SOCIAL MEDIA REVIEWS ANALYSIS

DONE BY

PRANAV SURESH BABU – 2884103

BARATH RAJ KUMARAVEL – 2886170

MANISHA CHEPURI – 2889626

RITHISH REDDY LATTUPALLY - 2887666

TABLE OF CONTENTS

GOAL OF THE PROJECT

- The project focuses on leveraging a dataset of 1.5 million social media reviews divided into 500,000 for training and 1 million for test reviews, to analyze sentiment on social media.
- Huge amount of data is posted on the social media platforms on a daily basis. The sentimental analysis is a process understanding opinions, feelings, and thoughts of people about a given subject. Its advantages being scalability, real-time analysis and consistent criteria.
- To ensure the quality of the dataset, preprocessing techniques such as noise removal will be applied, eliminating special characters, mentions, URLs and HTML tags.
- A key focus on enhancing feature representation of the model by mapping contractions found in reviews to their formal writing equivalents.
- The overarching objective of the project is to elevate sentiment analysis results by minimizing errors and ensuring a reliable and consistent portrayal of user sentiment.

DATASET OVERVIEW

1.5 Million Social Media reviews

- 500,000 for training
- 1 million for test reviews

Columns info and size for test and train

```
[24] df_train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 222607 entries, 0 to 222606
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   review  222607 non-null  object
 1   target  222606 non-null  float64
dtypes: float64(1), object(1)
memory usage: 3.4+ MB
```

```
[25] df_test.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 236272 entries, 0 to 236271
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   review  236272 non-null  object
 1   target  236271 non-null  float64
dtypes: float64(1), object(1)
```

SIZE OF THE DATASET FOR TRAINING AND TESTING

```
[20] df_train.shape
```

```
(222607, 2)
```

```
[21] df_test.shape
```

```
(236272, 2)
```

FEATURE DESCRIPTION

- In this dataset, reviews and target are two columns with target = 0 denoting a negative review and target = 1 denoting a positive review.

```
[22] df_train['target'].value_counts()

1.0    112982
0.0    109624
Name: target, dtype: int64
```

```
[23] df_test['target'].value_counts()

1.0    118520
0.0    117751
Name: target, dtype: int64
```

FEATURE EXTRACTION

Approach:

- Experimentation with three-word representations
- Utilization of N-gram models for feature extraction

Feature Vectorization:

- TdfVectorizer function in Scikit-learn library employed
- Conversion of reviews into feature vectors

TF-IDF Method:

- Term Frequency-Inverse Document Frequency technique
- Assigns unique indices to words, determining feature indexes
- Values set based on word frequency and count proportionately adjusted

Uni-gram, Bi-gram, Tri-gram Employment:

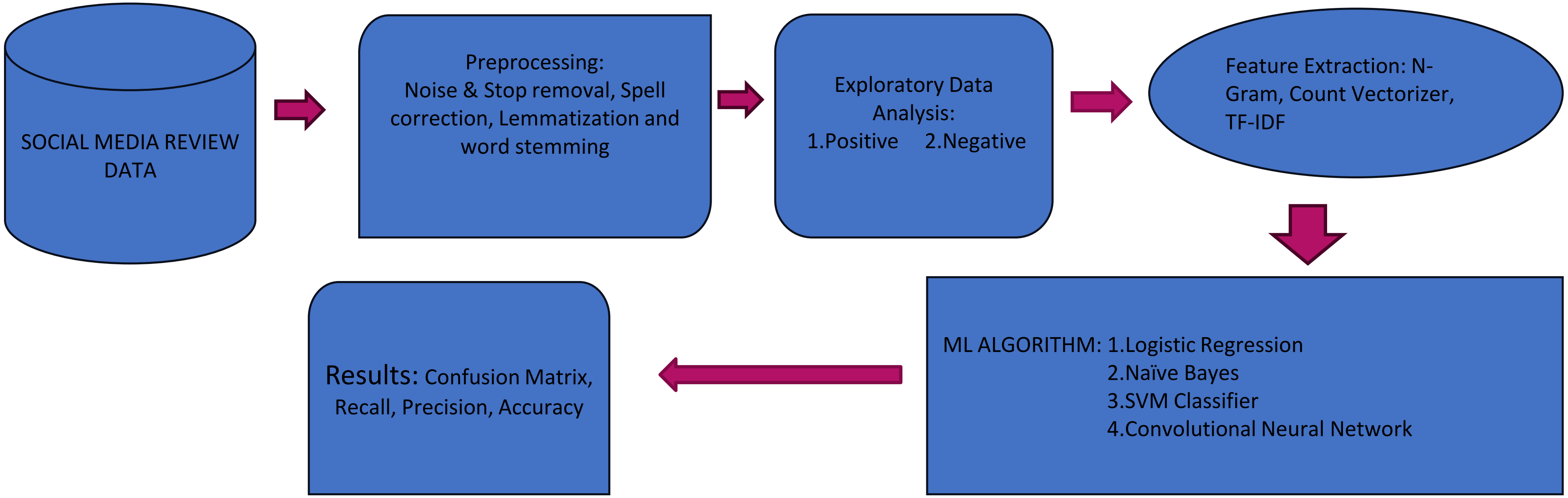
- Analysis showcasing the effectiveness of using word sequences in bi- and tri-grams over unigrams

PLATFORMS AND TOOLS USED

- **PLATFORM** : We will be using google colab, a cloud based platform that allows to write and run python code.
- **TOOLS** :

```
[17] import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from wordcloud import WordCloud
     from sklearn.model_selection import train_test_split
     from textblob import TextBlob
     from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
     from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
     from sklearn.pipeline import Pipeline
     from sklearn.pipeline import make_pipeline
     from sklearn.metrics import roc_curve, auc
     from sklearn.svm import LinearSVC
     from sklearn.linear_model import LogisticRegression
     from sklearn.svm import SVC
     from sklearn.decomposition import PCA
     from sklearn.decomposition import TruncatedSVD
     from time import time
```

FLOWCHART


SOCIAL MEDIA REVIEW DATA
Preprocessing:
Noise & Stop removal, Spell correction, Lemmatization and word stemming
Exploratory Data Analysis:
1.Positive 2.Negative
Feature Extraction: N-Gram, Count Vectorizer, TF-IDF
ML ALGORITHM: 1.Logistic Regression
2.Naïve Bayes
3.SVM Classifier
4.Convolutional Neural Network
Results: Confusion Matrix, Recall, Precision, Accuracy

ANALYSIS AND VISUALIZATION OF THE DATASET

- Taking a sample fraction of the dataset

```
[26] #Taking a sample data
     df_train.columns = ['review','target']
     fraction_to_sample = 0.177
     df_train = df_train.sample(frac=fraction_to_sample)
     df_train.shape

     (39401, 2)
```

```
X_train, X_validation, y_train, y_validation = train_test_split(df_train['review'], df_train['target'], test_size = 0.02, random_state = 45)

print("training data has {0} entries. {1:.2f}% positive and {2:.2f}% negative".format(len(y_train),
                                                                                      len(y_train[y_train==1])/len(y_train)*100,
                                                                                      len(y_train[y_train==0])/len(y_train)*100))
print("validation data has {0} entries. {1:.2f}% positive and {2:.2f}% negative".format(len(y_validation),
                                                                                        len(y_validation[y_validation==1])/len(y_validation)*100,
                                                                                        len(y_validation[y_validation==0])/len(y_validation)*100))

training data has 38612 entries. 50.77% positive and 49.22% negative
validation data has 789 entries. 51.33% positive and 48.67% negative
```

SENTIMENT POLARITY -method returns a value between 0 and 1, where negative values(0) represent negative sentiment, positive values(1) represent positive sentiment

```python
tb_sentiment = [TextBlob(text).sentiment.polarity for text in X_validation]
tb_sentiment_binary = [0 if x < 0 else 1 for x in tb_sentiment]
print(tb_sentiment_binary)
```

```
[1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1,
```

