Analysis of the Tonality of Texts Based on Machine Learning Method

A project report submitted
to MALLA REDDY
UNIVERSITY
in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLGY
in
COMPUTER SCIENCE & ENGINEERING (AI & ML)

Submitted by

Y. Chandra Shekar : 2011CS020435

Under the Guidance of

Dr. P. Venkateshwara Rao

Associate professor

MALLA REDDY UNIVERSITY
(Telangana State Private Universities Act No.13 of 2020 and G.O.Ms.No.14, Higher Education (UE) Department)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (AI & ML)

Text tonality analysis is a crucial aspect of natural language processing (NLP) that involves identifying the sentiment or emotion conveyed in a piece of text. This project aims to explorethe effectiveness of machine learning techniques for text tonality analysis by comparing several methods and evaluating their accuracy .Text-Based application deal with huge amountof text to perform classification or translation and involves a lot of work on the back end.Transforming text into something an algorithm can digest is a complicated process. In this, wewill discuss the steps involved in text processing

ABSTRACT

The analysis of the tonality of texts is an urgent problem in the field of natural language processing, which is often solved with the help of convolutional neural networks. However, most of these CNN models focus only on the study of local functions, ignoring global features. In this we are using a hybrid convolutional neural network with parallel-sequential connections between layers and from the layer of maximum pulling obtained from the matrix of the original text is proposed for the analysis of text tonality. The proposed hybrid convolutional neural network extracts text features using a parallel-connected convolutional block. Then the neural network classifies the features and combines these features with the original text features. The model of the proposed neural network is able to study both local and global features of short texts and has less convergence time and computing resource compared to the parallel DenseNet. Hybrid convolutional neural network with parallel sequential connections between layers has a higher efficiency of text tone classification in 6 different databases compared to the base models CNN, TextCNN, FastText, DPCNN

CONTENTS

Chapter 1

INTRODUCTION

1.1 PROBLEM DEFINITION

Text tonality analysis is a crucial aspect of natural language processing (NLP) that involves identifying the sentiment or emotion conveyed in a piece of text. This project aims to explore the effectiveness of machine learning techniques for text tonality analysis by comparing several methods and evaluating their accuracy .Text-Based application deal with huge amount of text to perform classification or translation and involves a lot of work on the back end. Transforming text into something an algorithm can digest is a complicated process. In this, we will discuss the steps involved in text processing.

1.2 OBJECTIVE OF PROJECT

Text analysis and text mining are synonyms. They describe the same process of extracting meaning from data by observing patterns.

However, text analysis and text analytics are a bit different things:

- Text analysis works with the concepts, the meaning of the text. Text analysis can be used to answer these questions: is a review positive or negative? What is the main topic of the text?

- Text analytics studies patterns. The results can be shown on graphs, schemes, and spreadsheets. If you want to estimate the percentage of positive customer feedback, you will need text analytics.

1.3 LIMITATIONS OF PROJECT

- Time consuming
- Complicated process

Chapter 2

ANALYSIS

2.1 INTODUCTION

In Text based analysis every piece of content can be analyzed on a deeper level in order to understand more about the author or the topic of the text. By introducing ML text analysis, we can provide users with better services:

- Provide answers to FAQs;
- Monitor public sentiment towards products and services;
- Facilitate paperwork through clustering and classification of documents.

Companies become much more efficient at communicating with their customers: by studying customer feedback, a company can discover public opinion about their products. ML algorithms can automatically classify customer support tickets or reviews by topic or language they are written in.

ML makes textual analysis much faster and more efficient than manual processing of texts. It allows to reduce labor costs and speed up the processing of texts without compromising on quality.

2.2 SOFTWARE REQUIREMENT SPECIFICATION

2.2.1 Software Requirement

- Jupyter Notebook
- VS Code

2.2.2 Hardware Requirement

- 8 GB RAM
- 128 GB ROM
- PROCESSOR ABOVE 1.4 GHz
- WINDOWS 10 OR HIGHER

2.3 EXISTING SYSTEM

- Rule-based approaches use a set of predefined rules to classify the text into different sentiment or tonality categories.

- The rules may be based on patterns in the text, such as the use of certain words or phrases, or they may be based on linguistic features such as the presence of negation or intensifiers.

- Rule-based approaches can be highly accurate, but may require significant expertise and effort to develop the rules.

2.4 PROPOSED SYSTEM

- Machine learning approaches use algorithms to train a model to classify the text into different sentiment or tonality categories.

