show()
```


BOW-Model Comparision on Valudation data
Accuracy
Number of Features in thousands
LG Regression
XGB
DT
SVM
Ridge
0.74
0.73
0.72
0.71
0.70
0.69
0.68
0.67
0.66
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0



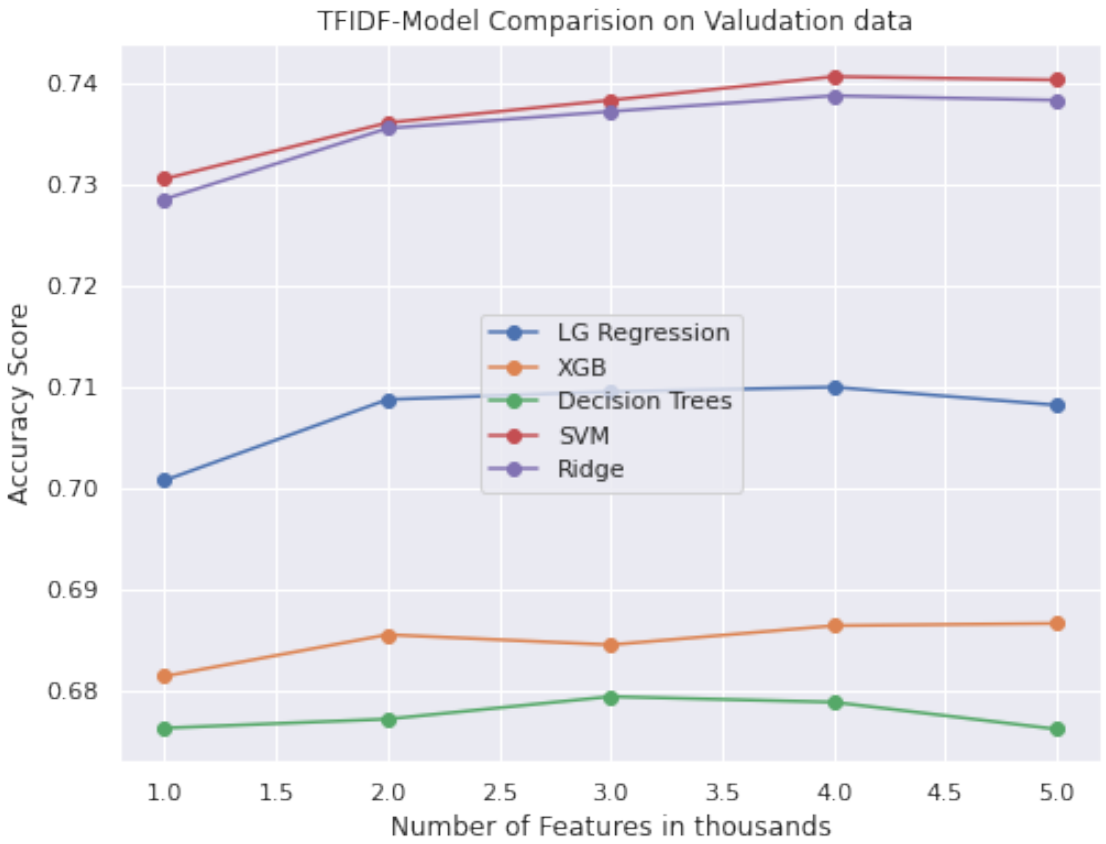
[0]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
# A function to provide the metrics scores for various classifiers
# The function uses TF-IDF vectorization feature extraction

def model_tfidf_features_scores(classifier,x,y,linear=False):
  results=[]
  results_acc=[]
  features_range=np.arange(1000,5001,1000)
  for num in features_range:
    #building tfidf matrix
    tfidf=TfidfVectorizer(max_df=0.90,␣
 ↪min_df=2,max_features=num,stop_words='english')
    train_tfidf_matrix=tfidf.fit_transform(x)

    #getting the train part of the data
    #train_tfidf_matrix = tfidf_matrix
    train_tfidf_matrix.todense()
    x_train_tfidf,x_valid_tfidf,y_train_tfidf,y_valid_tfidf= train_test_split(
        train_tfidf_matrix,y,test_size=0.1,random_state=200)
```