ACCURACY OF THE SAMPLE DATA

```
[ ] print("accuracy = {:.2f}".format(accuracy_score(y_validation, tb_sentiment_binary)))
    print(classification_report(y_validation, tb_sentiment_binary))
```

```
accuracy = 0.71
              precision    recall  f1-score   support

         0.0       0.92      0.45      0.60       427
         1.0       0.64      0.96      0.77       433

    accuracy                           0.71       860
   macro avg       0.78      0.71      0.69       860
weighted avg       0.78      0.71      0.69       860
```

```
[ ] cm = confusion_matrix(y_validation, tb_sentiment_binary)
    cm
```

```
array([[192, 235],
       [ 17, 416]])
```

NLP PREPROCESSING

- STOP WORD REMOVAL – Commonly used in preprocessing steps across different NLP applications. Main idea is to remove the words that occur commonly across all documents. Some examples of stop words are 'a', 'the','is', 'are'.

```
df_word_freq = pd.read_csv('word_frequency.csv')
df_word_freq.columns =['word','negative','positive','total']
```

```
[ ] df_word_freq.head(20)
```

	word	negative	positive	total
0	to	313164	252567	565731
1	the	257870	266013	523883
2	not	238226	103119	341345
3	my	190845	125979	316824
4	it	157482	147804	305286
5	and	153972	149649	303621

```
stop_words = list(df_word_freq.head(40).word)
del stop_words[3]
stop_words
```

```
['to',
 'the',
 'not',
 'it',
 'and',
 'you',
 'is',
 'in',
 'for',
 'of',
 'on',
 'that',
 'me',
 'have',
 'so',
 'do',
 'but',
 'just',
 'with',
 'be',
 'at',
 'can',
 'was',
 'this',
 'now',
 'good',
 'up',
 'day',
 'all',
 'out',
```

- **Lemmatizing** : The goal of lemmatizing is to reduce a word to its root form. For example the verb running could be identified as run.

```
from nltk.stem import WordNetLemmatizer
```

```
lematize = WordNetLemmatizer()
```

```
import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

```
df_train_1 = df_train.copy()
df_train_1['review'] = [' '.join([lematize.lemmatize(word) for word in text.split()]) for text in df_train_1['review'].tolist()]
X_train, X_validation, y_train, y_validation = train_test_split(df_train['review'], df_train['target'], test_size = 0.02, random_state = 45)
```

- **STEMMING –** The process of removing affixes from a word so that we are left with the stem of that word called stemming. For example if we consider the words runs, running and run all convert into the word ‘run’ after stemming is implemented for them.

