Title: AIDI 1002 Final Term Project Report

Members' Names or Individual's Name: Kavuri Mahesh, Kashish Sunil Shah

Emails: 200545458@student.georgianc.on.ca, 200545460@student.georgianc.on.ca

Introduction:

Problem Description:

In the article thats published by the author, it demonstrated the basics of building a text classification model comparing Bag-of-Words (with Tf-Idf) and Word Embedding with Word2Vec. To investigate and develop more accurate text classification models by incorporating advanced word-embedding techniques, such as BERT, and evaluate their performance in combination with various classification algorithms, in order to overcome the limitations of traditional methods that were used by the author and achieve better classification outcomes.

Context of the Problem:

In this study, we aim to evaluate and compare the performance of various text classification models that utilize different feature extraction methods and classification algorithms. The importance of conducting this analysis lies in several aspects:

Comprehensive comparison: By comparing Bag-of-Words (with Tf-Idf), Word Embedding with Word2Vec, and advanced word-embedding methods like BERT, we can identify the strengths and weaknesses of each approach in the context of text classification. This will allow researchers and practitioners to make informed decisions on which methods to adopt for their specific applications.

Enhanced understanding: Exploring the performance of different classification algorithms, such as Support Vector Machines (SVM), XgBoost, Neural Networks, Random Forest, GaussianNB, and Logistic Regression, will provide insights into the suitability of each algorithm for handling different text classification problems. This will help in selecting the most appropriate algorithm for the given task.

Model optimization: By comparing the accuracies of various models, we can identify the best performing combinations of feature extraction methods and classification algorithms. This will guide us towards building more efficient and accurate text classification models.

Limitation About other Approaches:

Prior approaches, as mentioned in the article, have certain limitations, such as over-reliance on the Bag-of-Words model, which ignores word order and context, leading to suboptimal performance. Additionally, these methods may not have thoroughly explored the potential of advanced word-embedding techniques, like BERT, in combination with various classification algorithms, leaving room for further investigation and improvement.

Solution:

we are going to evaluate the model using other classification algorithms like Support Vector Machines (SVM), XgBoost, and Neural networks anc check the accuracies and also we are going to using advanced word-embedding methods BERT and evaluate the model accuracies using Random forest, GuassianNB, Ensemble Model (Voting Classifier) and Logistic regression so that the accuracies will improve

Background

Explain the related work using the following table

Reference	Explanation	Dataset/Input	Weakness
Vijaya rani	author demonstrated the basics of building a text classification model comparing Bag-of-Words (with Tf-Idf) and Word Embedding with Word2Vec	Natural Language Processing with Disaster Tweets	Only 66% accuracy using Word2Vec

Methodology

In the existing paper, the author demonstrated the process of building a text classification model using Bag-of-Words (with Tf-Idf) and Word Embedding with Word2Vec techniques. The methodology primarily focused on these feature extraction methods for text representation and classification.

Our contribution to this work involves expanding the evaluation by implementing additional classification algorithms such as Support Vector Machines (SVM), XgBoost, and Neural Networks. Furthermore, we will explore advanced word-embedding methods like BERT and assess its performance with classification algorithms like Random Forest, GaussianNB, and Logistic Regression. This extended methodology aims to enhance the overall accuracy of text classification models by exploring the potential of advanced techniques and diverse algorithms in combination. The detailed process and results will be presented in the subsequent sections, along with supporting figures.

Implementation

```
#Importing Necessary Libraries
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt

#for text pre-processing
import re, string
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

#for model-building
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, f1_score,
accuracy_score, confusion_matrix
from sklearn.metrics import roc_curve, auc, roc_auc_score

# bag of words
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

#for word embedding
import gensim
from gensim.models import Word2Vec #Word2Vec is mostly used for huge
datasets
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\91913\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\91913\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to
```

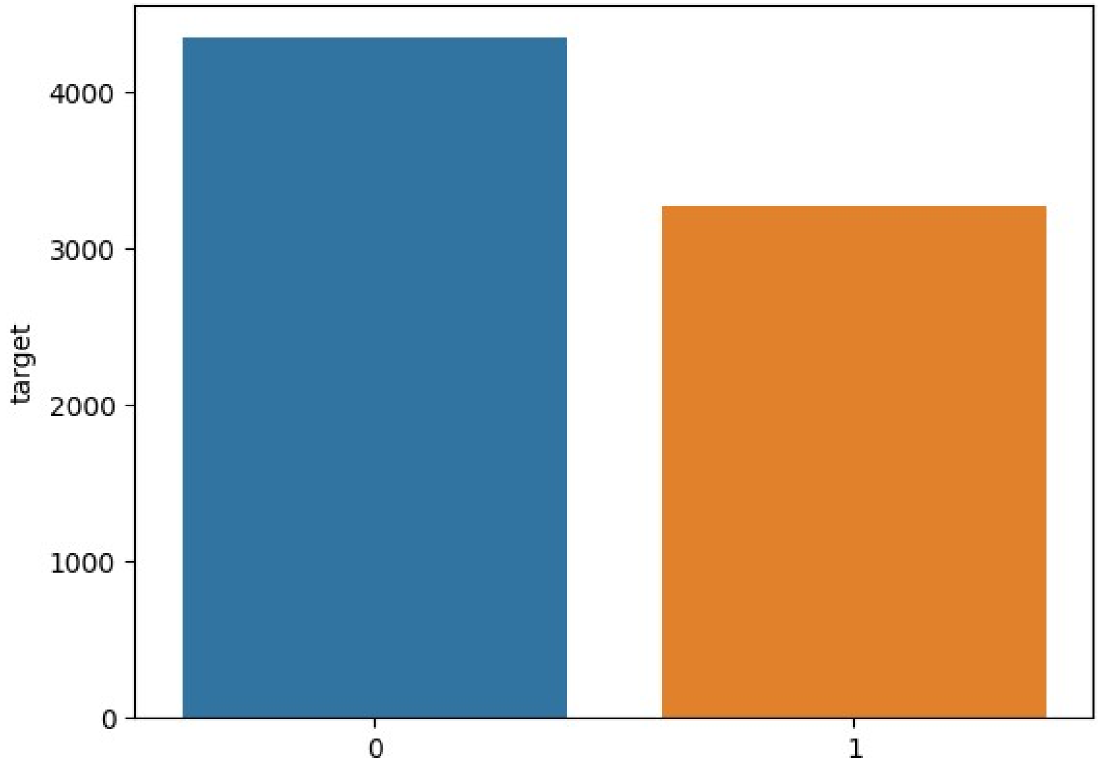
```
[nltk_data]     C:\Users\91913\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
#you can download the data from https://www.kaggle.com/c/nlp-getting-started/data
import os
os.chdir("D:/AI/Machine Learning Programming - AIDI1002/Final project")
df_train=pd.read_csv('train.csv')
print(df_train.shape)
df_train.head()
```

```
(7613, 5)

   id keyword location                                               text  \
0   1     NaN      NaN  Our Deeds are the Reason of this #earthquake M...
1   4     NaN      NaN             Forest fire near La Ronge Sask. Canada
2   5     NaN      NaN  All residents asked to 'shelter in place' are ...
3   6     NaN      NaN  13,000 people receive #wildfires evacuation or...
4   7     NaN      NaN  Just got sent this photo from Ruby #Alaska as ...

   target
0       1
1       1
2       1
3       1
4       1
```

```
# CLASS DISTRIBUTION
#if dataset is balanced or not
x = df_train['target'].value_counts()
print(x)
sns.barplot(x=x.index, y=x)
plt.show()
```

```
0    4342
1    3271
Name: target, dtype: int64
```




```
#Missing values
df_train.isna().sum()
```