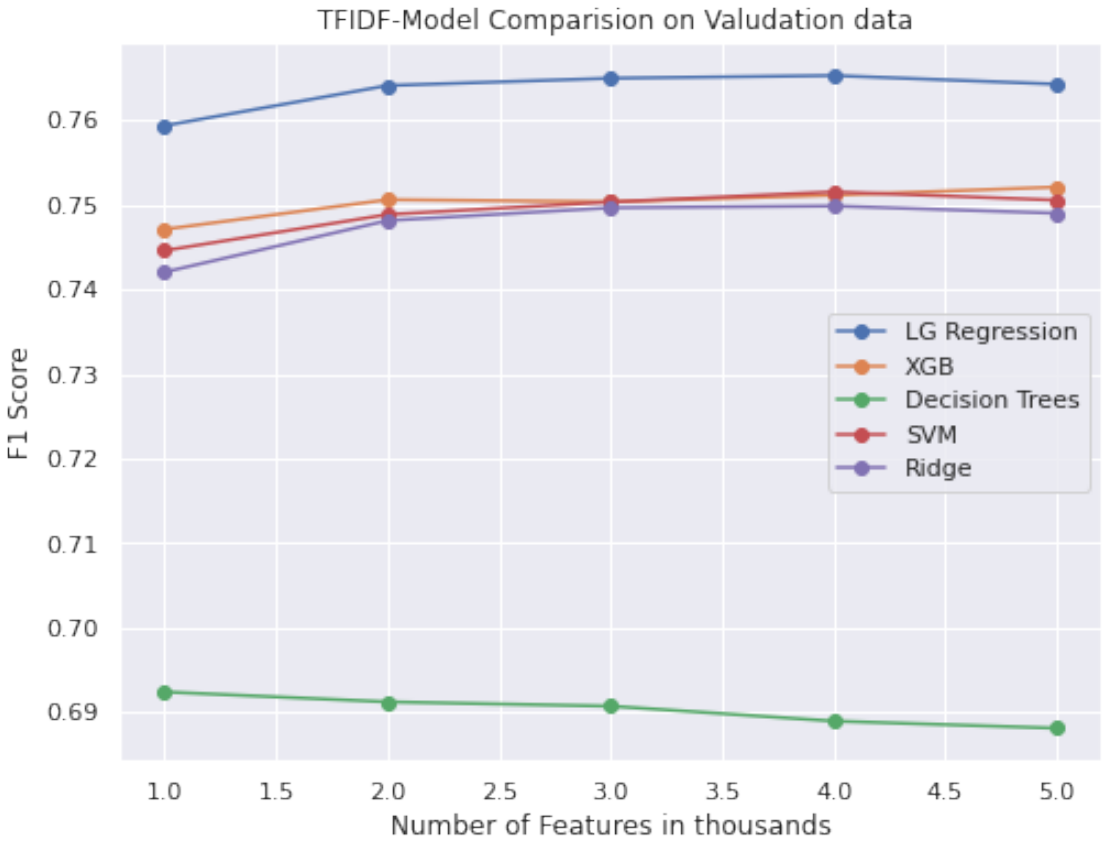
```
    classifier.fit(x_train_tfidf,y_train_tfidf)
    if linear:
      prediction_tfidf = classifier.predict(x_valid_tfidf)
      prediction_int = prediction_tfidf
    else:
      prediction_tfidf = classifier.predict_proba(x_valid_tfidf)
      prediction_int = prediction_tfidf[:,1]>=0.3

    prediction_int = prediction_int.astype(np.int)

    # calculating f1 score and accuracy
    score = f1_score(y_valid_tfidf, prediction_int)
    results.append(score)
    accuracy = accuracy_score(y_valid_tfidf, prediction_int)
    results_acc.append(accuracy)

  return results,results_acc
```

[0]:
```
#Fitting different classification algorithms and getting accuracy and
#F-1 scores

Log_Reg = LogisticRegression(solver='liblinear')
Log_tfidf_scores,Log_tfidf_accuracy=model_tfidf_features_scores(Log_Reg,x_train,y_train)

XGB_model= XGBClassifier(learning_rate=0.9)
XGB_tfidf_scores,XGB_tfidf_accuracy=model_tfidf_features_scores(XGB_model,x_train,y_train)

DT_model = DecisionTreeClassifier(criterion='entropy')
DT_tfidf_scores,DT_tfidf_accuracy=model_tfidf_features_scores(DT_model,x_train,y_train)

SVC_Reg=LinearSVC(max_iter=2000,random_state=200)
SVC_tfidf_scores,SVC_tfidf_accuracy=model_tfidf_features_scores(SVC_Reg,x_train,y_train,linear='

Ridge_Reg=RidgeClassifier(random_state=200)
Ridge_tfidf_scores,Ridge_tfidf_accuracy=model_tfidf_features_scores(Ridge_Reg,x_train,y_train,l
```

