SENTIMENT ANALYSIS FOR MARKETING

PHASE 3 PROJECT

BY:

THANUSRE A

SANGEETHA K

DHARSHINI T

SARUMATHY E

Sentiment Analysis Using Various ML Classifiers

In this document you will see text precessing on twitter data set and after that I have performed different Machine Learning Algorithms on the data such as Logistic Regression, RandomForestClassifier, SVC, Naive Bayes **to classifiy positive and negative tweets.**

Index

Importing libraries

In [1]:

```
# DataFrame
import pandas as pd

# plotting
import seaborn as sns
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# nltk
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# sklearn
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix, classification_report

from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import BernoulliNB


#tensorflow
import tensorflow.compat.v2 as tf
import tensorflow_datasets as tfds
```

```
# Utility
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
import re
import string
import pickle
```

Loading Dataset

In [2]:

```
# Construct a tf.data.Dataset
data = pd.read_csv('/kaggle/input/sentiment140/training.1600000.processed.noemoticon.csv',encoding='latin', names = ['polarity','id','date','query','user','text'])
```

In [3]:

```
data = data.sample(frac=1)
data = data[:200000]
```

Data Visualization

Dataset details target: the polarity of the tweet (0 = negative, 4 = positive)

- date : the date of the tweet (Sat May 16 23:58:44 PDT 2009)
- polarity : the polarity of the tweet (0 = negative 4 = positive)
- user : the user that tweeted (TerraScene)
- text : the text of the tweet (i'm 10x cooler than all of you)

In [4]:

```
print("Dataset shape:", data.shape)
```

```
Dataset shape: (200000, 6)
```

In [5]:

```
data.head(10)
```

Out[5]:

	polarity	Id	date	query	user	text
722084	0	2261382364	Sat Jun 20 20:42:18 PDT 2009	NO_QUERY	AdrianSpeck	Leg hurts can hardly walk
1157616	4	1979198314	Sun May 31 02:01:46 PDT 2009	NO_QUERY	makeupbylinvia	@thoughtcloud Cool that'd be awesome! Thanks....

	polarity	Id	date	query	user	text
1064123	4	1964603753	Fri May 29 14:30:11 PDT 2009	NO_QUERY	LeeAnnWalsh	@lilyroseallen sorry to keep saying
1483162	4	2067415190	Sun Jun 07 12:18:11 PDT 2009	NO_QUERY	Wildertamer23	Woke up at 11. I feel awesome
501963	0	2187369967	Mon Jun 15 20:10:55 PDT 2009	NO_QUERY	projvolunteer	Sorry for lack of updates I don't have the int...
1210662	4	1989073218	Mon Jun 01 00:15:28 PDT 2009	NO_QUERY	Lozzaquinn	@ home about todo mums hair.....straightening...
1030149	4	1932884564	Tue May 26 22:27:02 PDT 2009	NO_QUERY	maxlemesh	@blindmonk ??????.. ? ??? ??? ????? ???????? ???
390866	0	2054783653	Sat Jun 06 08:04:36 PDT 2009	NO_QUERY	pulz27	apparently its too early in the day to buy a c...
798832	0	2328735351	Thu Jun 25 09:55:53 PDT 2009	NO_QUERY	SarahTolson	@Tolsonii Ooooooohhhhh how sad.....
702103	0	2255294620	Sat Jun 20 11:03:02 PDT 2009	NO_QUERY	cmariewaters	i'm sad. i'm bored and nobody wants to text me...

In [6]:
```
data['polarity'].unique()
```

Out[6]:

```
array([0, 4])
```

In [7]:

```
# Replacing the value 4 -->1 for ease of understanding.
data['polarity'] = data['polarity'].replace(4,1)
data.head()
```

Out[7]:

	polarity	Id	date	query	user	text
722084	0	2261382364	Sat Jun 20 20:42:18 PDT 2009	NO_QUERY	AdrianSpeck	Leg hurts can hardly walk
1157616	1	1979198314	Sun May 31 02:01:46 PDT 2009	NO_QUERY	makeupbylinvia	@thoughtcloud Cool that'd be awesome! Thanks....
1064123	1	1964603753	Fri May 29 14:30:11 PDT 2009	NO_QUERY	LeeAnnWalsh	@lilyroseallen sorry to keep saying
1483162	1	2067415190	Sun Jun 07 12:18:11 PDT 2009	NO_QUERY	Wildertamer23	Woke up at 11. I feel awesome
501963	0	2187369967	Mon Jun 15 20:10:55 PDT 2009	NO_QUERY	projvolunteer	Sorry for lack of updates I don't have the int...