```
[ ] from nltk.stem import PorterStemmer
```

```
[ ] stemmer = PorterStemmer()
```

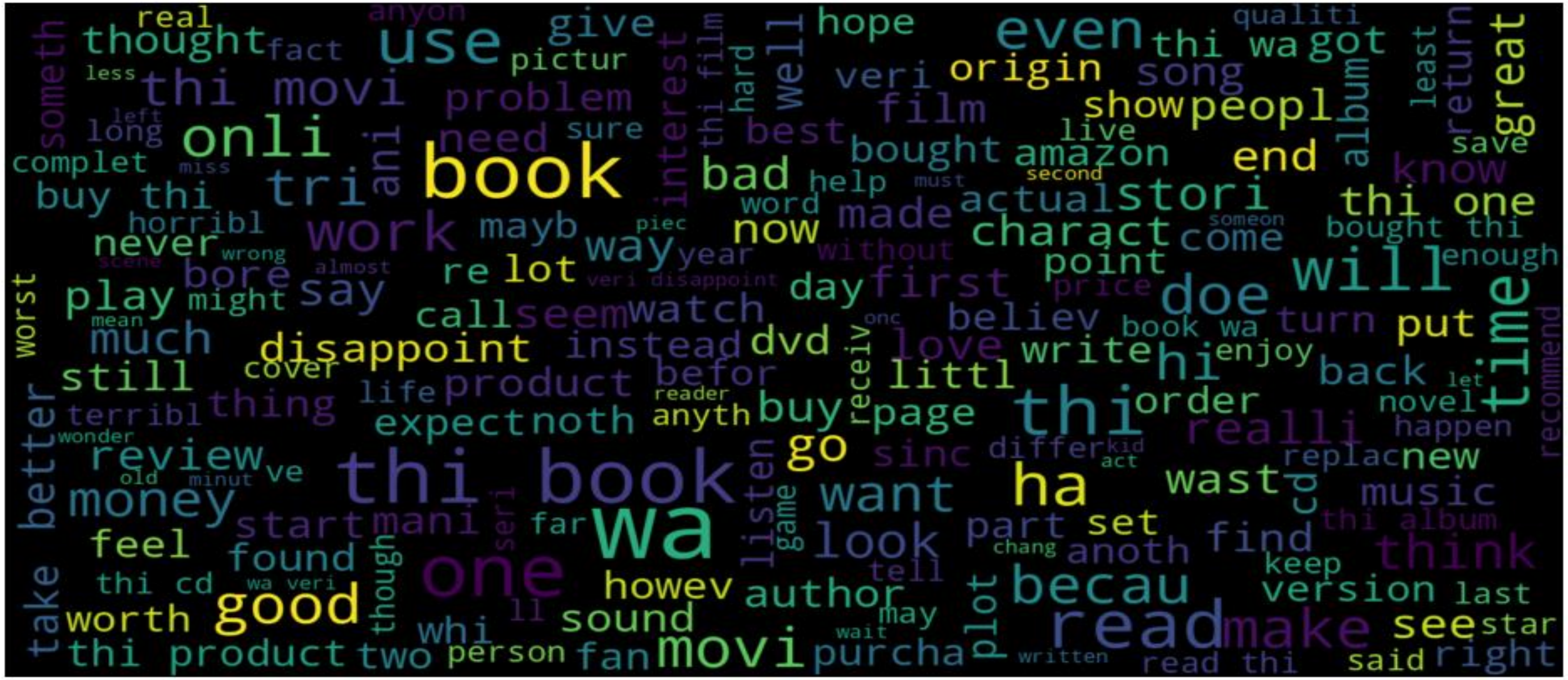
```
[ ] df_train_1 = df_train.copy()
    df_train_1['review'] = [' '.join([stemmer.stem(word) for word in text.split()]) for text in df_train_1['review'].tolist()]
    X_train, X_validation, y_train, y_validation = train_test_split(df_train['review'], df_train['target'], test_size = 0.02, random_state = 45)
```

POST NLP PREPROCESSING

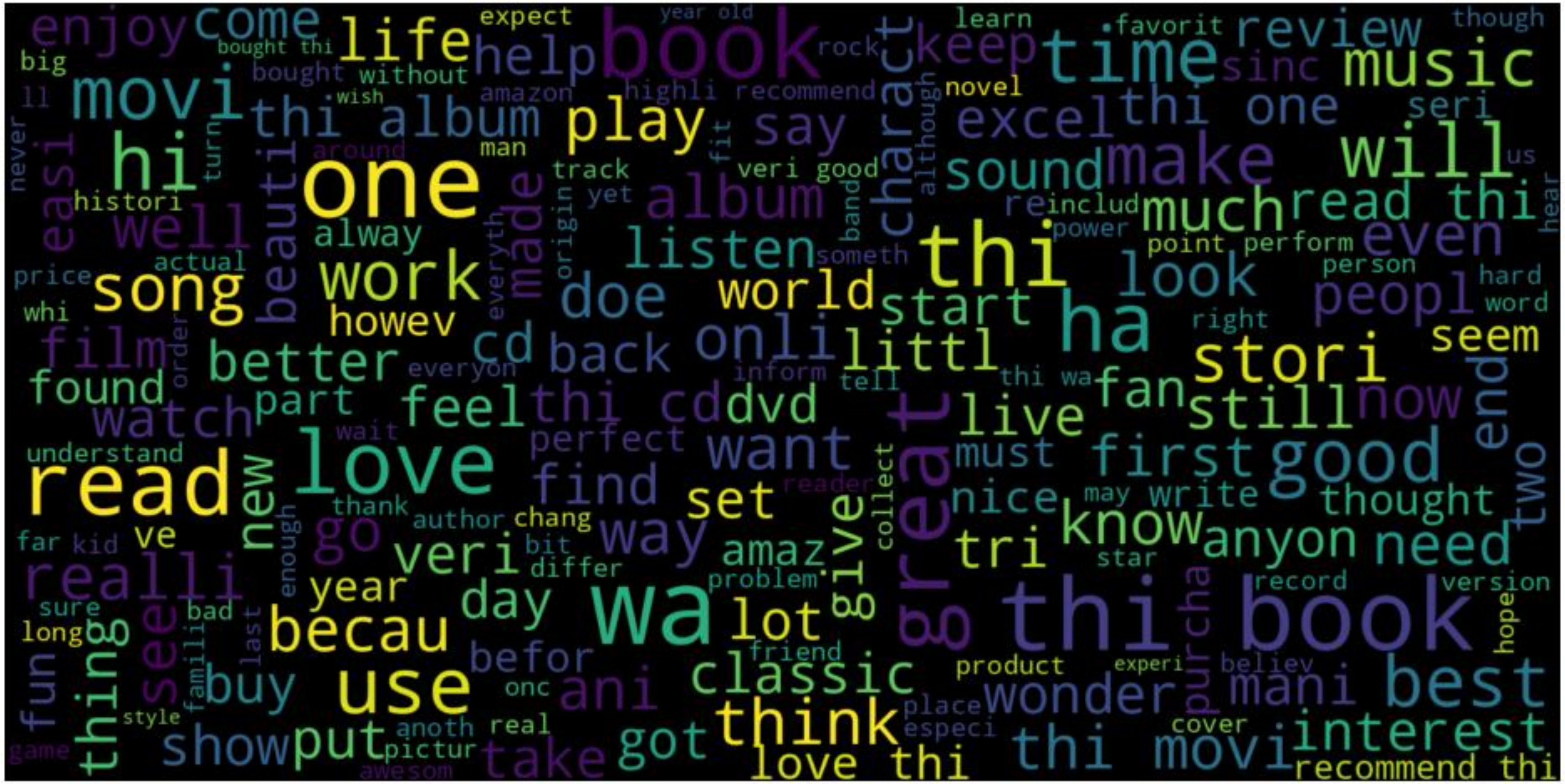
```
[ ] df_train_1.head(10)
```

	review	target
90841	not even close did anyon review here other tha...	0.0
151582	disappoint as dog toy got thi product becaus i...	0.0
59842	good text horribl diagram the actual text of t...	0.0
220212	not hi best it seem to me that the work of kos...	1.0
111957	limit use so far have been pleas with the poul...	0.0
168920	leav thi one alon am celibidach ophil especi w...	0.0
37982	great fantasi travel into excit world excit an...	1.0
132029	still abl to make good disc at thi point thin ...	1.0
166435	veri handi for forget kid thi product is veri ...	1.0
137448	view from the cherri tree there wa boy name ro...	1.0

NEGATIVE WORDS



POSITIVE WORDS

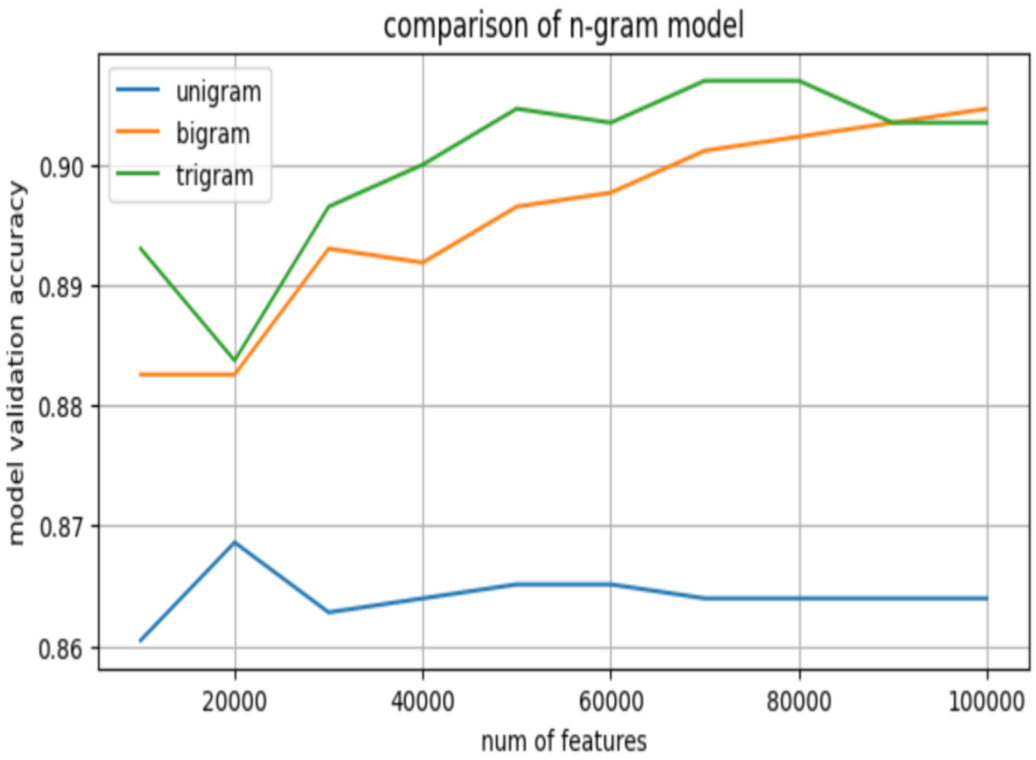


EXPERIMENTATION WITH N-GRAMS

- **Uni–gram** – A 1-gram is a single word sequence of words like ‘please’ or ‘turn’
- **Bi-gram** – A 2 gram is a two word sequence of words like ‘please turn’
- **Tri-gram** – A 3 gram is a three word sequence of words like ‘please turn your’
- We can find out the comparison of the 3 n gram model by selecting the range (1000 to 10,000 features) by using matplotlib visualization tool.

RESULTS OF N-GRAM

```
[ ] cols = ['n_features', 'val_acc', 'val_auc', 'time']
    result_tfidf_unigram = pd.DataFrame(result_tfidf_unigram, columns=cols)
    result_tfidf_bigram = pd.DataFrame(result_tfidf_bigram, columns=cols)
    result_tfidf_trigram = pd.DataFrame(result_tfidf_trigram, columns=cols)
    plt.figure(figsize=(8,4))
    plt.plot(result_tfidf_unigram.n_features, result_tfidf_unigram.val_acc, label='unigram')
    plt.plot(result_tfidf_bigram.n_features, result_tfidf_bigram.val_acc, label='bigram ')
    plt.plot(result_tfidf_trigram.n_features, result_tfidf_trigram.val_acc, label='trigram ')
    plt.title("comparison of n-gram model")
    plt.xlabel("num of features")
    plt.ylabel("model validation accuracy")
    plt.grid()
    plt.legend()
```



MODELLING

In this dataset we will be using 4 algorithm such as

- LOGISTIC REGRESSION
- NAÏVE BAYES
- SVM CLASSIFIER
- CNN (CONVOLUTIONAL NEURAL NETWORK)

LOGISTIC REGRESSION

- A supervised machine learning algorithm mainly used for classification tasks.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value.
- It is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete.

TRAIN AND TEST DATA

LOGISTIC REGRESSION

```
[ ] X_train, y_train = df_train['review'], df_train['target']
    X_test, y_test = df_test['review'], df_test['target']
```

```
[ ] y_test = y_test.dropna()
```

```
[ ] lnr = make_pipeline(TfidfVectorizer(), LogisticRegression())
```

```
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

```
[ ] lnr.fit(X_train,y_train)
    y_pred = lnr.predict(X_test.iloc[:len(y_test)]) #to ensure X_test and y_test have same length
```

```
[ ] y_pred = np.nan_to_num(y_pred) # np.nan - to replace nan value with a specific value
```