- The model is typically trained on a labeled dataset of text examples and learns to identify patterns and features in the text that are associated with each sentiment or tonality category.

- Machine learning approaches can be highly accurate and adaptable, but may require significant amounts of training data and computational resources.

2.5 MODULES

Data Pre-processing Module

Data Pre-processing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data Pre- processing task

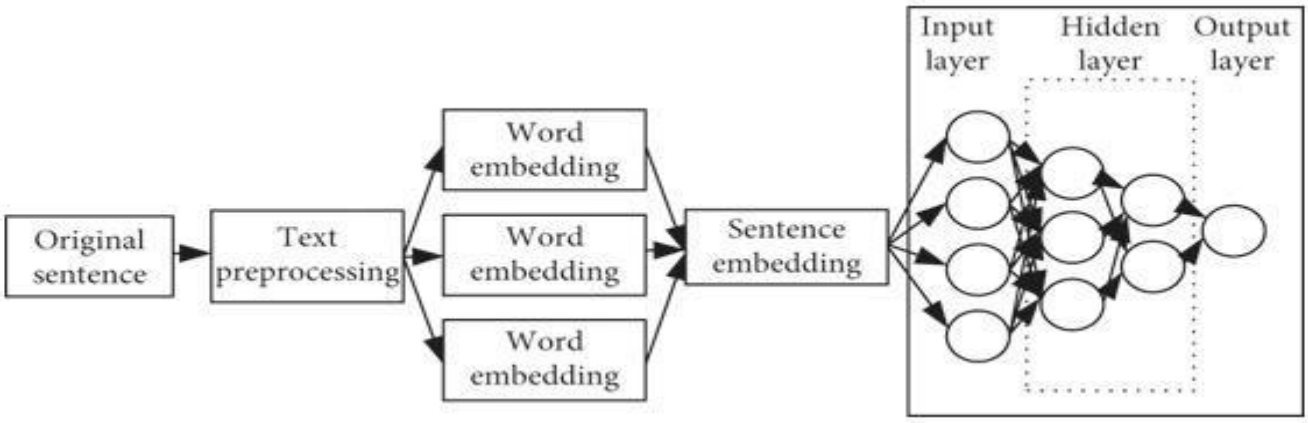
Data Visualization Module

Visualizing the data for data analysis. We can find the relations between the attributes and can work on them to make any necessary changes on our training data.

Training and Testing

In this Module we will train the system using Convolutional neural network . Using the training Model the system will produce the classifies the testing data

2.6 ARCHITECTURE



Chapter 3

DESIGN

3.1 INTRODUCTION

Text analysis (TA) is a machine learning technique used to automatically extract valuable insights from unstructured text data. Companies use text analysis tools to quickly digest online data and documents, and transform them into actionable insights.

we can us text analysis to extract specific information, like keywords, names, or company information from thousands of emails, or categorize survey responses by sentiment and topic.