In [8]:

```
data.describe()
```

Out[8]:

	polarity	id
count	200000.000000	2.000000e+05
mean	0.499785	1.997999e+09

	polarity	id
std	0.500001	1.939121e+08
min	0.000000	1.467811e+09
25%	0.000000	1.956809e+09
50%	0.000000	2.002005e+09
75%	1.000000	2.176795e+09
max	1.000000	2.329206e+09

In [9]:

```
# check the number of positive vs. negative tagged sentences
positives = data['polarity'][data.polarity == 1 ]
negatives = data['polarity'][data.polarity == 0 ]

print('Total length of the data is:         {}'.format(data.shape[0]))
print('No. of positve tagged sentences is:  {}'.format(len(positives)))
print('No. of negative tagged sentences is: {}'.format(len(negatives)))
```

```
Total length of the data is:         200000
No. of positve tagged sentences is:  99957
No. of negative tagged sentences is: 100043
```

In [10]:

```
# get a word count per of text
def word_count(words):
    return len(words.split())
```

In [11]:

```
# plot word count distribution for both positive and negative

data['word count'] = data['text'].apply(word_count)
p = data['word count'][data.polarity == 1]
n = data['word count'][data.polarity == 0]
plt.figure(figsize=(12,6))
plt.xlim(0,45)
plt.xlabel('Word count')
plt.ylabel('Frequency')
g = plt.hist([p, n], color=['g','r'], alpha=0.5, label=['positive','negative'])
plt.legend(loc='upper right')
```

Out[11]:

```
<matplotlib.legend.Legend at 0x7f08ae898ed0>
```

In [12]:

```
# get common words in training dataset
from collections import Counter
all_words = []
for line in list(data['text']):
    words = line.split()
    for word in words:
      if(len(word)>2):
        all_words.append(word.lower())
```

```
Counter(all_words).most_common(20)
```

Out[12]:

```
[('the', 65031),
 ('and', 37047),
 ('you', 29752),
 ('for', 26809),
 ('have', 17799),
 ('that', 16119),
 ("i'm", 15666),
 ('just', 15646),
 ('but', 15592),
 ('with', 14217),
 ('was', 12847),
 ('not', 12727),
 ('this', 10911),
 ('get', 10224),
 ('good', 9673),
 ('like', 9559),
 ('are', 9541),
 ('all', 9309),
 ('out', 8738),
 ('your', 8105)]
```

Data Processing

In [13]:

```
%matplotlib inline
sns.countplot(data['polarity'])
```

Out[13]:

```
<AxesSubplot:xlabel='polarity', ylabel='count'>
```

In [14]:

```
# Removing the unnecessary columns.
data.drop(['date','query','user','word count'], axis=1, inplace=True)
```

In [15]:

```
data.drop('id', axis=1, inplace=True)
```

In [16]:

```
data.head(10)
```

Out[16]:

	polarity	Text
722084	0	Leg hurts can hardly walk
1157616	1	@thoughtcloud Cool that'd be awesome! Thanks....
1064123	1	@lilyroseallen sorry to keep saying
1483162	1	Woke up at 11. I feel awesome
501963	0	Sorry for lack of updates I don't have the int...
1210662	1	@ home about todo mums hair.....straightening...
1030149	1	@blindmonk ??????.. ? ??? ??? ????? ???????? ???
390866	0	apparently its too early in the day to buy a c...
798832	0	@Tolsonii Ooooooohhhhh how sad.....
702103	0	i'm sad. i'm bored and nobody wants to text me...

In [17]:

```
#Checking if any null values present
(data.isnull().sum() / len(data))*100
```

Out[17]:

```
polarity    0.0
text        0.0
dtype: float64
```

In [18]:

```
#convrting pandas object to a string type
data['text'] = data['text'].astype('str')
```