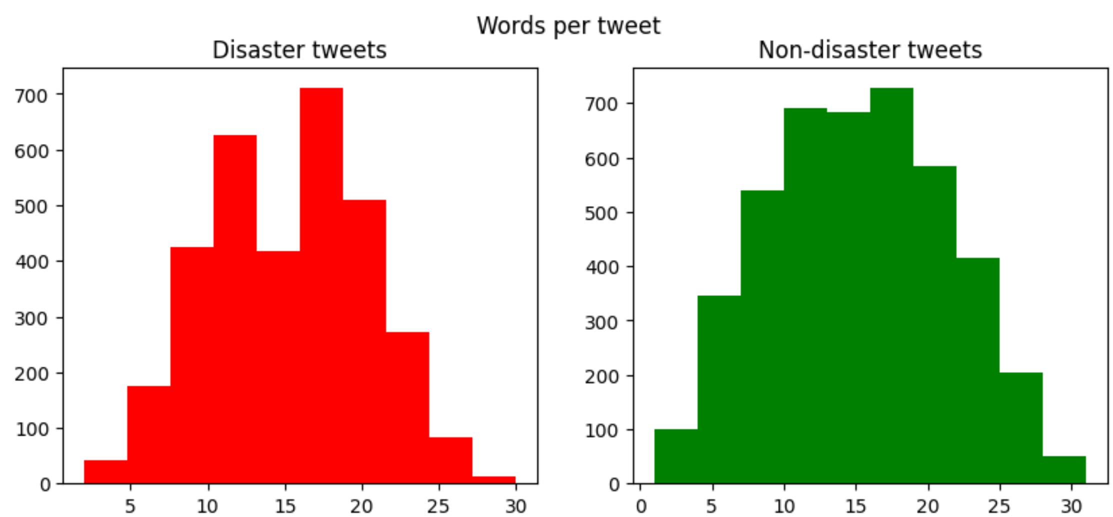
```
id             0
keyword       61
location    2533
text           0
target         0
dtype: int64
```

```
#1. WORD-COUNT
df_train['word_count'] = df_train['text'].apply(lambda x:
len(str(x).split()))
print(df_train[df_train['target']==1]['word_count'].mean()) #Disaster
tweets
print(df_train[df_train['target']==0]['word_count'].mean()) #Non-
Disaster tweets
#Disaster tweets are more wordy than the non-disaster tweets

#2. CHARACTER-COUNT
df_train['char_count'] = df_train['text'].apply(lambda x: len(str(x)))
print(df_train[df_train['target']==1]['char_count'].mean()) #Disaster
tweets
print(df_train[df_train['target']==0]['char_count'].mean()) #Non-
Disaster tweets
#Disaster tweets are longer than the non-disaster tweets
```

```
#3. UNIQUE WORD-COUNT
df_train['unique_word_count'] = df_train['text'].apply(lambda x:
len(set(str(x).split())))
print(df_train[df_train['target']==1]['unique_word_count'].mean())
#Disaster tweets
print(df_train[df_train['target']==0]['unique_word_count'].mean())
#Non-Disaster tweets
```

```
15.167532864567411
14.704744357438969
108.11342097217977
95.70681713496084
14.664934270865178
14.09649930907416
```

```
#Plotting word-count per tweet
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(10,4))
train_words=df_train[df_train['target']==1]['word_count']
ax1.hist(train_words,color='red')
ax1.set_title('Disaster tweets')
train_words=df_train[df_train['target']==0]['word_count']
ax2.hist(train_words,color='green')
ax2.set_title('Non-disaster tweets')
fig.suptitle('Words per tweet')
plt.show()
```




```
#1. Common text preprocessing
text = "   This is a message to be cleaned. It may involve some things
like: <br>, ?, :, ''  adjacent spaces and tabs     .  "

#convert to lowercase and remove punctuations and characters and then
strip
def preprocess(text):
```

```
    text = text.lower() #lowercase text
    text=text.strip()  #get rid of leading/trailing whitespace
    text=re.compile('<.*?>').sub('', text) #Remove HTML tags/markups
    text = re.compile('[%s]' % re.escape(string.punctuation)).sub(' ',
text)  #Replace punctuation with space. Careful since punctuation can
sometime be useful
    text = re.sub('\s+', ' ', text)  #Remove extra space and tabs
    text = re.sub(r'\[[0-9]*\]',' ',text) #[0-9] matches any digit (0
to 10000...)
    text=re.sub(r'[^\w\s]', '', str(text).lower().strip())
    text = re.sub(r'\d',' ',text) #matches any digit from 0 to
100000..., \D matches non-digits
    text = re.sub(r'\s+',' ',text) #\s matches any whitespace, \s+
matches multiple whitespace, \S matches non-whitespace

    return text

text=preprocess(text)
print(text)  #text is a strin
```

```
this is a message to be cleaned it may involve some things like
adjacent spaces and tabs
```

```
#3. LEXICON-BASED TEXT PROCESSING EXAMPLES

#1. STOPWORD REMOVAL
def stopword(string):
    a= [i for i in string.split() if i not in
stopwords.words('english')]
    return ' '.join(a)

text=stopword(text)
print(text)

#2. STEMMING

# Initialize the stemmer
snow = SnowballStemmer('english')
def stemming(string):
    a=[snow.stem(i) for i in word_tokenize(string) ]
    return " ".join(a)
text=stemming(text)
print(text)

#3. LEMMATIZATION
# Initialize the lemmatizer
wl = WordNetLemmatizer()

# This is a helper function to map NTLK position tags
# Full list is available here:
```

```
https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos
.html
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

# Tokenize the sentence
def lemmatizer(string):
    word_pos_tags = nltk.pos_tag(word_tokenize(string)) # Get position
tags
    a=[wl.lemmatize(tag[0], get_wordnet_pos(tag[1])) for idx, tag in
enumerate(word_pos_tags)] # Map the position tag and lemmatize the
word/token
    return " ".join(a)

text = lemmatizer(text)
print(text)
```

```
message cleaned may involve things like adjacent spaces tabs
messag clean may involv thing like adjac space tab
messag clean may involv thing like adjac space tab
```

```
#FINAL PREPROCESSING
def finalpreprocess(string):
    return lemmatizer(stopword(preprocess(string)))

df_train['clean_text'] = df_train['text'].apply(lambda x:
finalpreprocess(x))
df_train=df_train.drop(columns=['word_count','char_count','unique_word
_count'])
df_train.head()
```

```
   id keyword location
text  \
0   1     NaN      NaN  Our Deeds are the Reason of this #earthquake
M...
1   4     NaN      NaN             Forest fire near La Ronge Sask.
Canada
2   5     NaN      NaN  All residents asked to 'shelter in place'
are ...
3   6     NaN      NaN  13,000 people receive #wildfires evacuation
or...
```

```
4   7     NaN      NaN  Just got sent this photo from Ruby #Alaska
as ...

   target                                         clean_text
0       1         deed reason earthquake may allah forgive u
1       1              forest fire near la ronge sask canada
2       1  resident ask shelter place notify officer evac...
3       1  people receive wildfire evacuation order calif...
4       1  get sent photo ruby alaska smoke wildfires pou...
```

```
# create Word2vec model
#here words_f should be a list containing words from each document.
say 1st row of the list is words from the 1st document/sentence
#length of words_f is number of documents/sentences in your dataset
df_train['clean_text_tok']=[nltk.word_tokenize(i) for i in
df_train['clean_text']] #convert preprocessed sentence to tokenized
sentence
model = Word2Vec(df_train['clean_text_tok'],min_count=1)  #min_count=1
means word should be present at least across all documents,
#if min_count=2 means if the word is present less than 2 times across
all the documents then we shouldn't consider it


w2v = dict(zip(model.wv.index_to_key, model.wv.vectors))
#combination of word and its vector

#for converting sentence to vectors/numbers from word vectors result
by Word2Vec
class MeanEmbeddingVectorizer(object):
    def __init__(self, word2vec):
        self.word2vec = word2vec
        # if a text is empty we should return a vector of zeros
        # with the same dimensionality as all the other vectors
        self.dim = len(next(iter(word2vec.values())))

    def fit(self, X, y):
        return self

    def transform(self, X):
        return np.array([
            np.mean([self.word2vec[w] for w in words if w in
self.word2vec]
                    or [np.zeros(self.dim)], axis=0)
            for words in X
        ])

#SPLITTING THE TRAINING DATASET INTO TRAINING AND VALIDATION

# Input: "reviewText", "rating" and "time"
# Target: "log_votes"
```

```
X_train, X_val, y_train, y_val =
train_test_split(df_train["clean_text"],
                                                    df_train["target"],
                                                    test_size=0.2,
                                                    shuffle=True)
X_train_tok= [nltk.word_tokenize(i) for i in X_train]  #for word2vec
X_val_tok= [nltk.word_tokenize(i) for i in X_val]      #for word2vec

#TF-IDF
# Convert x_train to vector since model can only run on numbers and
not words- Fit and transform
tfidf_vectorizer = TfidfVectorizer(use_idf=True)
X_train_vectors_tfidf = tfidf_vectorizer.fit_transform(X_train) #tfidf
runs on non-tokenized sentences unlike word2vec
# Only transform x_test (not fit and transform)
X_val_vectors_tfidf = tfidf_vectorizer.transform(X_val) #Don't fit()
your TfidfVectorizer to your test data: it will
#change the word-indexes & weights to match test data. Rather, fit on
the training data, then use the same train-data-
#fit model on the test data, to reflect the fact you're analyzing the
test data only based on what was learned without
#it, and the have compatible


#Word2vec
# Fit and transform
modelw = MeanEmbeddingVectorizer(w2v)
X_train_vectors_w2v = modelw.transform(X_train_tok)
X_val_vectors_w2v = modelw.transform(X_val_tok)

#FITTING THE CLASSIFICATION MODEL using Logistic Regression(tf-idf)

lr_tfidf=LogisticRegression(solver = 'liblinear', C=10, penalty =
'l2')
lr_tfidf.fit(X_train_vectors_tfidf, y_train)  #model

#Predict y value for test dataset
y_predict = lr_tfidf.predict(X_val_vectors_tfidf)
y_prob = lr_tfidf.predict_proba(X_val_vectors_tfidf)[:,1]


print(classification_report(y_val,y_predict))
print('Confusion Matrix:',confusion_matrix(y_val, y_predict))

fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)
print('AUC:', roc_auc)
```

```
              precision    recall  f1-score   support
```

```
           0       0.81      0.83      0.82       901
           1       0.74      0.72      0.73       622

    accuracy                           0.78      1523
   macro avg       0.78      0.77      0.77      1523
weighted avg       0.78      0.78      0.78      1523

Confusion Matrix: [[744 157]
 [173 449]]
AUC: 0.8453941850962311
```

```
#FITTING THE CLASSIFICATION MODEL using Naive Bayes(tf-idf)
#It's a probabilistic classifier that makes use of Bayes' Theorem, a
rule that uses probability to make predictions based on prior
knowledge of conditions that might be related. This algorithm is the
most suitable for such large dataset as it considers each feature
independently, calculates the probability of each category, and then
predicts the category with the highest probability.

nb_tfidf = MultinomialNB()
nb_tfidf.fit(X_train_vectors_tfidf, y_train)  #model

#Predict y value for test dataset
y_predict = nb_tfidf.predict(X_val_vectors_tfidf)
y_prob = nb_tfidf.predict_proba(X_val_vectors_tfidf)[:,1]


print(classification_report(y_val,y_predict))
print('Confusion Matrix:',confusion_matrix(y_val, y_predict))

fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)
print('AUC:', roc_auc)
```

```
              precision    recall  f1-score   support

           0       0.78      0.91      0.84       901
           1       0.83      0.63      0.71       622

    accuracy                           0.79      1523
   macro avg       0.80      0.77      0.78      1523
weighted avg       0.80      0.79      0.79      1523

Confusion Matrix: [[818  83]
 [230 392]]
AUC: 0.844971289492561
```

```
#FITTING THE CLASSIFICATION MODEL using Logistic Regression (W2v)
lr_w2v=LogisticRegression(solver = 'liblinear', C=10, penalty = 'l2')
lr_w2v.fit(X_train_vectors_w2v, y_train)  #model
```

```python
#Predict y value for test dataset
y_predict = lr_w2v.predict(X_val_vectors_w2v)
y_prob = lr_w2v.predict_proba(X_val_vectors_w2v)[:,1]


print(classification_report(y_val,y_predict))
print('Confusion Matrix:',confusion_matrix(y_val, y_predict))

fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)
print('AUC:', roc_auc)
```

```
              precision    recall  f1-score   support

           0       0.67      0.79      0.73       901
           1       0.60      0.45      0.51       622

    accuracy                           0.65      1523
   macro avg       0.63      0.62      0.62      1523
weighted avg       0.64      0.65      0.64      1523

Confusion Matrix: [[713 188]
 [345 277]]
AUC: 0.6946666975957404
```