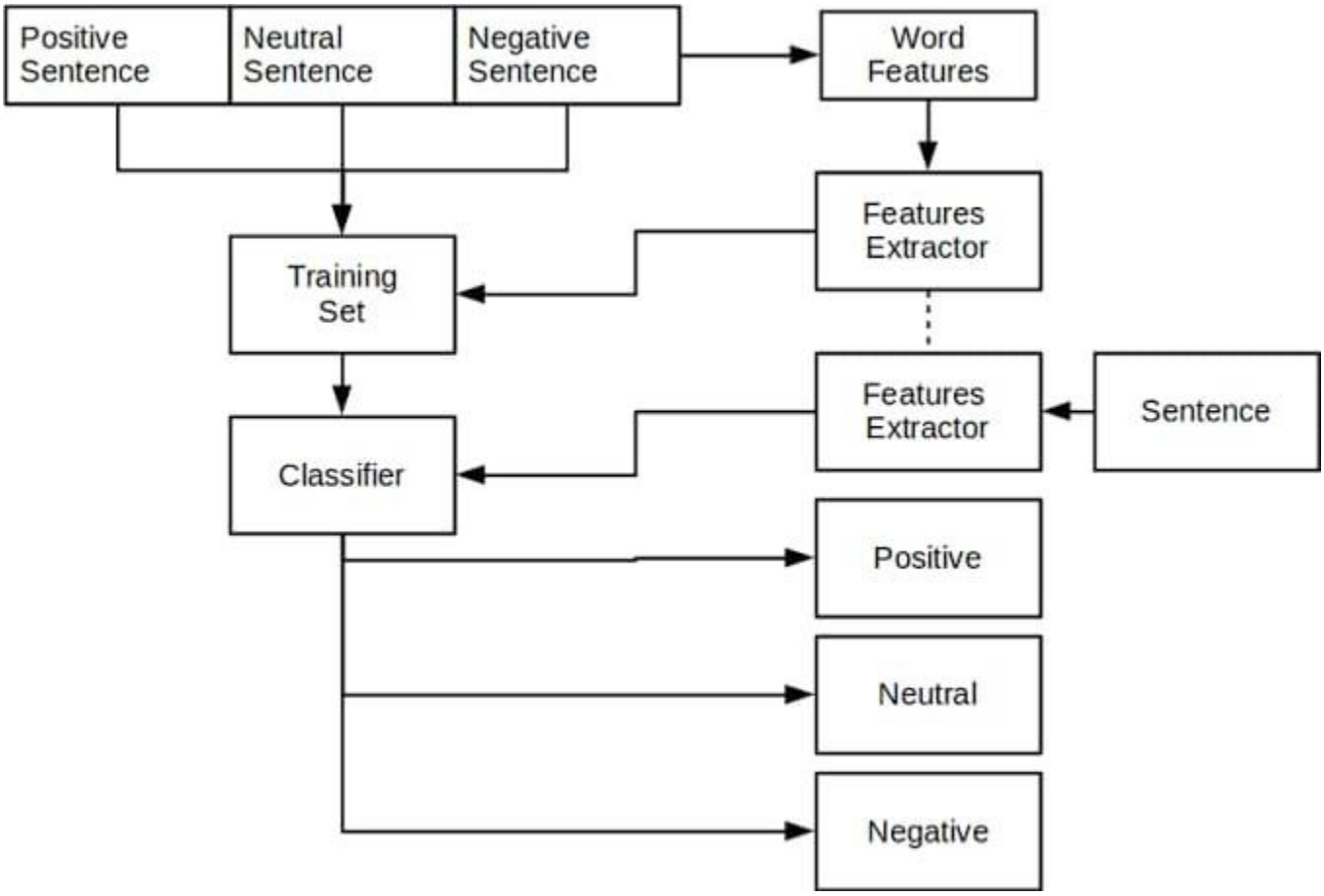
Text analysis delivers qualitative results and text analytics delivers quantitative results. If a machine performs text analysis, it identifies important information within the text itself, but if it performs text analytics, it reveals patterns across thousands of texts, resulting in graphs, reports, tables etc.

Sentiment analysis uses powerful machine learning algorithms to automatically read and classify for opinion polarity (positive, negative, neutral) and beyond, into the feelings and emotions of the writer, even context and sarcasm.

As part of text analysis, there's also natural language processing (NLP), also termed natural language understanding. It's a form of sentiment analysis that helps technology to "read" or understand text from natural human language. Natural language processing algorithms can use machine learning to understand and evaluate valuable data, consistently and without any bias. It can be sophisticated enough to understand the context of text data, even with complicated concepts and ambiguities.

Hence, it is very important to use specialized text analytics platforms for Voice of the Customer or Employee data as opposed to general text mining tools available out there.

3.2 UML diagram



3.3 Data set description

ML, a branch of Artificial Intelligence, relates the problem of learning from data samples to the general concept of inference .Every learning process consists of two phases:

(i) Estimation of unknown dependencies in a system from a given dataset and
(ii) Use of estimated dependencies to predict new outputs of the system.

ML has also been proven an interesting area in biomedical research with many applications, where an acceptable generalization is obtained by searching through an n-dimensional space for a given set of biological samples, using different techniques and algorithms. There are two main common types of ML methods known as

(i) Supervised learning
(ii) Unsupervised learning

In supervised learning a labeled set of training data is used to estimate or map the input data to the desired output. In contrast, under the unsupervised learning methods no labeled examples are provided and there is no notion of the output during the learning process. As a result, it is up to the learning scheme/model to find patterns or discover the groups of the input data

3.4 Data Pre Processing Techniques

Text Cleaning: This involves removing any irrelevant or unnecessary data from the text such as special characters, punctuations, and stop words. This is to ensure that the text data is in a clean and structured format for analysis.

Tokenization: This is the process of splitting the text into individual words or tokens. Tokenization is an essential step in sentiment analysis because it helps to identify and categorize individual words that may carry a positive, negative, or neutral sentiment.

Part-of-speech (POS) tagging: This involves assigning a part-of-speech tag to each token in the text such as noun, verb, adjective, or adverb. POS tagging helps to identify the context of each word in the text, which can help in determining the sentiment.