[31]:
```
#Plotting different classification models performances
#over the bag of words features

plt.figure(figsize=(8, 6))

plt.plot([*range(1,6,1)] ,Log_tfidf_accuracy,'o-',label='LG Regression')
plt.plot([*range(1,6,1)] ,XGB_tfidf_accuracy,'o-',label='XGB')
plt.plot([*range(1,6,1)] ,DT_tfidf_accuracy,'o-',label='Decision Trees')
plt.plot([*range(1,6,1)] ,SVC_tfidf_accuracy,'o-',label='SVM')
plt.plot([*range(1,6,1)] ,Ridge_tfidf_accuracy,'o-',label='Ridge')
```

```
plt.title(" TFIDF-Model Comparision on Valudation data")
plt.xlabel("Number of Features in thousands")
plt.ylabel("Accuracy Score")
plt.legend()
plt.show()

############################
plt.figure(figsize=(8, 6))


plt.plot([*range(1,6,1)] ,Log_tfidf_scores,'o-',label='LG Regression')
plt.plot([*range(1,6,1)] ,XGB_tfidf_scores,'o-',label='XGB')
plt.plot([*range(1,6,1)] ,DT_tfidf_scores,'o-',label='Decision Trees')
plt.plot([*range(1,6,1)] ,SVC_tfidf_scores,'o-',label='SVM')
plt.plot([*range(1,6,1)] ,Ridge_tfidf_scores,'o-',label='Ridge')

plt.title(" TFIDF-Model Comparision on Valudation data")
plt.xlabel("Number of Features in thousands")
plt.ylabel("F1 Score")
plt.legend()
plt.show()
```





[34]:
```
results = {
    'LG Regression':[max(Log_bow_accuracy), max(Log_tfidf_accuracy)],
    'SVM':[max(SVC_bow_accuracy), max(SVC_tfidf_accuracy)],
    'Ridge':[max(Ridge_bow_accuracy),max(Ridge_tfidf_accuracy)],
    'XGB':[max(XGB_bow_accuracy), max(XGB_tfidf_accuracy)],
    'Decision Trees':[max(DT_bow_accuracy), max(DT_tfidf_accuracy)]

}
pd.DataFrame(results, index=['Bag-of-words Accuracy', 'TF-IDF Accuracy']).
 ↪round(3)
```

[34]:

	LG Regression	SVM	Ridge	XGB	Decision Trees
Bag-of-words Accuracy	0.709	0.739	0.739	0.679	0.670
TF-IDF Accuracy	0.710	0.741	0.739	0.687	0.679

[35]:
```
# A summary of the model comparision plots
results = {
    'LG Regression':[max(Log_bow_scores), max(Log_tfidf_scores)],
    'SVM':[max(SVC_bow_scores), max(SVC_tfidf_scores)],
```

```
    'Ridge':[max(Ridge_bow_scores),max(Ridge_tfidf_scores)],
    'XGB':[max(XGB_bow_scores), max(XGB_tfidf_scores)],
    'Decision Trees':[max(DT_bow_scores), max(DT_tfidf_scores)]

}
pd.DataFrame(results, index=['Bag-of-words F-1 Score', 'TF-IDF F-1 Score']).
 ↪round(3)
```

[35]:

	LG Regression	SVM	Ridge	XGB	Decision Trees
Bag-of-words F-1 Score	0.761	0.752	0.753	0.749	0.691
TF-IDF F-1 Score	0.765	0.752	0.750	0.752	0.692