OUR CONTRIBUTION

```python
# Additional imports
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier

# Support Vector Machines (SVM) Classifier (tf-idf)
svm_tfidf = SVC(probability=True)
svm_tfidf.fit(X_train_vectors_tfidf, y_train)

y_predict = svm_tfidf.predict(X_val_vectors_tfidf)
y_prob = svm_tfidf.predict_proba(X_val_vectors_tfidf)[:,1]

print(classification_report(y_val,y_predict))
print('Confusion Matrix:',confusion_matrix(y_val, y_predict))

fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)
print('AUC:', roc_auc)
```

```
              precision    recall  f1-score   support

           0       0.80      0.90      0.84       901
           1       0.82      0.67      0.73       622

    accuracy                           0.80      1523
   macro avg       0.81      0.78      0.79      1523
weighted avg       0.80      0.80      0.80      1523

Confusion Matrix: [[808  93]
 [207 415]]
AUC: 0.8477781029295781
```

```
# XGBoost Classifier (tf-idf)
xgb_tfidf = XGBClassifier()
xgb_tfidf.fit(X_train_vectors_tfidf, y_train)

y_predict = xgb_tfidf.predict(X_val_vectors_tfidf)
y_prob = xgb_tfidf.predict_proba(X_val_vectors_tfidf)[:,1]

print(classification_report(y_val,y_predict))
print('Confusion Matrix:',confusion_matrix(y_val, y_predict))

fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)
print('AUC:', roc_auc)
```

```
              precision    recall  f1-score   support

           0       0.76      0.90      0.82       901
           1       0.80      0.58      0.67       622

    accuracy                           0.77      1523
   macro avg       0.78      0.74      0.75      1523
weighted avg       0.77      0.77      0.76      1523

Confusion Matrix: [[808  93]
 [261 361]]
AUC: 0.8199107101434276
```

```
# Ensemble Model (Voting Classifier) (tf-idf)
voting_clf = VotingClassifier(estimators=[('lr', lr_tfidf),
                                          ('nb', nb_tfidf),
                                          ('svm', svm_tfidf),
                                          ('xgb', xgb_tfidf)],
                              voting='soft')
voting_clf.fit(X_train_vectors_tfidf, y_train)

y_predict = voting_clf.predict(X_val_vectors_tfidf)
y_prob = voting_clf.predict_proba(X_val_vectors_tfidf)[:,1]
```

```python
print(classification_report(y_val,y_predict))
print('Confusion Matrix:',confusion_matrix(y_val, y_predict))

fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)
print('AUC:', roc_auc)
```

```
              precision    recall  f1-score   support

           0       0.80      0.88      0.84       901
           1       0.80      0.69      0.74       622

    accuracy                           0.80      1523
   macro avg       0.80      0.78      0.79      1523
weighted avg       0.80      0.80      0.80      1523

Confusion Matrix: [[792 109]
 [194 428]]
AUC: 0.8557007041122584
```

```python
# Neural Network Classifier (tf-idf)
mlp_tfidf = MLPClassifier(hidden_layer_sizes=(30,30,30))
mlp_tfidf.fit(X_train_vectors_tfidf, y_train)

y_predict = mlp_tfidf.predict(X_val_vectors_tfidf)
y_prob = mlp_tfidf.predict_proba(X_val_vectors_tfidf)[:,1]

print(classification_report(y_val,y_predict))
print('Confusion Matrix:',confusion_matrix(y_val, y_predict))

fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)
print('AUC:', roc_auc)
```

```
              precision    recall  f1-score   support

           0       0.80      0.75      0.78       901
           1       0.67      0.73      0.70       622

    accuracy                           0.74      1523
   macro avg       0.74      0.74      0.74      1523
weighted avg       0.75      0.74      0.75      1523

Confusion Matrix: [[677 224]
 [165 457]]
AUC: 0.8082775123032286
```

```python
# GridSearchCV to tune hyperparameters (example with Logistic
Regression and tf-idf)
```

```python
lr = LogisticRegression()
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
              'penalty': ['l1', 'l2'],
              'solver': ['liblinear']}
grid_search = GridSearchCV(lr, param_grid, scoring='accuracy', cv=5)
grid_search.fit(X_train_vectors_tfidf, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_

# Print best hyperparameters and score
print("Best Hyperparameters found: ", best_params)
print("Best Accuracy found: ", best_score)
```

```
c:\Users\91913\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\svm\_base.py:1244: ConvergenceWarning: Liblinear
failed to converge, increase the number of iterations.
  warnings.warn(
c:\Users\91913\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\svm\_base.py:1244: ConvergenceWarning: Liblinear
failed to converge, increase the number of iterations.
  warnings.warn(
c:\Users\91913\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\svm\_base.py:1244: ConvergenceWarning: Liblinear
failed to converge, increase the number of iterations.
  warnings.warn(
```