In [19]:

```
nltk.download('stopwords')
stopword = set(stopwords.words('english'))
print(stopword)
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
{'you', "that'll", 'too', 'that', 'ours', 'as', 'not', 'hadn', 'don', 'needn',
'do', 'them', 'yourself', 'because', 'once', 'more', 'aren', 'by', 'both', 'ho
w', 'mustn', 'under', 'having', 'those', 'which', "mightn't", 'against', 'didn
', 'will', 'won', 've', 'are', 'did', 't', 'above', 'haven', 'few', 'such', 'w
eren', 'ma', 'some', 'each', "needn't", 'll', 'our', 'at', 'in', 'shan', "must
n't", 'their', 'what', 'm', 'yourselves', 'mightn', 'just', 'down', 'further',
"couldn't", 'again', 'there', 'had', 'during', 'where', 'own', 'over', 'y', "w
ouldn't", 'whom', 'if', 'him', 'to', 'same', 'yours', 'up', "you'd", "aren't",
"hasn't", 'itself', "shouldn't", 'wouldn', 'the', 'has', 'other', "she's", 'sh
ould', 'from', 'your', 'no', 'doesn', 'a', "you'll", 'his', 'an', 'can', "it's
", 'who', "haven't", 'about', 'me', 'am', 'doing', 'below', 's', 'does', 'hers
elf', 'hasn', 'when', 'or', 're', 'myself', 'her', 'all', 'nor', "weren't", "d
oesn't", 'now', "won't", 'ourselves', "you're", 'most', 'but', 'very', 'she',
'have', "you've", 'been', 'shouldn', 'isn', 'they', 'for', 'on', 'were', 'wasn
', 'these', "shan't", 'off', 'themselves', 'only', 'was', 'couldn', "hadn't",
'and', 'between', 'himself', 'any', "should've", 'my', "didn't", 'theirs', 'of
', 'here', 'its', 'o', 'while', 'be', 'it', 'before', 'until', "isn't", 'i', '
he', 'hers', 'being', 'then', 'than', "don't", "wasn't", 'so', 'we', 'after',
'why', 'ain', 'into', 'through', 'out', 'this', 'd', 'with', 'is'}
```

In [20]:

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

```
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /usr/share/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[20]:

```
True
```

The Preprocessing steps taken are:

- Lower Casing: Each text is converted to lowercase.
- Removing URLs: Links starting with "http" or "https" or "www" are replaced by "".
- Removing Usernames: Replace @Usernames with word "". (eg: "@XYZ" to "")
- Removing Short Words: Words with length less than 2 are removed.
- Removing Stopwords: Stopwords are the English words which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. (eg: "the", "he", "have")
- Lemmatizing: Lemmatization is the process of converting a word to its base form. (e.g: “wolves” to “wolf”)

In [21]:

```
urlPattern = r"((http://)[^ ]*|(https://)[^ ]*|( www\.)[^ ]*)"
```

```
userPattern = '@[^\s]+'
def process_tweets(tweet):
  # Lower Casing
    tweet = tweet.lower()
    tweet=tweet[1:]
    # Removing all URls
    tweet = re.sub(urlPattern,'',tweet)
    # Removing all @username.
    tweet = re.sub(userPattern,'', tweet)
    #Remove punctuations
    tweet = tweet.translate(str.maketrans("","",string.punctuation))
    #tokenizing words
    tokens = word_tokenize(tweet)
    #Removing Stop Words
    final_tokens = [w for w in tokens if w not in stopword]
    #reducing a word to its word stem
    wordLemm = WordNetLemmatizer()
    finalwords=[]
    for w in final_tokens:
      if len(w)>1:
        word = wordLemm.lemmatize(w)
        finalwords.append(word)
    return ' '.join(finalwords)
```

In [22]:

```
data['processed_tweets'] = data['text'].apply(lambda x: process_tweets(x))
print('Text Preprocessing complete.')
```

```
Text Preprocessing complete.
```

In [23]:

```
data.head(10)
```

Out[23]:

	polarity	Text	processed_tweets
722084	0	Leg hurts can hardly walk	eg hurt hardly walk
1157616	1	@thoughtcloud Cool that'd be awesome! Thanks....	thoughtcloud cool thatd awesome thanks seems l...
1064123	1	@lilyroseallen sorry to keep saying	lilyroseallen sorry keep saying
1483162	1	Woke up at 11. I feel awesome	oke 11 feel awesome
501963	0	Sorry for lack of updates I don't have the int...	orry lack update dont internet home yet

	polarity	Text	processed_tweets
1210662	1	@ home about todo mums hair.....straightening...	home todo mum hairstraightening bottom layer c...
1030149	1	@blindmonk ??????.. ? ??? ??? ????? ???????? ???	blindmonk
390866	0	apparently its too early in the day to buy a c...	pparently early day buy cheeseburger
798832	0	@Tolsonii Oooooohhhhh how sad.....	tolsonii ooooooohhhhh sad
702103	0	i'm sad. i'm bored and nobody wants to text me...	sad im bored nobody want text back today guess...

Analyzing the data

Now we're going to analyse the preprocessed data to get an understanding of it. We'll plot Word Clouds for Positive and Negative tweets from our dataset and see which words occur the most.

Word-Cloud for Negative tweets.

In [24]:

```
plt.figure(figsize = (15,15))
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(d
ata[data.polarity == 0].processed_tweets))
plt.imshow(wc , interpolation = 'bilinear')
```

Out[24]:

```
<matplotlib.image.AxesImage at 0x7f08765ff290>
```

Word-Cloud for Positive tweets.

In [25]:

```
plt.figure(figsize = (15,15))
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(d
ata[data.polarity == 1].processed_tweets))
plt.imshow(wc , interpolation = 'bilinear')
```

Out[25]:

```
<matplotlib.image.AxesImage at 0x7f08765f77d0>
```

Vectorization and Splitting the data

Storing input variable-processes_tweets to X and output variable-polarity to y

In [26]:

```
X = data['processed_tweets'].values
y = data['polarity'].values
```

In [27]:

```
print(X.shape)
print(y.shape)
(200000,)
(200000,)
```

Convert text to word frequency vectors

TF-IDF

This is an acronym than stands for **Term Frequency – Inverse Document** Frequency which are the components of the resulting scores assigned to each word.

- Term Frequency: This summarizes how often a given word appears within a document.
- Inverse Document Frequency: This downscales words that appear a lot across documents.

In [28]:

```
#Convert a collection of raw documents to a matrix of TF-IDF features.
vector = TfidfVectorizer(sublinear_tf=True)
X = vector.fit_transform(X)
print(f'Vector fitted.')
print('No. of feature_words: ', len(vector.get_feature_names()))
Vector fitted.
No. of feature_words:  170137
```

In [29]:

```
print(X.shape)
print(y.shape)
(200000, 170137)
(200000,)
```

Split train and test

The Preprocessed Data is divided into 2 sets of data:

- Training Data: The dataset upon which the model would be trained on. Contains 80% data.
- Test Data: The dataset upon which the model would be tested against. Contains 20% data.

In [30]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random
_state=101)
```

In [31]:

```
print("X_train", X_train.shape)
print("y_train", y_train.shape)
print()
print("X_test", X_test.shape)
```

```
print("y_test", y_test.shape)
X_train (160000, 170137)
y_train (160000,)

X_test (40000, 170137)
y_test (40000,)
```

Model Building

Model evaluating function

In [32]:

```
def model_Evaluate(model):
    #accuracy of model on training data
    acc_train=model.score(X_train, y_train)
    #accuracy of model on test data
    acc_test=model.score(X_test, y_test)

    print('Accuracy of model on training data : {}'.format(acc_train*100))
    print('Accuracy of model on testing data : {} \n'.format(acc_test*100))

    # Predict values for Test dataset
    y_pred = model.predict(X_test)

    # Print the evaluation metrics for the dataset.
    print(classification_report(y_test, y_pred))

    # Compute and plot the Confusion matrix
    cf_matrix = confusion_matrix(y_test, y_pred)

    categories  = ['Negative','Positive']
    group_names = ['True Neg','False Pos', 'False Neg','True Pos']
    group_percentages = ['{0:.2%}'.format(value) for value in cf_matrix.flatten()
/ np.sum(cf_matrix)]

    labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)

    sns.heatmap(cf_matrix, annot = labels, cmap = 'Reds',fmt = '',
                xticklabels = categories, yticklabels = categories)

    plt.xlabel("Predicted values", fontdict = {'size':14}, labelpad = 10)
    plt.ylabel("Actual values"   , fontdict = {'size':14}, labelpad = 10)
    plt.title ("Confusion Matrix", fontdict = {'size':18}, pad = 20)
```

Logistic Regression

In [33]:

```
lg = LogisticRegression()
history=lg.fit(X_train, y_train)
model_Evaluate(lg)
```

Accuracy of model on training data : 83.460625
Accuracy of model on testing data : 77.065

	precision	recall	f1-score	support
0	0.78	0.75	0.77	19967
1	0.76	0.79	0.78	20033
accuracy			0.77	40000
macro avg	0.77	0.77	0.77	40000
weighted avg	0.77	0.77	0.77	40000