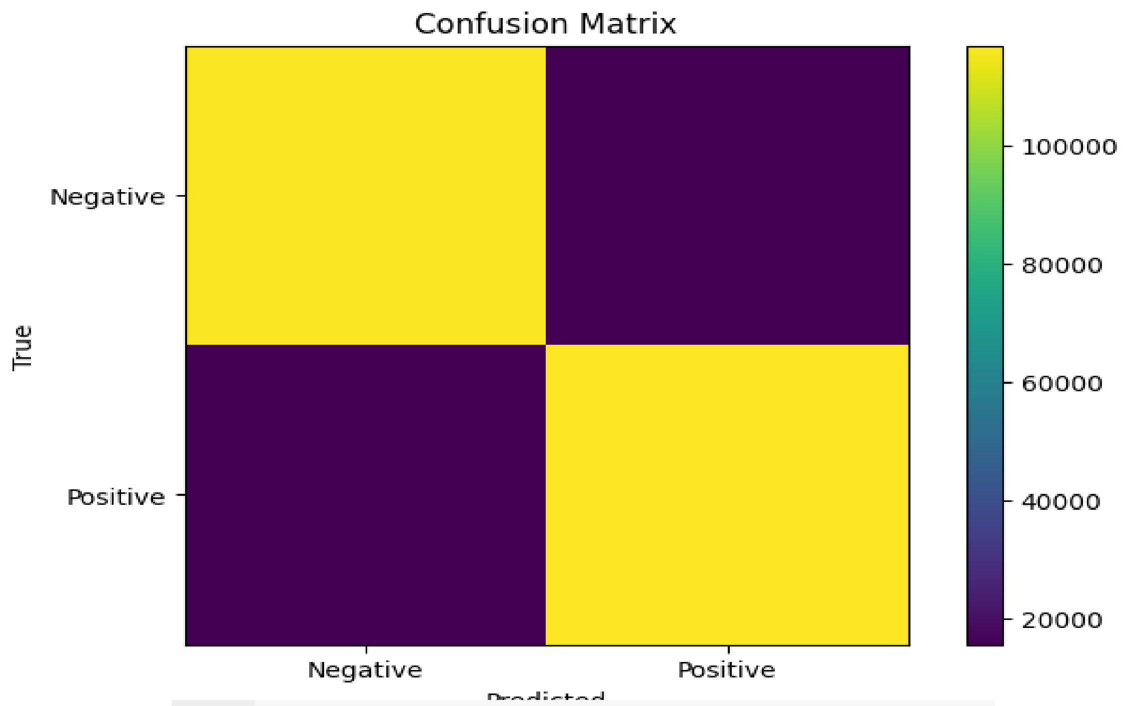
RESULTS AND ACCURACY OF LOGISTIC REGRESSION

```
[ ]  accuracy = accuracy_score(y_test, y_pred)
     print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.90
```

```
[ ]  print(classification_report(y_test, y_pred, digits=3))
```

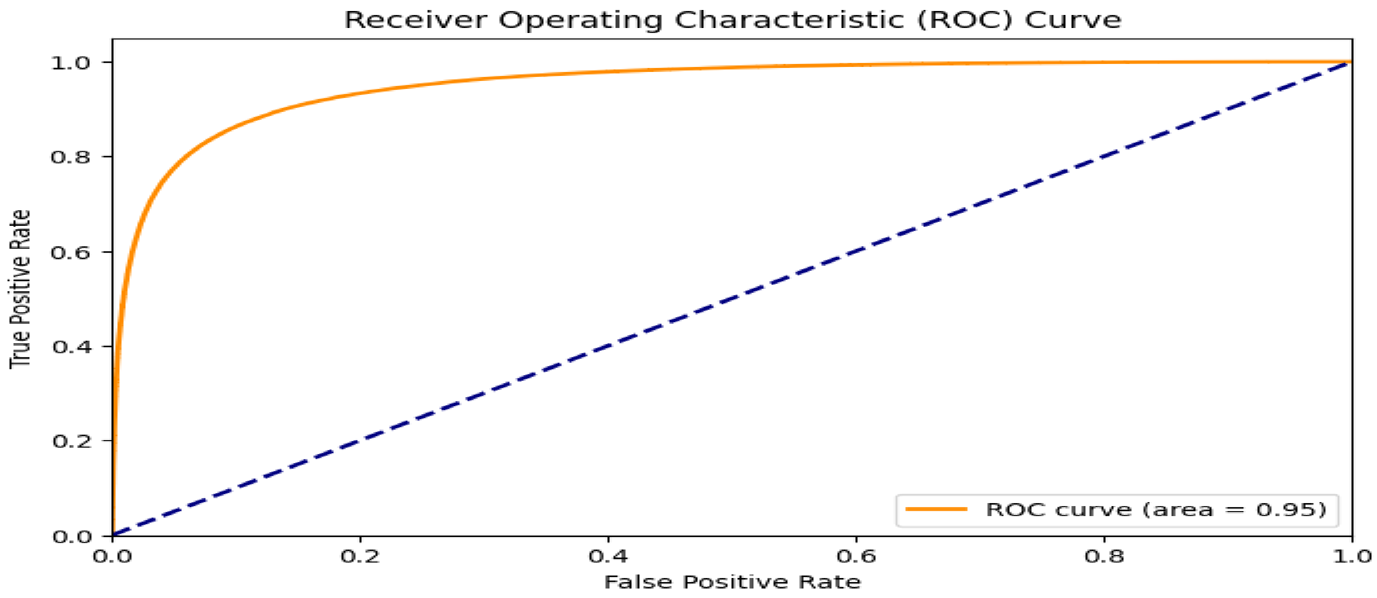
	precision	recall	f1-score	support
0.0	0.900	0.897	0.898	131992
1.0	0.898	0.901	0.899	132681
accuracy			0.899	264673
macro avg	0.899	0.899	0.899	264673
weighted avg	0.899	0.899	0.899	264673




```
The confusion matrix: [[116357  15635]
 [ 15714 116967]]
```

ROC CURVE

- The ROC curve is a graph showing the performance of a classification model at all classification thresholds.



- An ROC curve of 0.95 indicates the model performed excellently.

NAÏVE BAYES CLASSIFIER

- It is a collection of classification algorithms based on bayes theorem.
- The "naive" aspect of Naive Bayes comes from the assumption that features used to describe an observation are conditionally independent, given the class label.
- Naive Bayes can be applied to different types of data. The most common variants include:
- **Multinomial Naive Bayes:** Suitable for discrete data, like word counts in text classification.
- **Gaussian Naive Bayes:** Applicable when features follow a Gaussian distribution, commonly used for continuous data.