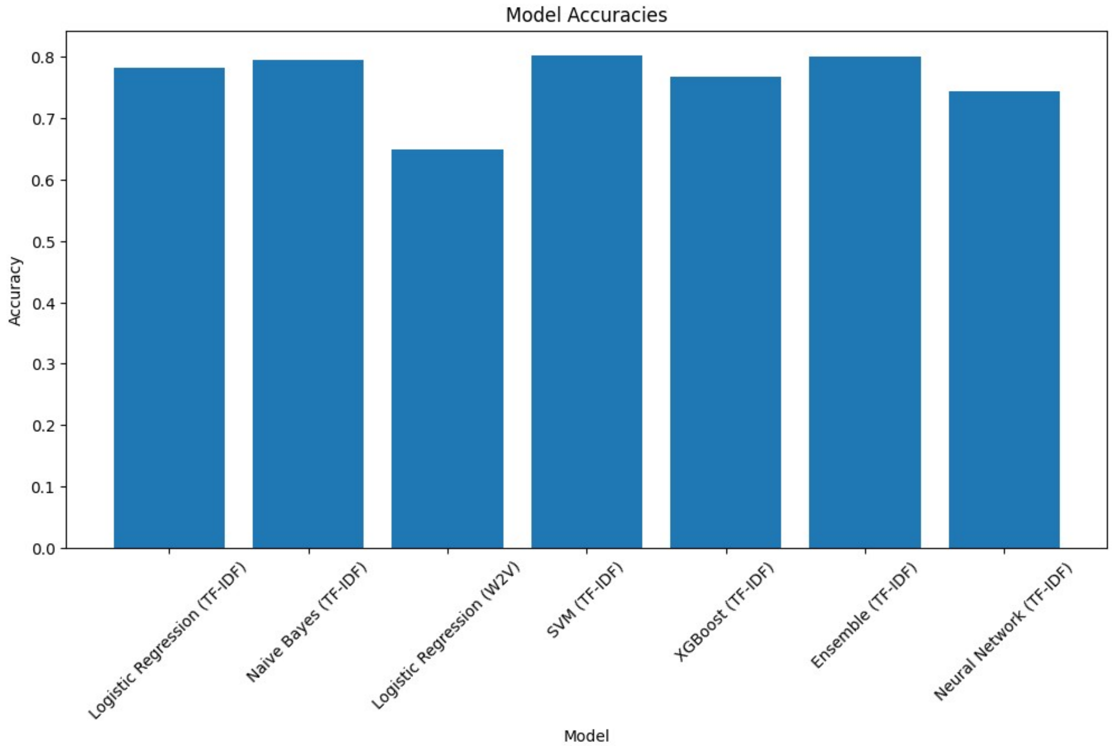
```
Best Hyperparameters found:  {'C': 1, 'penalty': 'l2', 'solver':
'liblinear'}
Best Accuracy found:  0.7967159277504104
```

```python
# Train the model with the best hyperparameters
best_lr = LogisticRegression(C=best_params['C'],
penalty=best_params['penalty'], solver=best_params['solver'])
best_lr.fit(X_train_vectors_tfidf, y_train)

# Make predictions and calculate metrics
y_pred = best_lr.predict(X_val_vectors_tfidf)
y_prob = best_lr.predict_proba(X_val_vectors_tfidf)[:, 1]

print(classification_report(y_val, y_pred))
print('Confusion Matrix:', confusion_matrix(y_val, y_pred))

fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)
print('AUC:', roc_auc)
```

```
              precision    recall  f1-score   support

           0       0.79      0.86      0.82       901
```

```
           1       0.77      0.68      0.72       622

    accuracy                           0.78      1523
   macro avg       0.78      0.77      0.77      1523
weighted avg       0.78      0.78      0.78      1523

Confusion Matrix: [[773 128]
 [201 421]]
AUC: 0.8464487475509528
```