[76]:
```
from sklearn.metrics import classification_report, confusion_matrix
#Evaluating the test data
rann = 2000
tfidf=TfidfVectorizer(max_df=0.90,␣
 ↪min_df=2,max_features=5000,stop_words='english')
train_tfidf_matrix=tfidf.fit_transform(model_data['Clean_Tweets'])
#getting the train part of the data
#train_tfidf_matrix = tfidf_matrix
train_tfidf_matrix.todense()

x_train,x_vtest,y_train,y_vtest=train_test_split
(train_tfidf_matrix,model_data['sentiment'],test_size=0.2,random_state=rann)

x_test,x_valid,y_test,y_valid= train_test_split(
    x_vtest,y_vtest,test_size=0.5,random_state=200)


classifier = LinearSVC(max_iter=2000,random_state=200)
classifier.fit(x_train,y_train)
prediction_int = classifier.predict(x_test)
prediction_int = prediction_int.astype(np.int)

# Metrics
print("Accuracy Score on Test Data:",accuracy_score(y_test, prediction_int))
print("F-1 Score on Test Data:",f1_score(y_test, prediction_int))
print("-------------------------------------")
print("The model classification report:")
print(classification_report(y_test, prediction_int))
print("-------------------------------------")
print("The model confusion matrix:")
print(confusion_matrix(y_test, prediction_int))
```

```
Accuracy Score on Test Data: 0.7422
F-1 Score on Test Data: 0.7503389502227389
-------------------------------------
```

```
The model classification report:
              precision    recall  f1-score   support

           0       0.75      0.72      0.73      4931
           1       0.74      0.76      0.75      5069

    accuracy                           0.74     10000
   macro avg       0.74      0.74      0.74     10000
weighted avg       0.74      0.74      0.74     10000

-------------------------------------
The model confusion matrix:
[[3548 1383]
 [1195 3874]]
```

GridsearchCV-SVM

[119]:
```
#from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(LinearSVC(), {"C": [0.1,1,10,100]},cv=3, n_jobs=-1,␣
 ↪verbose=1)
grid.fit(x_train, y_train)
print(gs.best_params_, gs.best_score_)
print("Train Error:")
print(classification_report(y_train,gs.best_estimator_.predict(x_train)))
print("Test Error:")
print(classification_report(y_test,gs.best_estimator_.predict(x_test)))
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:   12.3s finished

{'C': 0.1} 0.7399750064813233
Train Error:
              precision    recall  f1-score   support

           0       0.79      0.74      0.76     39834
           1       0.76      0.80      0.78     40166

    accuracy                           0.77     80000
   macro avg       0.77      0.77      0.77     80000
weighted avg       0.77      0.77      0.77     80000

Test Error:
              precision    recall  f1-score   support

           0       0.76      0.72      0.74      4931
           1       0.74      0.77      0.76      5069
```

```
    accuracy                           0.75     10000
   macro avg       0.75      0.75      0.75     10000
weighted avg       0.75      0.75      0.75     10000
```

Nueral Network

[38]:
```
seed = 200
np.random.seed(seed)
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import Flatten
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
```

```
Using TensorFlow backend.
```

[88]:
```
#Building NN Model
model1 = Sequential()
model1.add(Dense(100, activation='relu', input_dim=5000))
model1.add(Dropout(0.5))
model1.add(Dense(1, activation='sigmoid'))
model1.compile(optimizer='adam',
               loss='binary_crossentropy',
               metrics=['accuracy'])

model1.fit(x_train,y_train
                  , validation_data=(x_valid, y_valid),
                  batch_size=30,
                  epochs=8,
                  verbose=1)
```