TRAIN AND TEST DATA

```
[ ] X_train, y_train = df_train['review'], df_train['target']
    X_test, y_test = df_test['review'], df_test['target']
```

```
[ ] clf = MultinomialNB()
    vectorizer = CountVectorizer(ngram_range=(1,3), max_features=90000)
    X_train = vectorizer.fit_transform(X_train)
    X_test = vectorizer.transform(X_test)
```

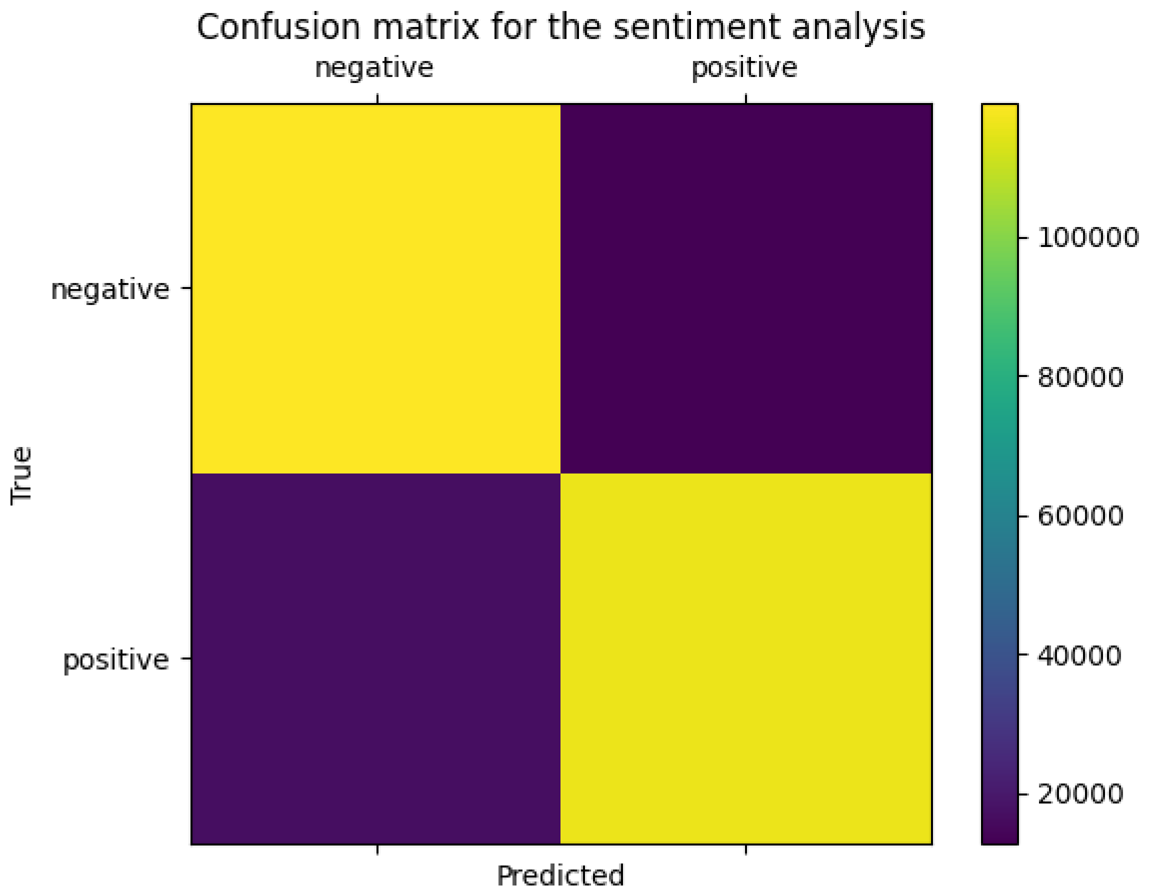
```
[ ] nan_indices = np.isnan(y_test)
    X_test = X_test[~nan_indices]
    y_test = y_test[~nan_indices]
```

```
[ ] clf.fit(X_train, y_train)
    prob = clf.predict_proba(X_test)
    y_pred = np.argmax(prob, axis=-1)
    y_pred_prob = [i[1] for i in prob]
    acc_score = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)

    print(acc_score)
    print(cm)
```

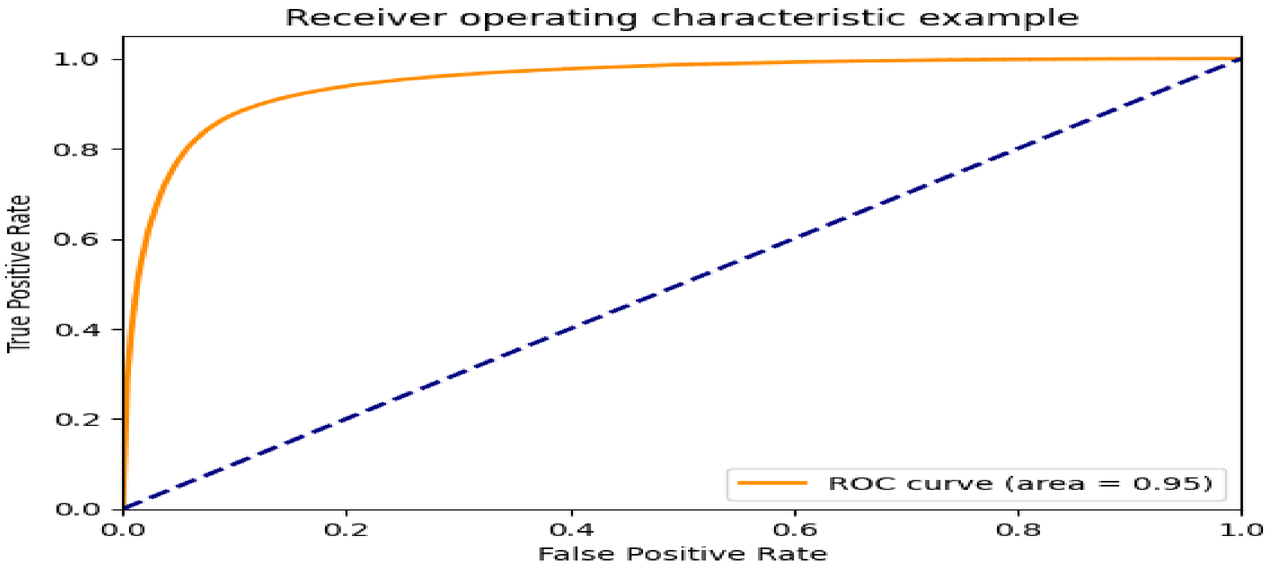
```
0.8884548102753209
[[119210  12782]
 [ 16741 115940]]
```

RESULTS AND ACCURACY

	precision	recall	f1-score	support
0.0	0.877	0.903	0.890	131992
1.0	0.901	0.874	0.887	132681
accuracy			0.888	264673
macro avg	0.889	0.888	0.888	264673
weighted avg	0.889	0.888	0.888	264673



ROC CURVE



An ROC curve of 0.95 indicates the model performed excellently.

SVM CLASSIFIER

- **Support vector machines (SVMs)** are a set of supervised learning methods used for classification and regression.
- Effective in high dimensional spaces
- SVM can handle non-linear relationships by transforming the input features using a kernel function, allowing it to operate effectively in higher-dimensional spaces.
- SVM selects the hyperplane with the maximum margin, enhancing its generalization performance and robustness to noisy data.