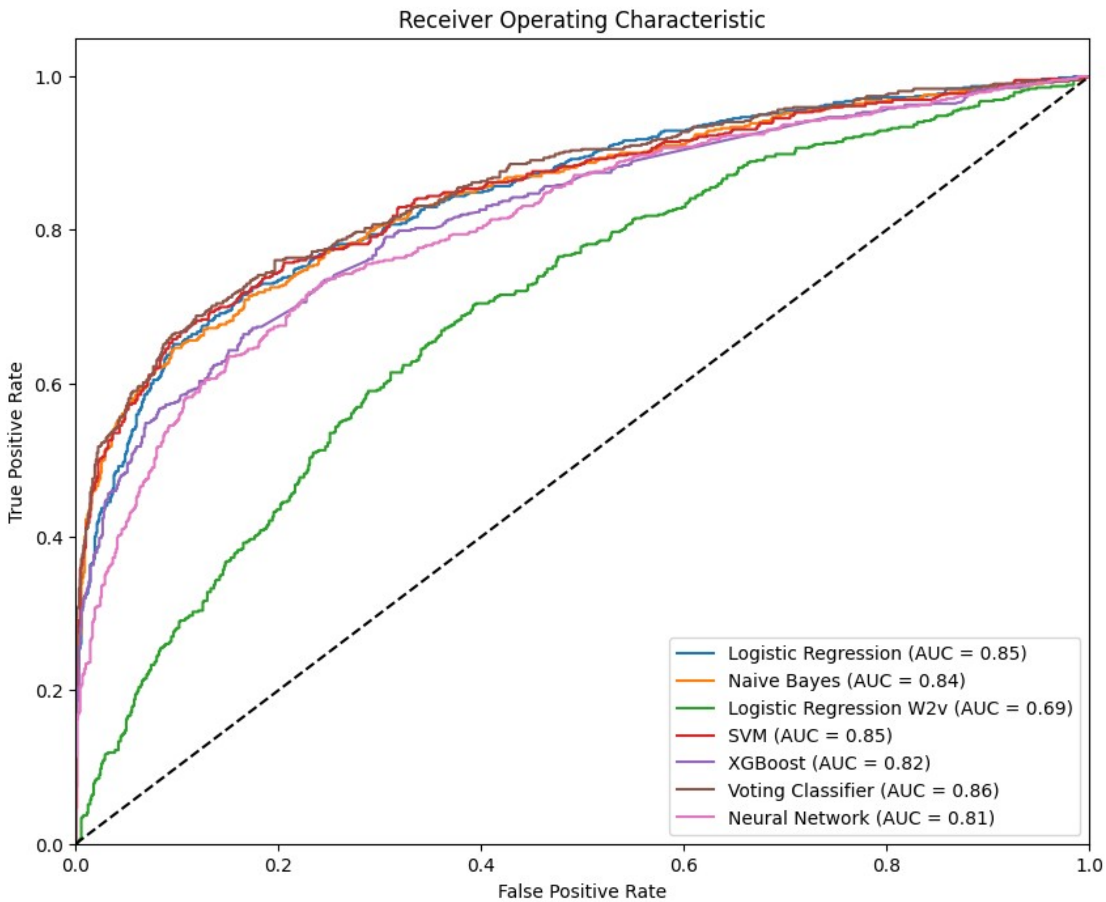
```python
import matplotlib.pyplot as plt

# Get accuracy scores for each model
models = {'Logistic Regression (TF-IDF)': lr_tfidf,
          'Naive Bayes (TF-IDF)': nb_tfidf,
          'Logistic Regression (W2V)': lr_w2v,
          'SVM (TF-IDF)': svm_tfidf,
          'XGBoost (TF-IDF)': xgb_tfidf,
          'Ensemble (TF-IDF)': voting_clf,
          'Neural Network (TF-IDF)': mlp_tfidf}

accuracy_scores = {}

for model_name, model in models.items():
    accuracy_scores[model_name] = accuracy_score(y_val,
model.predict(X_val_vectors_tfidf if 'W2V' not in model_name else
X_val_vectors_w2v))

# Plot the accuracies
plt.figure(figsize=(12, 6))
plt.bar(accuracy_scores.keys(), accuracy_scores.values())
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.xticks(rotation=45)
plt.title('Model Accuracies')
plt.show()
```