Stemming and Lemmatization: These techniques are used to reduce words to their base or root form. Stemming involves removing the suffixes from words, while lemmatization involves converting words to their dictionary form. This technique can help to reduce the number of unique words in a text and improve the accuracy of the analysis.

Sentiment lexicon: This is a dictionary of words that are pre-assigned a positive, negative, or neutral sentiment score. The lexicon is used to identify the sentiment polarity of each word in the text and can help to determine the overall sentiment of the text.

Machine learning algorithms: Machine learning algorithms can be used to identify patterns in the text data that can be used to predict the sentiment. These algorithms can be trained on labeled data and can be used to classify the sentiment of new, unlabeled text data.

3.5 Methods and Algorithm:

Support Vector Machines (SVMs) are a popular machine learning algorithm used in sentiment analysis to classify text data as positive, negative, or neutral based on the emotional content of the text.

SVMs work by finding the optimal hyperplane that separates positive and negative examples in the feature space. In sentiment analysis, the feature space is typically defined by a bag-of-words model, where each word in the text is represented as a feature, and the frequency or presence of each word is used as the value of the feature.

To train an SVM for sentiment analysis, labeled data is needed, which consists of text examples labeled as positive, negative, or neutral. The SVM is then trained to learn the optimal hyperplane that separates positive and negative examples based on their feature values.

Once the SVM is trained, it can be used to predict the sentiment of new, unlabeled text data by representing the text data as a bag-of-words feature vector and applying the SVM's learned decision boundary to classify the text as positive, negative, or neutral.

SVMs have been shown to be effective in sentiment analysis tasks, particularly when the feature space is high-dimensional and the data is well-separated. However, SVMs can be sensitive to the choice of kernel function, which determines how the feature space is transformed to a higher-dimensional space, and to the choice of hyper parameters, which can affect the performance of the SVM.

3.6 Building a model:

Building a model for sentiment analysis typically involves the following steps:

Data Collection: The first step in building a sentiment analysis model is to collect data. This can be done by scraping text data from various sources, such as social media platforms, product reviews, or customer feedback.

Data Preprocessing: Once the data is collected, it needs to be preprocessed to make it suitable for the model. This typically involves tasks such as removing stop words, stemming, and lemmatization to reduce the dimensionality of the data.

Feature Extraction: The next step is to extract features from the preprocessed text data. This can be done using techniques such as the bag-of-words model, word embeddings, or topic modeling.

Model Training: Once the features are extracted, the next step is to train a sentiment analysis model using labeled data. This can be done using various machine learning algorithms, such as Naive Bayes, Support Vector Machines, or Deep Learning models such as Recurrent Neural Networks.

Model Evaluation: After training the model, it is important to evaluate its performance using metrics such as accuracy, precision, recall, and F1-score. This helps to identify any weaknesses or areas for improvement in the model.

Model Deployment: Once the model has been trained and evaluated, it can be deployed in a production environment, such as a web application, to perform sentiment analysis on new text data.

It is important to note that building an effective sentiment analysis model requires careful

consideration of the data, feature extraction techniques, and machine learning algorithms used. Additionally, it is important to continuously monitor and update the model to ensure it remains effective as the data and language evolve over time.