TRAIN AND TEST DATA

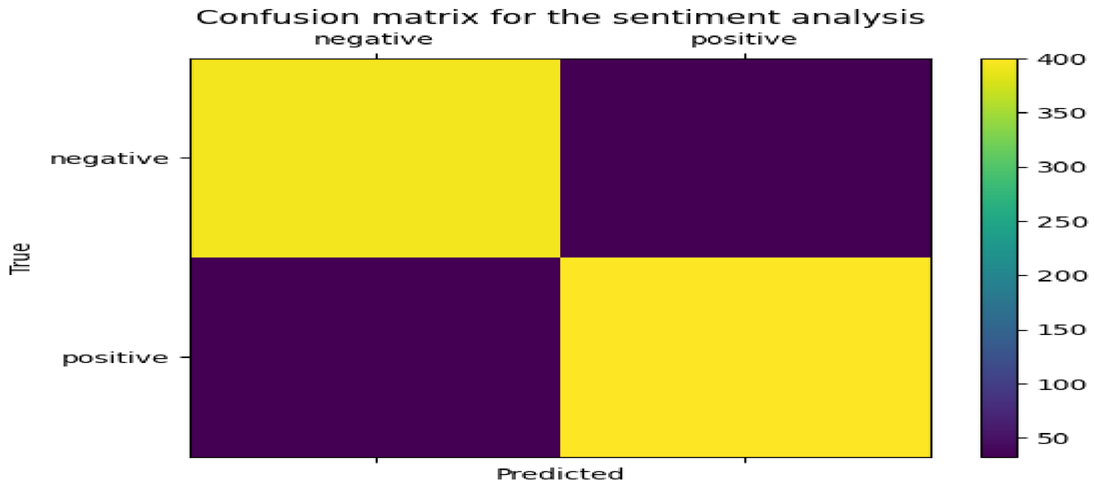
```python
from sklearn.model_selection import StratifiedShuffleSplit, StratifiedKFold
from sklearn.model_selection import GridSearchCV

def find_params(x_train, y_train):
    C_range = np.logspace(-3, 10, 8)
    param_grid = dict(C=C_range)
    cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
    grid = GridSearchCV(LinearSVC(), param_grid=param_grid, cv=cv)
    grid.fit(x_train, y_train)


    score_dict = grid.grid_scores_

    scores = [x[1] for x in score_dict]
    scores = np.array(scores).reshape(len(C_range))

    plt.figure(figsize=(8, 6))
    plt.subplots_adjust(left=0.15, right=0.95, bottom=0.15, top=0.95)
    plt.imshow(scores, interpolation='nearest', cmap=plt.cm.get_cmap("Spectral"))
    plt.ylabel('C')
    plt.colorbar()
    plt.yticks(np.arange(len(C_range)), C_range)
    plt.show()
    return grid.best_params_
```

```python
def preprocess_data(data, frac=1, random_state=1234):
    df = data.sample(frac=frac, random_state=random_state)
    X = df['review']
    y = df['target']
    return X, y

def train_test_split_and_vectorize(X_train, X_test,kernel = 'linear', max_features=None, ngram_range=(1, 3)):
    tf_vectorizer = TfidfVectorizer(ngram_range=ngram_range, max_features=max_features)
    X_train = tf_vectorizer.fit_transform(X_train)
    X_test = tf_vectorizer.transform(X_test)
    return X_train, X_test

def train_and_evaluate(X_train, X_test, y_train, y_test, use_params=False, C=None):
    if use_params:
        params = find_params(X_train, y_train)
        print('Best params:', params)
        svm = CalibratedClassifierCV(base_estimator=LinearSVC(C=params['C']), cv=5)
    else:
        svm = CalibratedClassifierCV(base_estimator=LinearSVC(C=C), cv=5)

    print(svm)
t0 = time()
svm.fit(X_train, y_train)
train_test_time = time() - t0
print("training time: {0:.2f}s".format(train_test_time))

probas = svm.predict_proba(X_test)
y_pred = np.argmax(probas, axis=-1)
y_pred_prob = [i[1] for i in probas]

acc_score = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print("Model accuracy = {0:.3f}%".format(acc_score * 100))
print(classification_report(y_test, y_pred, digits=4))

plot_auc(y_test, y_pred_prob)
plot(cm)
```

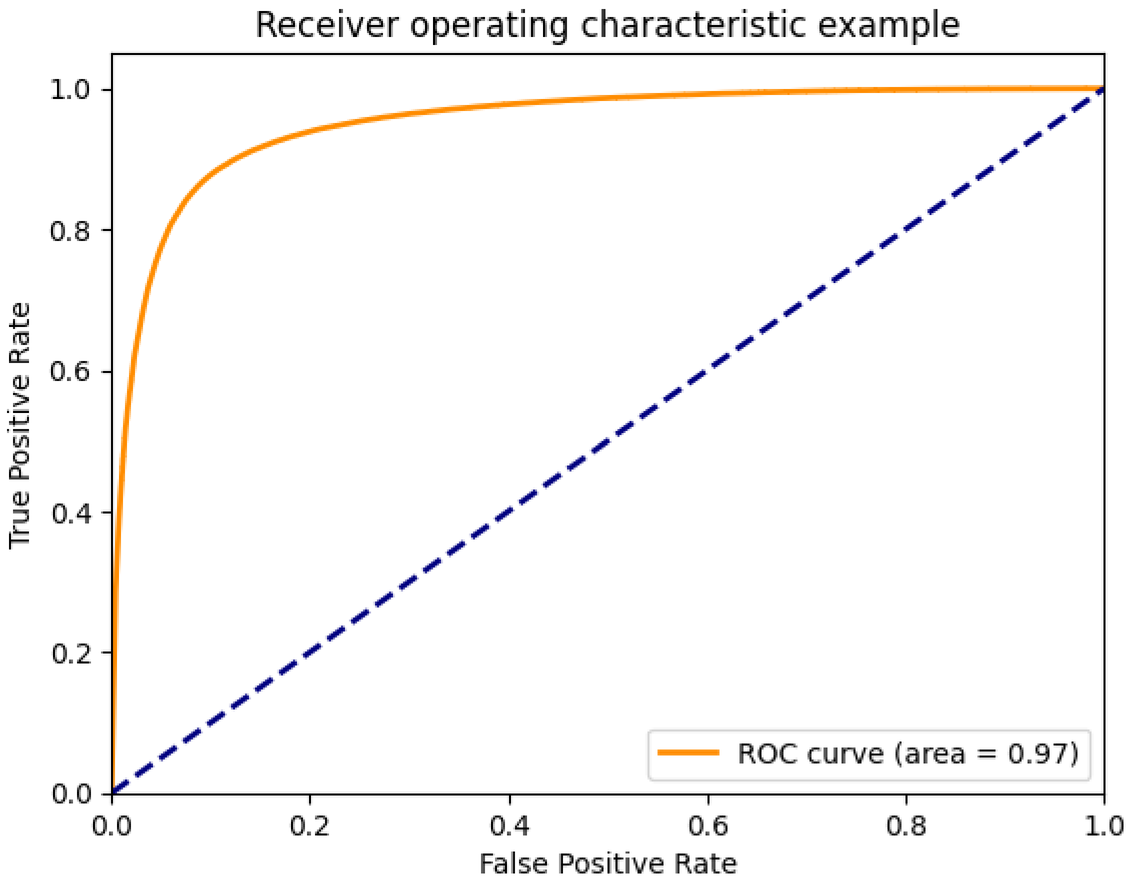
RESULTS AND ACCURACY

```
[ ] train_data, test_data = train_test_split(df_train, test_size=0.02, random_state=45)
    X_train, y_train = preprocess_data(train_data)
    X_test, y_test = preprocess_data(test_data)

    X_train, X_test = train_test_split_and_vectorize(X_train, X_test, max_features=None, ngram_range=(1, 3))

    train_and_evaluate(X_train, X_test, y_train, y_test, use_params=False, C=1.0)
```

```
  warnings.warn(
training time: 11.88s
Model accuracy = 92.209%
              precision    recall  f1-score   support

         0.0     0.9225    0.9204    0.9215       427
         1.0     0.9217    0.9238    0.9227       433

    accuracy                         0.9221       860
   macro avg     0.9221    0.9221    0.9221       860
weighted avg     0.9221    0.9221    0.9221       860
```



ROC CURVE



Receiver operating characteristic example
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
True Positive Rate
False Positive Rate
ROC curve (area = 0.97)