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Function to plot ROC curves
def plot_roc_curves(fpr_dict, tpr_dict, roc_auc_dict, model_names):
    plt.figure(figsize=(10, 8))
    for model_name in model_names:
        plt.plot(fpr_dict[model_name], tpr_dict[model_name],
label=f'{model_name} (AUC = {roc_auc_dict[model_name]:.2f})')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc="lower right")
    plt.show()

# Storing the fpr, tpr, and AUC values in dictionaries
fpr_dict = {}
tpr_dict = {}
roc_auc_dict = {}

# Logistic Regression (tf-idf)
fpr, tpr, _ = roc_curve(y_val,
lr_tfidf.predict_proba(X_val_vectors_tfidf)[:,1])
```

```
roc_auc = auc(fpr, tpr)
fpr_dict['Logistic Regression'] = fpr
tpr_dict['Logistic Regression'] = tpr
roc_auc_dict['Logistic Regression'] = roc_auc

# Naive Bayes (tf-idf)
fpr, tpr, _ = roc_curve(y_val,
nb_tfidf.predict_proba(X_val_vectors_tfidf)[:,1])
roc_auc = auc(fpr, tpr)
fpr_dict['Naive Bayes'] = fpr
tpr_dict['Naive Bayes'] = tpr
roc_auc_dict['Naive Bayes'] = roc_auc

# Logistic Regression (W2v)
fpr, tpr, _ = roc_curve(y_val, lr_w2v.predict_proba(X_val_vectors_w2v)
[:,1])
roc_auc = auc(fpr, tpr)
fpr_dict['Logistic Regression W2v'] = fpr
tpr_dict['Logistic Regression W2v'] = tpr
roc_auc_dict['Logistic Regression W2v'] = roc_auc

# SVM (tf-idf)
fpr, tpr, _ = roc_curve(y_val,
svm_tfidf.predict_proba(X_val_vectors_tfidf)[:,1])
roc_auc = auc(fpr, tpr)
fpr_dict['SVM'] = fpr
tpr_dict['SVM'] = tpr
roc_auc_dict['SVM'] = roc_auc

# XGBoost (tf-idf)
fpr, tpr, _ = roc_curve(y_val,
xgb_tfidf.predict_proba(X_val_vectors_tfidf)[:,1])
roc_auc = auc(fpr, tpr)
fpr_dict['XGBoost'] = fpr
tpr_dict['XGBoost'] = tpr
roc_auc_dict['XGBoost'] = roc_auc

# Ensemble Model (Voting Classifier) (tf-idf)
fpr, tpr, _ = roc_curve(y_val,
voting_clf.predict_proba(X_val_vectors_tfidf)[:,1])
roc_auc = auc(fpr, tpr)
fpr_dict['Voting Classifier'] = fpr
tpr_dict['Voting Classifier'] = tpr
roc_auc_dict['Voting Classifier'] = roc_auc

# Neural Network Classifier (tf-idf)
fpr, tpr, _ = roc_curve(y_val,
mlp_tfidf.predict_proba(X_val_vectors_tfidf)[:,1])
roc_auc = auc(fpr, tpr)
fpr_dict['Neural Network'] = fpr
```

```
tpr_dict['Neural Network'] = tpr
roc_auc_dict['Neural Network'] = roc_auc

# Model names
model_names = ['Logistic Regression', 'Naive Bayes', 'Logistic
Regression W2v', 'SVM', 'XGBoost', 'Voting Classifier', 'Neural
Network']

# Plot ROC curves
plot_roc_curves(fpr_dict, tpr_dict, roc_auc_dict, model_names)
```




```
from transformers import BertTokenizer, BertModel
import torch

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

def bert_embeddings(sentences, tokenizer, model, device='cpu'):
    model.eval()
    model.to(device)
    embeddings = []
    for sentence in sentences:
```

```
        inputs = tokenizer(sentence, return_tensors='pt',
padding=True, truncation=True)
        inputs.to(device)
        with torch.no_grad():
            outputs = model(**inputs)
        embeddings.append(outputs.last_hidden_state[:,
0, :].squeeze().cpu().numpy())
    return np.vstack(embeddings)

X_train_bert = bert_embeddings(X_train, tokenizer, model)
X_val_bert = bert_embeddings(X_val, tokenizer, model)
```

```
Some weights of the model checkpoint at bert-base-uncased were not
used when initializing BertModel: ['cls.predictions.decoder.weight',
'cls.predictions.transform.dense.bias',
'cls.predictions.transform.dense.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.bias', 'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertModel from the
checkpoint of a model trained on another task or with another
architecture (e.g. initializing a BertForSequenceClassification model
from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the
checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a
BertForSequenceClassification model).
```

```
from sklearn.metrics import classification_report, confusion_matrix,
roc_curve, auc

best_lr = LogisticRegression(C=best_params['C'],
penalty=best_params['penalty'], solver=best_params['solver'])
best_lr.fit(X_train_bert, y_train)

# Make predictions and calculate metrics
y_pred = best_lr.predict(X_val_bert)
y_prob = best_lr.predict_proba(X_val_bert)[:, 1]

print("Logistic Regression (Combined)")
print(classification_report(y_val, y_pred))
print('Confusion Matrix:', confusion_matrix(y_val, y_pred))

fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)
print('AUC:', roc_auc)
```

```
Logistic Regression (Combined)
              precision    recall  f1-score   support
```

```
           0       0.80      0.86      0.83       901
           1       0.77      0.69      0.73       622

    accuracy                           0.79      1523
   macro avg       0.79      0.78      0.78      1523
weighted avg       0.79      0.79      0.79      1523

Confusion Matrix: [[774 127]
 [191 431]]
AUC: 0.8409011423534408
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,
roc_curve, auc