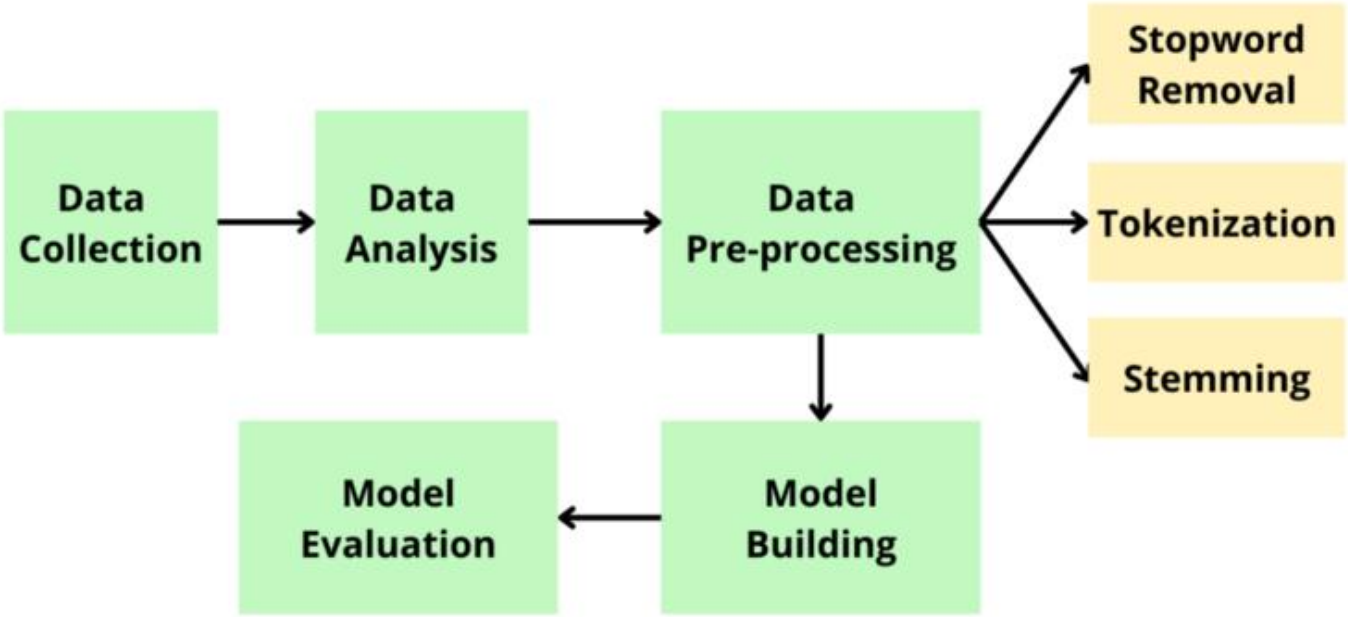
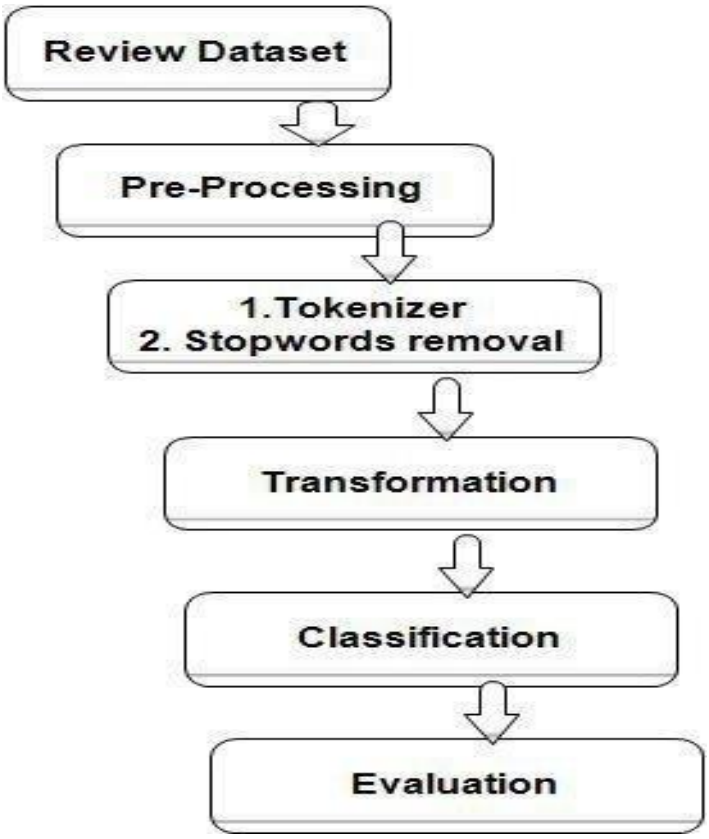


Figure 3.6

3.7 Evalution

It's important to use new data when evaluating our model to prevent the likelihood of overfitting to the training set. However, sometimes it's useful to evaluate our model as we're building it to find that best parameters of a model - but we can't use the test set for this evaluation or else we'll end up selecting the parameters that perform best on the test data but maybe not the parameters that generalize best.

To evaluate the model while still building and tuning the model, we create a third subset of the data known as the validation set. A typical train/test/validation split would be to use 60% of the data for training, 20% of the data for validation, and 20% of the data for testing The Holdout method is used to evaluate the model performance and uses two types of data for testing and training. The test data is used to calculate the performance of the model whereas it is trained using the training data set. This method is used to check how well the machine learning model developed using different algorithm techniques



Chapter 4

DEPLOYMENT AND RESULTS

4.1 Introduction

Sentiment analysis, also known as opinion mining, is a technique used to extract and analyze emotions, attitudes, and opinions expressed in text-based data such as social media posts, product reviews, customer feedback, and news articles. The main objective of sentiment analysis is to identify the polarity (positive, negative, or neutral) of a piece of text and to extract and categorize the emotions and attitudes expressed in the text.

Sentiment analysis has become increasingly important in recent years, as the volume of text-based data continues to grow exponentially. Companies use sentiment analysis to monitor their brand reputation, track customer feedback, and gain insights into customer preferences and trends. Governments and public agencies use sentiment analysis to monitor public opinion on social and political issues. Researchers use sentiment analysis to analyze public

opinion on various topics and to identify emerging trends.

There are various techniques and algorithms used in sentiment analysis, ranging from rule-based methods to machine learning models such as Naive Bayes, Support Vector Machines, and deep learning models such as Recurrent Neural Networks. These algorithms use various features such as word frequency, word embedding, and topic modeling to identify and categorize sentiments in text-based data.

In summary, sentiment analysis is a powerful technique used to extract and analyze emotions, attitudes, and opinions expressed in text-based data, and it has numerous applications in various fields such as business, politics, and research**.**

4.2 Source Code

```
##  Importing the libraries

import pandas as pd
import numpy as np
import csv
import re
import string
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt

df = pd.read_csv("IMDB Dataset.csv")

df.head()
df.shape
df.size
df.describe()
df.info()
df['sentiment'].value_counts()
df0=df[df['sentiment']== 'positive']
df1=df[df['sentiment'] == 'negative']
df0.shape[0], df1.shape[0]

##Positive reviews
from wordcloud import WordCloud
positive_review = df[df.sentiment == 'positive']["review"]
positive_review_string = ' '.join(positive_review)
plt.figure(figsize=(20,20))
wc = WordCloud(max_words=1200,width = 1200, height=600,background_color=
    "white").generate(positive_review_string)
plt.imshow(wc,interpolation='bilinear')
plt.axis('off')
plt.title("Word count for positive reviews",fontsize=20)
plt.show()


##Negative reviews
```

```
negative_data = df[df.sentiment == 'negative']['review']
negative_data_string = ' '.join(negative_data)
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 2000, width=1200,
    height=600,background_color="white").generate(negative_data_string)
plt.imshow(wc , interpolation = 'bilinear')
plt.axis('off')
plt.title('Word cloud for negative reviews',fontsize = 20)
plt.show()

fig,(ax1,ax2)=plt.subplots(1,2,figsize=(12,8))
text_len=positive_review.str.len()
ax1.hist(text_len,color='green')
ax1.set_title('Positive Reviews')
ax1.set_xlabel('Number of Characters')
ax1.set_ylabel('Count')
text_len=negative_data.str.len()
ax2.hist(text_len,color='red')
ax2.set_title('Negative Reviews')
ax2.set_xlabel('Number of Characters')
ax2.set_ylabel('Count')
fig.suptitle('Number of characters in texts')
plt.show()

##Most common postive words and negative words
df.head()

def get_corpus(data):
    corpus = []
    for i in data:
        for j in i.split():
            corpus.append(j.strip())
    return corpus
corpus = get_corpus(df.review)
corpus[:5]

from collections import Counter
count = Counter(corpus)
most_words = count.most_common(10)
most_common = pd.DataFrame(most_words,columns=["words","count_"])
```

```
# Sorting
most_common= most_common.sort_values('count_')
most_common

plt.figure(figsize=(10,10))
plt.yticks(range(len(most_common)),list(most_common.words))
plt.barh(range(len(most_common)),list(most_common.count_),align='center',color =
    'lightgreen')
plt.title("Most common words in dataset")
plt.show()

##Data PreProcessing
##checking for null values

total_null = df.isnull().sum().sort_values(ascending = False)
percent = ((df.isnull().sum()/df.isnull().count())*100).sort_values(ascending = False)
print("Total records = ", df.shape[0])

missing_data = pd.concat([total_null,percent.round(2)],axis=1,keys=['Total Missing','In
    Percent'])
missing_data

df0=df[df['sentiment']== 'positive']
df1=df[df['sentiment'] == 'negative']
df0.shape, df1.shape

df0=df0[:int(df0.shape[0]/10)]
df1=df1[:int(df1.shape[0]/10)]