CONVOLUTIONAL NEURAL NETWORK

- Renowned in computer vision, CNNs find utility beyond imagery, extending their effectiveness to text classification tasks, showcasing their adaptability across domains.
- Leveraging Keras with a TensorFlow backend, CNNs are harnessed for natural language processing (NLP) tasks, showcasing their versatility and robust performance in diverse applications.
- Employing tokenization, textual reviews are systematically converted into numerical sequences, facilitating CNNs in processing and understanding the underlying patterns within the language data.
- Characterized by four hidden layers and the integration of multiple optimization algorithms, CNNs for NLP tasks are meticulously crafted, with built-in mechanisms to prevent overfitting and enhance model generalization.

```
from keras.layers import *
from keras.models import Model
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from sklearn.utils import shuffle
```

```
tokenizer = Tokenizer(num_words=max_feature)
tokenizer.fit_on_texts(X_train)
```

```
token_train = tokenizer.texts_to_sequences(X_train)
token_test = tokenizer.texts_to_sequences(X_test)
```

```
X_train_final = pad_sequences(token_train, maxlen=maxlen, padding='post')
X_test_final = pad_sequences(token_test, maxlen=maxlen, padding='post')
```

```
X_train = list(df_train['review'])
X_test = list(df_test['review'])
```

```
y_train = [[1,0] if x == 0 else [0,1] for x in df_train['target']]
y_test = [[1,0] if x == 0 else [0,1] for x in df_test['target']]
```

```
X_train, y_train = shuffle(X_train, y_train)
X_test, y_test = shuffle(X_test, y_test)

y_train = np.array(y_train)
y_test = np.array(y_test)
```

```
max_feature = 5000
maxlen = 100
embed_size = 25
```

```python
from IPython.display import clear_output
import keras

class PlotLosses(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.i = 0
        self.x = []
        self.losses = []
        self.val_losses = []

        self.fig = plt.figure()

        self.logs = []

    def on_epoch_end(self, epoch, logs={}):

        self.logs.append(logs)
        self.x.append(self.i)
        self.losses.append(logs.get('loss'))
        self.val_losses.append(logs.get('val_loss'))
        self.i += 1

        clear_output(wait=True)
        plt.plot(self.x, self.losses, label="loss")
        plt.plot(self.x, self.val_losses, label="val_loss")
        plt.legend()
        plt.show();

plot_losses = PlotLosses()
```

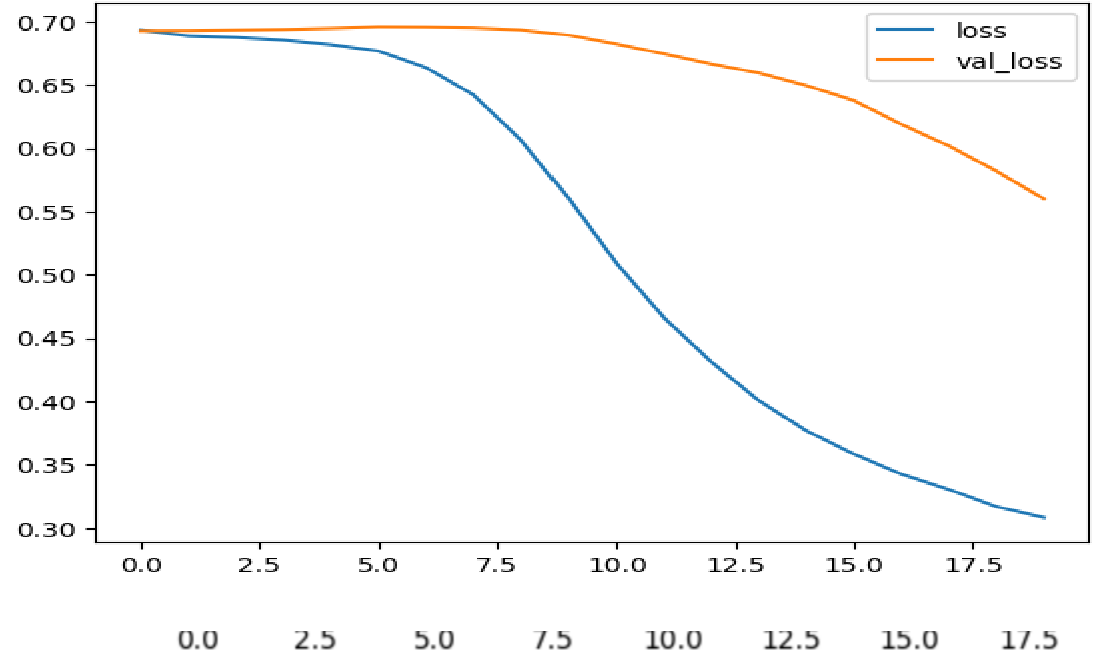
```python
from keras import optimizers
from keras import regularizers
eta = 1
maxlen=100
input = Input(shape=(maxlen,))
net = Embedding(max_feature, embed_size)(input)
net = Dropout(0.2)(net)
net = BatchNormalization()(net)

net = Conv1D(16, 8, padding='same', activation='relu')(net)
net = Dropout(0.2)(net)
net = BatchNormalization()(net)
net = Conv1D(16, 4, padding='same', activation='relu')(net)
net = Dropout(0.2)(net)
net = BatchNormalization()(net)
net = Conv1D(16, 4, padding='same', activation='relu')(net)
net = Dropout(0.2)(net)
net = BatchNormalization()(net)
net = Conv1D(16, 4, padding='same', activation='relu')(net)
net = Dropout(0.2)(net)
net1 = BatchNormalization()(net)

net = Conv1D(2, 1)(net)
net = GlobalAveragePooling1D()(net)
output = Activation('softmax')(net)
model = Model(inputs = input, outputs = output)
ada = optimizers.legacy.Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
model.compile(optimizer=ada, loss='categorical_crossentropy', metrics=['acc'])
model.summary()
```

[ ]

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 100)]             0

 embedding_2 (Embedding)     (None, 100, 25)           125000

 dropout_10 (Dropout)        (None, 100, 25)           0

 batch_normalization_10 (Ba  (None, 100, 25)           100
 tchNormalization)

 conv1d_10 (Conv1D)          (None, 100, 16)           3216

 dropout_11 (Dropout)        (None, 100, 16)           0

 batch_normalization_11 (Ba  (None, 100, 16)           64
 tchNormalization)

 conv1d_11 (Conv1D)          (None, 100, 16)           1040

 dropout_12 (Dropout)        (None, 100, 16)           0

 batch_normalization_12 (Ba  (None, 100, 16)           64
 tchNormalization)

 conv1d_12 (Conv1D)          (None, 100, 16)           1040

 dropout_13 (Dropout)        (None, 100, 16)           0

 batch_normalization_13 (Ba  (None, 100, 16)           64
 tchNormalization)

 conv1d_13 (Conv1D)          (None, 100, 16)           1040
```

```
 batch_normalization_12 (Ba  (None, 100, 16)           64
 tchNormalization)

 conv1d_12 (Conv1D)          (None, 100, 16)           1040

 dropout_13 (Dropout)        (None, 100, 16)           0

 batch_normalization_13 (Ba  (None, 100, 16)           64
 tchNormalization)

 conv1d_13 (Conv1D)          (None, 100, 16)           1040

 dropout_14 (Dropout)        (None, 100, 16)           0

 conv1d_14 (Conv1D)          (None, 100, 2)            34

 global_average_pooling1d_2  (None, 2)                 0
  (GlobalAveragePooling1D)

 activation_2 (Activation)   (None, 2)                 0

=================================================================
Total params: 131662 (514.30 KB)
Trainable params: 131516 (513.73 KB)
Non-trainable params: 146 (584.00 Byte)
```