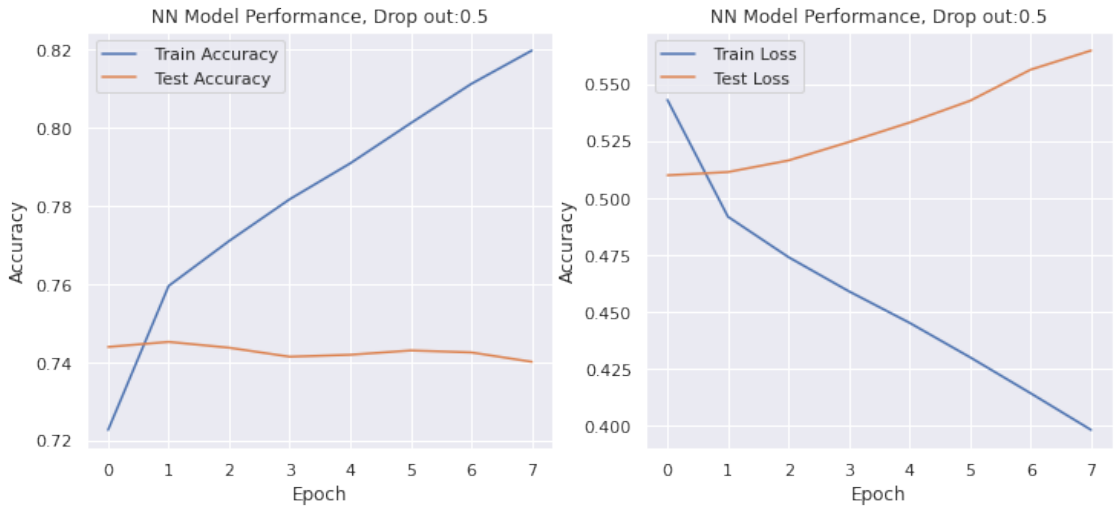
```
Train on 80000 samples, validate on 10000 samples
Epoch 1/8
80000/80000 [==============================] - 101s 1ms/step - loss: 0.5430 -
accuracy: 0.7227 - val_loss: 0.5099 - val_accuracy: 0.7439
Epoch 2/8
80000/80000 [==============================] - 102s 1ms/step - loss: 0.4918 -
accuracy: 0.7595 - val_loss: 0.5113 - val_accuracy: 0.7452
Epoch 3/8
80000/80000 [==============================] - 101s 1ms/step - loss: 0.4740 -
accuracy: 0.7710 - val_loss: 0.5164 - val_accuracy: 0.7437
Epoch 4/8
80000/80000 [==============================] - 101s 1ms/step - loss: 0.4590 -
accuracy: 0.7817 - val_loss: 0.5245 - val_accuracy: 0.7414
Epoch 5/8
80000/80000 [==============================] - 102s 1ms/step - loss: 0.4452 -
accuracy: 0.7908 - val_loss: 0.5330 - val_accuracy: 0.7419
Epoch 6/8
```

```
80000/80000 [==============================] - 103s 1ms/step - loss: 0.4301 -
accuracy: 0.8011 - val_loss: 0.5426 - val_accuracy: 0.7430
Epoch 7/8
80000/80000 [==============================] - 101s 1ms/step - loss: 0.4143 -
accuracy: 0.8112 - val_loss: 0.5562 - val_accuracy: 0.7425
Epoch 8/8
80000/80000 [==============================] - 101s 1ms/step - loss: 0.3982 -
accuracy: 0.8197 - val_loss: 0.5646 - val_accuracy: 0.7401
```

[88]:
```
<keras.callbacks.callbacks.History at 0x7f67404e8710>
```

[92]:
```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(model1.history.history['accuracy'], label=' Train Accuracy')
plt.plot(model1.history.history['val_accuracy'],label=' Test Accuracy')

plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("NN Model Performance, Drop out:0.5")
plt.legend()


plt.subplot(1, 2, 2)
plt.plot(model1.history.history['loss'], label=' Train Loss')
plt.plot(model1.history.history['val_loss'],label=' Test Loss')

plt.xlabel("Epoch")
plt.ylabel("Accuracy")

plt.title("NN Model Performance, Drop out:0.5")
plt.legend()
```

[92]:
```
<matplotlib.legend.Legend at 0x7f67381e1470>
```



[93]:
```
#Calculating the F-1 score of the data
NNprediction=model1.predict(x_test)
NNprediction_int=[]
for i in NNprediction:
  if i >=0.3:
    NNprediction_int.append(1)
  else:
    NNprediction_int.append(0)

# Metrics
print("Accuracy Score on Test Data:",accuracy_score(y_test, NNprediction_int))
print("F-1 Score on Test Data:",f1_score(y_test, NNprediction_int))
print("-------------------------------------")
print("The model classification report:")
print(classification_report(y_test, NNprediction_int))
print("-------------------------------------")
print("The model confusion matrix:")
print(confusion_matrix(y_test, NNprediction_int))
```

```
Accuracy Score on Test Data: 0.718
F-1 Score on Test Data: 0.7626662178084498
-------------------------------------
The model classification report:
              precision    recall  f1-score   support

           0       0.83      0.54      0.65      4931
           1       0.67      0.89      0.76      5069

    accuracy                           0.72     10000
   macro avg       0.75      0.72      0.71     10000
```

```
weighted avg       0.75      0.72      0.71     10000

-------------------------------------
The model confusion matrix:
[[2649 2282]
 [ 538 4531]]
```