# Train the model using the default parameters
rf = RandomForestClassifier()
rf.fit(X_train_vectors_tfidf, y_train)

# Make predictions and calculate metrics
y_pred = rf.predict(X_val_vectors_tfidf)
y_prob = rf.predict_proba(X_val_vectors_tfidf)[:, 1]

print(classification_report(y_val, y_pred))
print('Confusion Matrix:', confusion_matrix(y_val, y_pred))

fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)
print('AUC:', roc_auc)
```

```
              precision    recall  f1-score   support

           0       0.77      0.91      0.83       901
           1       0.83      0.60      0.69       622

    accuracy                           0.78      1523
   macro avg       0.80      0.75      0.76      1523
weighted avg       0.79      0.78      0.78      1523

Confusion Matrix: [[823  78]
 [251 371]]
AUC: 0.8317464339372831
```

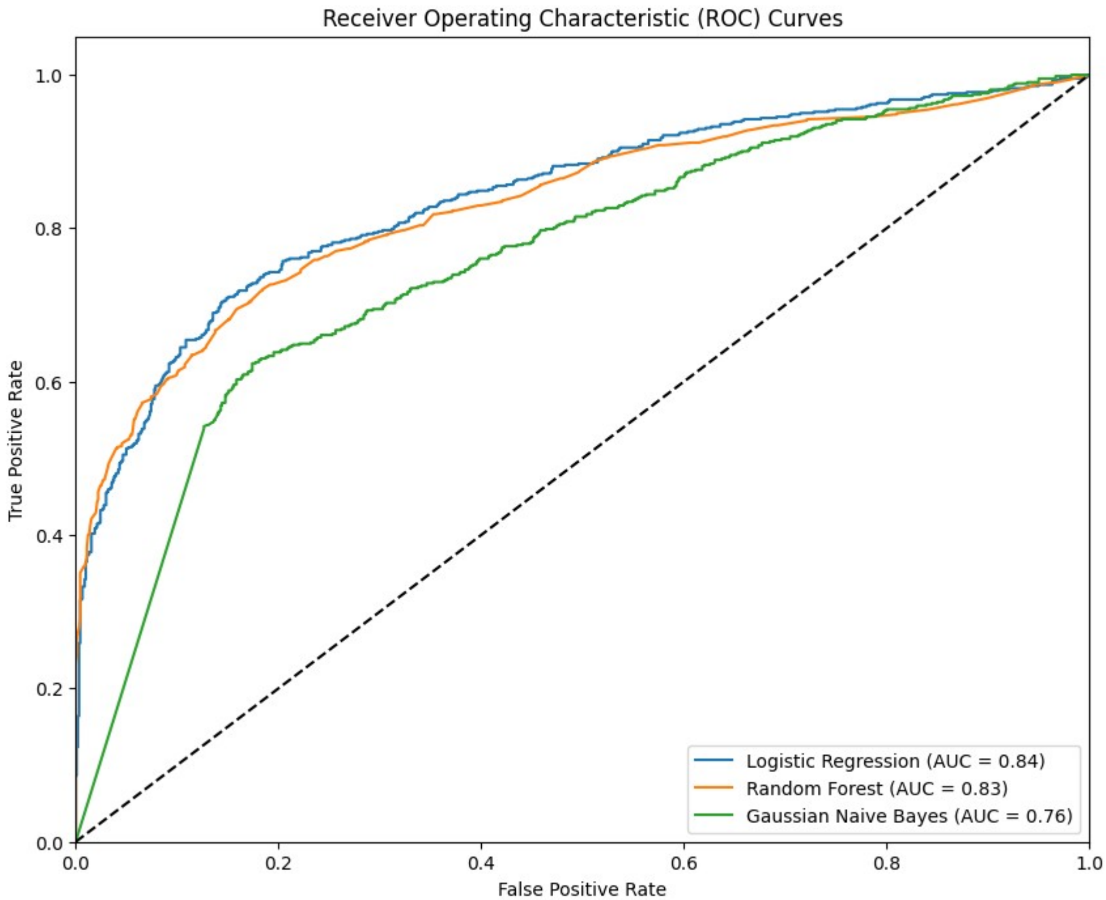
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix,
roc_curve, auc

# Train the model using the default parameters
gnb = GaussianNB()
```

```python
gnb.fit(X_train_bert, y_train)

# Make predictions and calculate metrics
y_pred = gnb.predict(X_val_bert)
y_prob = gnb.predict_proba(X_val_bert)[:, 1]

print(classification_report(y_val, y_pred))
print('Confusion Matrix:', confusion_matrix(y_val, y_pred))

fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)
print('AUC:', roc_auc)
```