```python
train_res = model.fit(X_train_final, y_train, batch_size=2048, epochs=20, validation_split=0.1, callbacks=[plot_losses])
acc = train_res.history['val_acc']
print('Accuracy: {}'.format(np.mean(acc)))
```

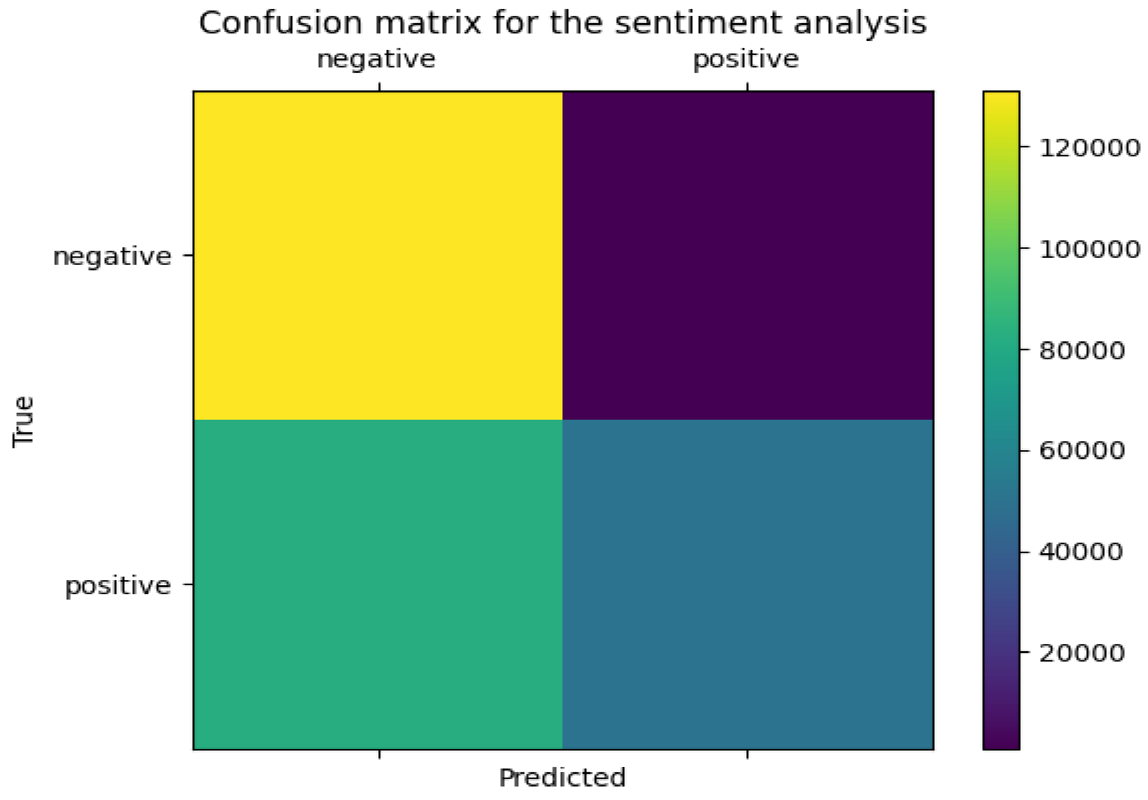



```
19/19 [==============================] - 16s 871ms/step - loss: 0.3083 - acc: 0.8753 - val_loss: 0.5601 - val_acc: 0.6857
Accuracy: 0.5231216564774513
```

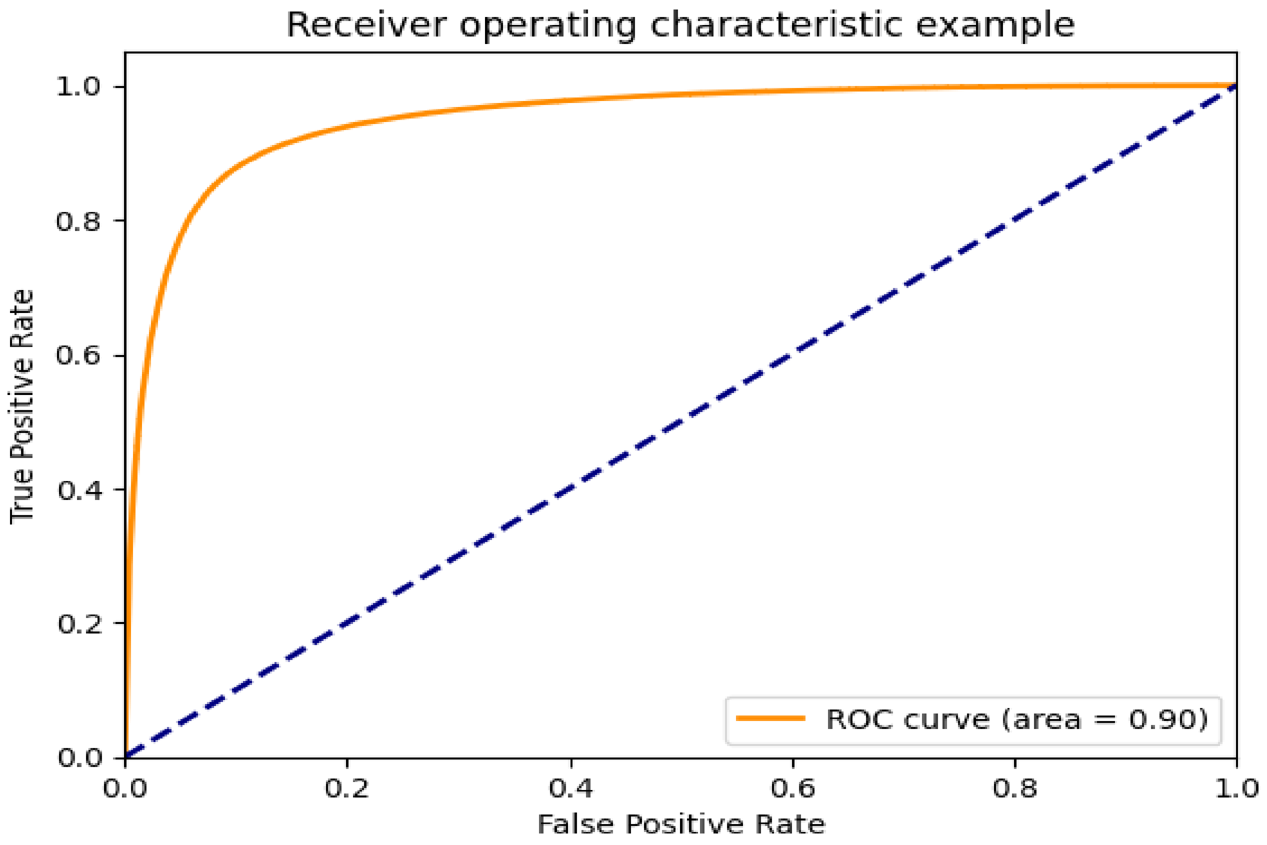
RESULTS AND ACCURACY

```
[ ] print(classification_report(y_test_raw, y_pred, digits=4))
```

	precision	recall	f1-score	support
0	0.6145	0.9931	0.7592	131992
1	0.9823	0.3802	0.5482	132682
accuracy			0.6858	264674
macro avg	0.7984	0.6866	0.6537	264674
weighted avg	0.7989	0.6858	0.6534	264674



ROC CURVE


Receiver operating characteristic example
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.90)

CONCLUSION

- **Linear SVM Superiority:** Demonstrating supremacy over Logistic Regression Naive Bayes and CNN, linear SVM emerged as the top performer, capitalizing on the linear separability inherent in the dataset.
- Advocating for SVM with a Linear Kernel as the preferred model, achieving an impressive accuracy of 92.09%, especially effective in scenarios where data exhibits linear separability, as evident in the analysis of Social Media reviews.
- With linear SVM hinges on meticulous data collection, preprocessing and feature extraction, underscoring the pivotal role of data quality in shaping model outcomes.

PROBLEMS ENCOUNTERED

- Ease of training post-preprocessing for Naive Bayes & SVM due to fewer parameters
- Neural Network complexity in parameter tuning due to numerous variables
- Preferred Model Choice: SVM with linear kernel for Social Media review sentiment analysis
- Emphasis on high-quality data for optimal performance
- Naive Bayes and SVM proved efficient; Neural Network requires meticulous tuning

REFERENCES

- Andreea Salinca. "Convolutional Neural Networks for Sentiment Classification on Business Reviews". In: arXiv (2017).
- Korovkinas, K., & Danėnas, P. (2017). SVM and Naïve Bayes Classification Ensemble Method for sentiment analysis. Baltic Journal of Modern Computing, 5(4).
- Alhashmi, S. M., Khedr, A. M., Arif, I., & El Bannany, M. (2021). Using a Hybrid-Classification Method to Analyze Twitter Data During Critical Events. IEEE Access, 9, 141023–141035.
- Y. Xu, X. Wu, and Q. Wang. Sentiment analysis of yelps ratings based on text reviews, 2015.
- Shaikh, T., & Deshpande, D. (2016). Feature Selection Methods in Sentiment Analysis and Sentiment Classification of Amazon Product Reviews. International Journal of Computer Trends and Technology, 36(4), 225–230.

THANK YOU