Linear SVM

In [34]:

```
svm = LinearSVC()
svm.fit(X_train, y_train)
model_Evaluate(svm)
```

Accuracy of model on training data : 93.191875
Accuracy of model on testing data : 76.0025

	precision	recall	f1-score	support
0	0.77	0.74	0.76	19967
	0.75	0.78	0.76	20033
accuracy			0.76	40000
macro avg	0.76	0.76	0.76	40000
weighted avg	0.76	0.76	0.76	40000

Random Forest

In [35]:

```
rf = RandomForestClassifier(n_estimators = 20, criterion = 'entropy', max_depth=50
)
rf.fit(X_train, y_train)
model_Evaluate(rf)
```

Accuracy of model on training data : 74.87125
Accuracy of model on testing data : 70.1975

	precision	recall	f1-score	support
0	0.74	0.62	0.68	19967
1	0.67	0.78	0.72	20033
accuracy			0.70	40000
macro avg	0.71	0.70	0.70	40000
weighted avg	0.71	0.70	0.70	40000

Naive Bayes

In [36]:

```
nb = BernoulliNB()
nb.fit(X_train, y_train)
model_Evaluate(nb)
```

Accuracy of model on training data : 86.86874999999999
Accuracy of model on testing data : 75.69749999999999

	precision	recall	f1-score	support
0	0.75	0.78	0.76	19967
1	0.77	0.73	0.75	20033
accuracy			0.76	40000
macro avg	0.76	0.76	0.76	40000
weighted avg	0.76	0.76	0.76	40000

Project Conclusion: Sentiment Analysis of Twitter Data

In this project, our team embarked on a journey to perform sentiment analysis on a Twitter dataset. Our primary goal was to understand and implement text preprocessing techniques and apply various machine learning algorithms to classify tweets as positive or negative sentiment.

Text Preprocessing:
We started by recognizing the importance of data quality and cleanliness. In this regard, we performed text preprocessing on the Twitter dataset. This step was crucial to ensure the data was in a suitable format for machine learning. Our preprocessing included:

Text Cleaning: We cleaned the text data by removing special characters, symbols, and irrelevant characters. This step enhanced the quality of the text and reduced noise in our dataset.

Tokenization: Breaking down text into individual words or tokens is a fundamental step in text analysis. Tokenization allowed us to work with the granular components of the text, making it easier to analyze.

Feature Engineering: We considered techniques such as TF-IDF (Term Frequency-Inverse Document Frequency) to convert text data into numerical features, which are suitable for machine learning algorithms. This enabled us to represent text data quantitatively.

Machine Learning Algorithms:
With our preprocessed dataset in hand, we proceeded to implement various machine learning algorithms for sentiment classification. We experimented with the following algorithms:

Logistic Regression: A classic algorithm for binary classification tasks. We used it as a baseline model to gauge the performance of more complex algorithms.

Random Forest Classifier: A powerful ensemble algorithm that can capture complex relationships in the data. It is known for its robustness and versatility.

Support Vector Classifier (SVC): SVC is widely used for classification tasks. It aims to find the hyperplane that best separates data points into different classes.

Naive Bayes: We explored the Naive Bayes algorithm, which is efficient for text classification tasks and is known for its simplicity.

Performance Evaluation:
After implementing these algorithms, we assessed their performance through various evaluation metrics, such as accuracy, precision, recall, F1-score, and ROC curves. This evaluation process allowed us to compare the algorithms and determine their effectiveness in sentiment analysis.

Conclusion:
In conclusion, this project was an insightful exploration of sentiment analysis on Twitter data. Our team successfully executed text preprocessing, a critical step in preparing unstructured text data for machine learning. We implemented a range of machine learning algorithms, each with its unique characteristics, to classify tweets as positive or negative.

Through rigorous performance evaluation, we identified the strengths and weaknesses of each algorithm in handling this specific sentiment classification task. This knowledge will be invaluable for future projects and real-world applications where understanding customer sentiment is crucial. Sentiment analysis has a wide range of applications, from customer feedback analysis to brand reputation management and social media monitoring, and our project marks a significant step in harnessing the power of machine learning in these areas.

As we continue to refine our models and explore advanced techniques, we are excited about the potential of sentiment analysis and its impact on understanding public sentiment and customer experiences. This project serves as a solid foundation for future endeavors in the realm of natural language processing and sentiment analysis.