df0.shape, df1.shape

df=pd.concat([df0,df1],axis=0)
df = df.sample(frac = 1)
df.shape

df.head()

#Remove Hashtags
df['review'].replace( { r"#(\w+)" : " }, inplace= True, regex = True)

#Remove Mention
```

```
df['review'].replace( { r"@(\w+)" : '' }, inplace= True, regex = True)

#Remove URL
df['review'].astype(str).replace( { r"http\S+" : '' }, inplace= True, regex = True)

#to lowercase
df['review']=df['review'].str.lower()


import nltk
nltk.download('stopwords')

# Import stopwords with nltk.
from nltk.corpus import stopwords
stop = stopwords.words('english')

df['review'] = df['review'].apply(lambda x: ' '.join([word for word in x.split() if word not in
    (stop)]))

def punctuations_removal(text):
    puntuations = "[\.\?!,;:]+"
    text = re.sub(puntuations,"",text)
    return text

df['review'] = df['review'].apply(lambda x: punctuations_removal(x))

import nltk
nltk.download('punkt')

import nltk
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
def stem_sentence(sentence):
    words = nltk.word_tokenize(sentence.lower())
    stemmed_words = [stemmer.stem(word) for word in words if word not in stop]
    stemmed_sentence = " ".join(stemmed_words)
    return stemmed_sentence
df["review"] = df["review"].apply(stem_sentence)

df.head()
```

```
##Same EDA graph after reduction of dataset

positive_review2 = df[df.sentiment == 'positive']["review"]
positive_review_string2 = ' '.join(positive_review2)
plt.figure(figsize=(20,20))
wc = WordCloud(max_words=1200,width = 1200, height=600,background_color=
    "white").generate(positive_review_string2)
plt.imshow(wc,interpolation='bilinear')
plt.axis('off')
plt.title("Word count for positive reviews",fontsize=20)
plt.show()

negative_data2 = df[df.sentiment == 'negative']['review']
negative_data_string2 = ' '.join(negative_data2)
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 2000, width=1200,
    height=600,background_color="white").generate(negative_data_string2)
plt.imshow(wc , interpolation = 'bilinear')
plt.axis('off')
plt.title('Word cloud for negative reviews',fontsize = 20)
plt.show()

from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['sentiment']= label_encoder.fit_transform(df['sentiment'])
df.head()

y = df['sentiment']
x = df['review']
x
y

from sklearn.feature_extraction.text import TfidfVectorizer


# Creating a word corpus for vectorization
corpus = []
for i in range(x.shape[0]):
   corpus.append(x.iloc[i])
```

```
vectorizer1 = TfidfVectorizer(max_features=1000)
X1 = vectorizer1.fit_transform(x)
feature_names1 = vectorizer1.get_feature_names()
denselist1 = X1.todense().tolist()
df = pd.DataFrame(denselist1, columns=feature_names1)


corpus = []
for i in range(x.shape[0]):
    corpus.append(x.iloc[i])

vectorizer = CountVectorizer(max_features=1000)
X = vectorizer.fit_transform(corpus)
feature_names = vectorizer.get_feature_names()
denselist = X.todense().tolist()
train = pd.DataFrame(denselist, columns=feature_names)
x
y

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn import preprocessing
import warnings
warnings.filterwarnings('ignore')
from sklearn.tree import DecisionTreeClassifier


# splitting the training and testing part from the data
X_temp, X_test, y_temp, y_test = train_test_split(df, y, test_size=0.2, random_state=0)

# For creating a table of the accuracies in the end
accuracy = {'TF-IDF':[]}
```

```
# Linear Regression
regressor_LL_tf = LinearRegression()
regressor_LL_tf.fit(X_temp, y_temp)
y_predict_LL_tf = regressor_LL_tf.predict(X_test)
a=(regressor_LL_tf.score(X_test, y_test))
accuracy['TF-IDF'].append(a)

# Logistic Regression
regressor_LR_tf = LogisticRegression(C=1.0,penalty='l2',solver='newton-cg')
regressor_LR_tf.fit(X_temp, y_temp)
y_predict_LR_tf = regressor_LR_tf.predict(X_test)
a=(regressor_LR_tf.score(X_test, y_test))
accuracy['TF-IDF'].append(a)

# Decision Tree
model_DT_tf = DecisionTreeClassifier(criterion = 'gini', max_depth=2)
model_DT_tf.fit(X_temp, y_temp)
y_predict_DT_tf = model_DT_tf.predict(X_test)
a=(model_DT_tf.score(X_test,y_test))
accuracy['TF-IDF'].append(a)

# Random Forest
model_RF_tf = RandomForestClassifier(n_estimators= 100, max_features = 'log2')
model_RF_tf.fit(X_temp, y_temp)
y_predict_RF_tf = model_RF_tf.predict(X_test)
a=(model_RF_tf.score(X_test,y_test))
accuracy['TF-IDF'].append(a)

# K-Neighbors Classifier
model_KN_tf = KNeighborsClassifier(metric= 'manhattan', n_neighbors= 5, weights=
    'distance')
model_KN_tf.fit(X_temp, y_temp)
y_predict_KN_tf = model_KN_tf.predict(X_test)
a=(model_KN_tf.score(X_test,y_test))
accuracy['TF-IDF'].append(a)

# Evaluation
model = ['Linear','Logistic','DT','RF','KN']
data = {'model':model,'accuracy':accuracy['TF-IDF']}
```

```
compare_models = pd.DataFrame(data)

compare_models

##Plotting of Accuracies

compare_models["accuracy"].tolist()

model = ['linear','logistic','DT','RF','KN']
acc = compare_models["accuracy"].tolist()
plt.figure(figsize=(10,8))
graph = plt.bar(model,acc)
plt.xlabel('Accuracy')
plt.ylabel('Models')
graph[0].set_color('green')
graph[1].set_color('yellow')
graph[2].set_color('orange')
graph[3].set_color('lightblue')
graph[4].set_color('pink')

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score,
    confusion_matrix

##Linear Regression
lr = LinearRegression()
param_grid = {
   'fit_intercept': [True, False],
   'normalize': [True, False]
}
grid_search = GridSearchCV(estimator=lr, param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_temp, y_temp)

print("Best hyperparameters: ", grid_search.best_params_)
print("Best accuracy: ", grid_search.best_score_)

y_pred_lin_reg_ht = grid_search.predict(X_test)


## LogisticRegression
```

```
param_grid = {
    'C': [0.1, 1, 10],
    'penalty': ['l2']
}
log = LogisticRegression(random_state=42, solver='liblinear')
grid_search = GridSearchCV(estimator=log, param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_temp, y_temp)

print("Best hyperparameters: ", grid_search.best_params_)
print("Best accuracy: ", grid_search.best_score_)

y_pred_log_ht = grid_search.predict(X_test)

##DecisionTree
param_grid = {
    'max_depth': [2, 4],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1]
}
dt = DecisionTreeClassifier(random_state=42)
grid_search = GridSearchCV(estimator=dt, param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_temp, y_temp)

print("Best hyperparameters: ", grid_search.best_params_)
print("Best accuracy: ", grid_search.best_score_)

y_pred_dt_ht = grid_search.predict(X_test)

##RandomForest
param_grid = {
    'n_estimators': [50],
    'max_depth': [2, 4],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1]
}
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_temp, y_temp)

print("Best hyperparameters: ", grid_search.best_params_)
```

```
print("Best accuracy: ", grid_search.best_score_)

y_pred_rf_ht = grid_search.predict(X_test)

##KNN
param_grid = {
   'n_neighbors': [3, 5],
   'weights': ['uniform', 'distance']
}
knn = KNeighborsClassifier()
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_temp, y_temp)

print("Best hyperparameters: ", grid_search.best_params_)
print("Best accuracy: ", grid_search.best_score_)

y_pred_knn_ht = grid_search.predict(X_test)

##Accuracies calculations
#lin_reg_acc_ht=accuracy_score(y_test,y_pred_lin_reg_ht)
log_reg_acc_ht=accuracy_score(y_test,y_pred_log_ht)
dt_acc_ht=accuracy_score(y_test,y_pred_dt_ht)
rf_acc_ht=accuracy_score(y_test,y_pred_rf_ht)
knn_acc_ht=accuracy_score(y_test,y_pred_knn_ht)

##Plotting accuracies
names=['Logistic Regression', 'Decision Tree Classification' ,'Random Forest
    Classification','KNN']
acc=[log_reg_acc_ht,dt_acc_ht,rf_acc_ht,knn_acc_ht]

plt.figure(figsize=(10, 8))
graph = plt.barh(names, acc)
plt.xlabel('Accuracy')
plt.ylabel('Models')
graph[0].set_color('yellowgreen')
graph[1].set_color('lightcoral')
graph[2].set_color('cyan')
graph[3].set_color('gold')
for i, v in enumerate(acc):
   plt.text(v, i, str(round(v, 2)), color='black', fontsize=12, va='center')
```

```
plt.show()
```

4.3 FINAL RESULTS

	model	accuracy
0	Linear	0.414117
1	Logistic	0.839000
2	DT	0.676000
3	RF	0.816000
4	KN	0.559000