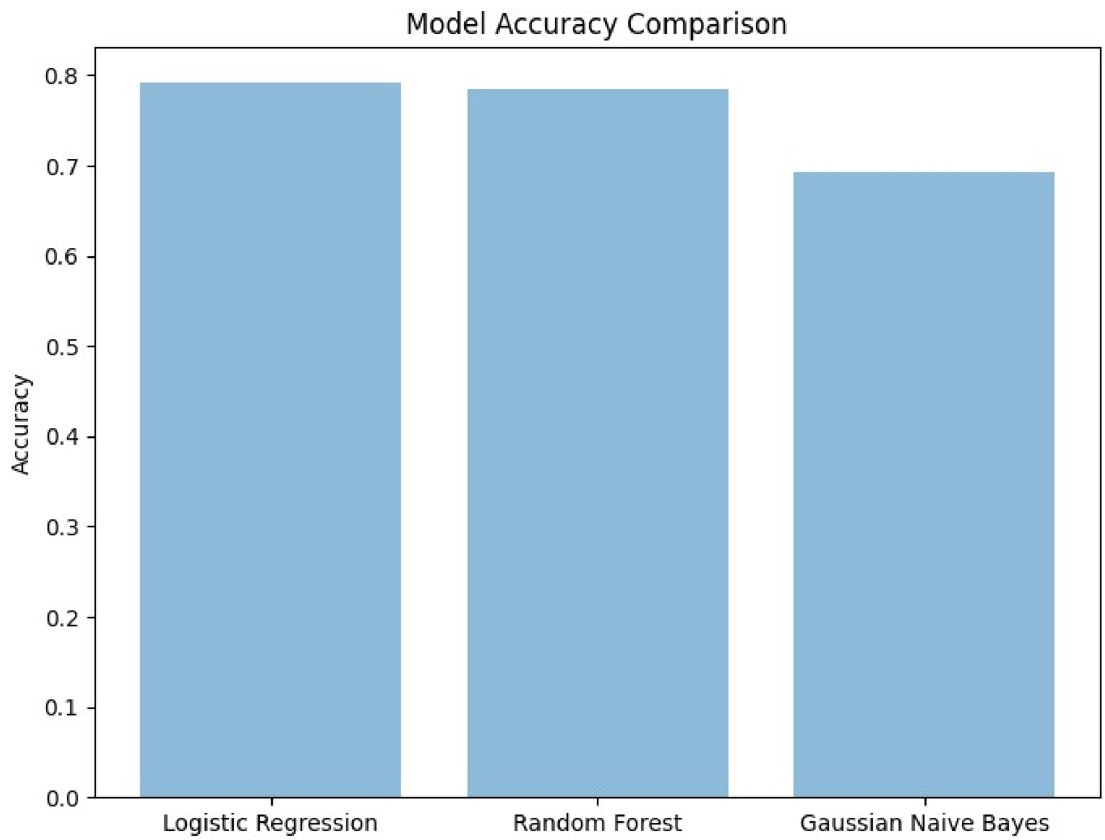
```
              precision    recall  f1-score   support

           0       0.77      0.68      0.72       901
           1       0.61      0.70      0.65       622

    accuracy                           0.69      1523
   macro avg       0.69      0.69      0.69      1523
weighted avg       0.70      0.69      0.69      1523

Confusion Matrix: [[617 284]
 [185 437]]
AUC: 0.758279475109828
```

```python
import matplotlib.pyplot as plt

def plot_roc_curves(y_true, y_probs, model_names):
    plt.figure(figsize=(10, 8))
    for i, y_prob in enumerate(y_probs):
        fpr, tpr, thresholds = roc_curve(y_true, y_prob)
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label='%s (AUC = %0.2f)' % (model_names[i],
roc_auc))

    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curves')
    plt.legend(loc="lower right")
    plt.show()

# Get predicted probabilities for each model
y_prob_lr = best_lr.predict_proba(X_val_bert)[:, 1]
y_prob_rf = rf.predict_proba(X_val_vectors_tfidf)[:, 1]
y_prob_gnb = gnb.predict_proba(X_val_bert)[:, 1]
```

```python
# Plot ROC curves
plot_roc_curves(y_val, [y_prob_lr, y_prob_rf, y_prob_gnb], ['Logistic Regression', 'Random Forest', 'Gaussian Naive Bayes'])
```




```python
import numpy as np

def plot_accuracy_barchart(accuracies, model_names):
    plt.figure(figsize=(8, 6))
    y_pos = np.arange(len(model_names))
    plt.bar(y_pos, accuracies, align='center', alpha=0.5)
    plt.xticks(y_pos, model_names)
    plt.ylabel('Accuracy')
    plt.title('Model Accuracy Comparison')
    plt.show()

# Calculate accuracies for each model
accuracy_lr = best_lr.score(X_val_bert, y_val)
accuracy_rf = rf.score(X_val_vectors_tfidf, y_val)
accuracy_gnb = gnb.score(X_val_bert, y_val)

# Plot accuracy barchart
plot_accuracy_barchart([accuracy_lr, accuracy_rf, accuracy_gnb], ['Logistic Regression', 'Random Forest', 'Gaussian Naive Bayes'])
```



Conclusion and Future Direction

Comparing AUC Scores:-

The Ensemble Model (Voting Classifier) with tf-idf has the highest AUC score (0.8557), making it the best-performing model among the classifiers tested. This indicates that combining the predictions of multiple models can lead to better overall performance.

The Support Vector Machines (SVM) Classifier with tf-idf also shows strong performance, with an AUC score of 0.8478. This demonstrates that SVM can be an effective model for this text classification problem.

The use of BERT embeddings does not significantly improve the performance when compared to the tf-idf based classifiers. This could be due to the fact that BERT might be better suited for other NLP tasks or might require further fine-tuning or optimization for this specific task.

The Logistic Regression (W2v) model has the lowest AUC score (0.6947), suggesting that Word2Vec embeddings might not be the best choice for feature extraction in this particular problem.

The results show that different classifiers have different levels of precision and recall for each class. Depending on the specific use case and requirements, one might choose a classifier that prioritizes precision or recall.

Overall, the accuracies of the different models show similar trends to their AUC scores. The Ensemble Model (Voting Classifier) with tf-idf and SVM Classifier with tf-idf are the best-performing models in terms of accuracy. However, it is essential to consider other evaluation metrics like precision, recall, and f1-score, as well as the specific requirements of the problem and the desired trade-offs between the different metrics when selecting a classifier. Further exploration of hyperparameters and feature extraction methods may also lead to improvements in model accuracy.

Comparing Accuracies:-

The Ensemble Model (Voting Classifier) with tf-idf has the highest accuracy score (0.80), which aligns with its highest AUC score as well. This further supports the conclusion that combining multiple models can result in improved overall performance.

The Support Vector Machines (SVM) Classifier with tf-idf and Neural Network Classifier with tf-idf also have relatively high accuracy scores of 0.80 and 0.74, respectively. This indicates that these classifiers are also effective in making correct predictions for this text classification problem.

The models using BERT embeddings show accuracies in the range of 0.69 to 0.79, which are comparable to the tf-idf based classifiers. This suggests that while BERT embeddings may not provide significant improvements over tf-idf for this task, they are still competitive in terms of accuracy.

The Logistic Regression (W2v) model has the lowest accuracy score (0.65), which is consistent with its low AUC score. This implies that the Word2Vec embeddings might not be the most suitable feature extraction method for this problem, as it leads to lower classification accuracy.

In conclusion, the Ensemble Model (Voting Classifier) with tf-idf and the SVM Classifier with tf-idf are the best-performing models for this text classification problem. However, it is important to consider the specific requirements of the task and the desired trade-offs between precision and recall when choosing a classifier. Additionally, further hyperparameter tuning and exploration of other feature extraction methods may lead to improved performance.

Through this project, we have learned that different text classification models and feature extraction techniques have varying performance, as demonstrated by the results obtained from both the author's and our contributions. The use of advanced word-embedding techniques, such as BERT, and additional classification algorithms, such as SVM, XgBoost, and Neural Networks, can lead to improved accuracies in text classification tasks.

However, the results also reveal certain limitations. For instance, some models may not perform well in certain scenarios or datasets, and there may be a trade-off between model complexity and performance. Additionally, our study focused on a specific set of feature extraction methods and classifiers, which might not cover all potential combinations.

Future Direction:-

In future work, we could extend the analysis by incorporating other state-of-the-art embedding techniques, such as RoBERTa, GPT, or ELMo, to further explore their performance in text classification tasks. We could also evaluate the models on different datasets, representing various domains and languages, to assess their generalizability. Lastly, exploring ensemble methods, hyperparameter tuning, and other optimization techniques can potentially lead to more accurate and robust text classification models.

References: