Comparative analysis of Machine Learning Classification Models to predict Social Media Sentiments using Natural Langauge Processing

This report contains the comparative analysis of Logistic Regression, Naive Baye's Classifier, XGBoost Model and Decision Tree Classifier to predict social media sentiments. Textual analysis of Tweets is done using Natural Language Processing Models; Bag of Words and Term Frequency - Inverse Document Frequency (Tf-IDF) Models

To compare all the models, F1-score as well as Accuracy percentage has been calculated. Steps involved in the project are as follows:

1. Importing Required Libraries
2. Importing Dataset
3. Data Preprocessing
   3.1 Data Cleaning
   3.1.1 Removing user handles from Tweets
   3.1.2 Removing Special Characters, Punctuation, Numbers
   3.1.3 Removing Stop words
   3.2 Tokeninzation
   3.3 Stemming
4. Hashtag, Positive and Negative Sentiment Analysis
5. Creating Bag-of-Words and TF-IDF Models
6. Splitting the dataset into training and test sets
7. Training the Machine Learning Models on Training Set
   7.1 Logistic Regression
   7.2 Naive Baye's
   7.3 XGBoost
   7.4 Decision Tree
8. Creating Comparison matrix for f1-score and accuracy of all the models

RESULT

1. HASTAG ANALYSIS
   After analysing the Hastags, it was found out that LOVE was the mostly used hashtag in positive comments and TRUMP was the mostly used hashtag in Negative comments

2. WORD CLOUD ANALYSIS
   After generating the word Cloud based on positive and negative comments, highest frequency positive comments were - LOVE, GOLD, HAPPY, SMILE, FAMILY, DAD, etc. On the other side, mostly used negative comments had words like - TRUMP, RACISIM, etc.

3. MODEL COMPARISON
   After training the supervised models, and comparing them, it was found out that Logistic Regression has the highest accuracy among all. And comparing NLP models, it was found that TF-IDF gives the better result as compared to Bag-of-Words Model.

Importing Required Libraries

In [302…

```
import pandas as pd
import numpy as np
import nltk
##nltk.download('stopwords')
#nltk.download('punkt')
import string
import matplotlib.pyplot as plt
import re
import warnings
import seaborn as sns
import xgboost as xgb
warnings.filterwarnings("ignore", category=FutureWarning)
```

Importing the Dataset

In [303…

```
data = pd.read_csv("S:/Nitin/Consultancy/Projects/Project 4 - Sentiment Analysis NLP_Python/Data.csv")
data_original = data
Tweet = data['Tweet']
data_original.head(8)
```

Out[303…

	ID	Label	Tweet
0	1	0.0	@user when a father is dysfunctional and is s...
1	2	0.0	@user @user thanks for #lyft credit i can't us...
2	3	0.0	bihday your majesty
3	4	0.0	#model i love u take with u all the time in ...
4	5	0.0	factsguide: society now #motivation
5	6	0.0	[2/2] huge fan fare and big talking before the...
6	7	0.0	@user camping tomorrow @user @user @user @use...
7	8	0.0	the next school year is the year for exams.??...

Data Pre-Processing

In [304…

```
# Removing username handle from the Tweet
Tweet = Tweet.apply(lambda x:re.sub('@[\w]+','',x))

# Removing Special characters, punctuation, numbers from the document
Tweet = Tweet.str.replace("[^a-zA-Z#]", " ")

# Removing Stop Words
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
stop_words = set(stopwords.words('english'))
def remove_stop_words(sentence):
  words = sentence.split() # Split the sentence into individual words
  filtered_words = [word for word in words if word not in stop_words]  # Use a list comprehension to remove stop words
  return ' '.join(filtered_words) # Join the filtered words back into a sentence

Tweet = Tweet.apply(lambda x:remove_stop_words(x))

# Tokenization
Tweet = Tweet.apply(lambda x:x.split())
```

```
# Stemming
from nltk import PorterStemmer
ps = PorterStemmer()
Tweet = Tweet.apply(lambda x: [ps.stem(i) for i in x])
for i in range(len(Tweet)):
    Tweet[i] = ' '.join(Tweet[i])

data['Tidy_Tweets'] = Tweet
data.head(8)
```

Out[304…

	ID	Label	Tweet	Tidy_Tweets
0	1	0.0	@user when a father is dysfunctional and is s...	father dysfunct selfish drag kid dysfunct #run
1	2	0.0	@user @user thanks for #lyft credit i can't us...	thank #lyft credit use caus offer wheelchair v...
2	3	0.0	bihday your majesty	bihday majesti
3	4	0.0	#model i love u take with u all the time in ...	#model love u take u time ur
4	5	0.0	factsguide: society now #motivation	factsguid societi #motiv
5	6	0.0	[2/2] huge fan fare and big talking before the...	huge fan fare big talk leav chao pay disput ge...
6	7	0.0	@user camping tomorrow @user @user @user @use...	camp tomorrow danni
7	8	0.0	the next school year is the year for exams.😂...	next school year year exam think #school #exam...

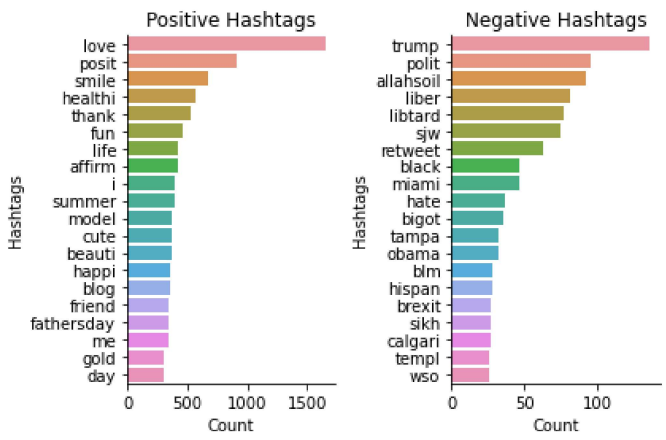
In [305…

```
data.to_csv("S:/Nitin/Consultancy/Projects/Project 4 - Sentiment Analysis NLP_Python/positive.csv")
train = data[:31962]
test = data[31963:]
```

Analysing Hashtags in Comments

In [306…

```
# Finding hashtag data to know what people are thinking about
data['Hashtag'] = data['Tidy_Tweets'].apply(lambda x:re.findall(r'#(\w+)',x))
HashTags = data['Tidy_Tweets'].apply(lambda x:re.findall(r'#(\w+)',x))
ht_pos = data['Hashtag'][data['Label']==0]
ht_neg = data['Hashtag'][data['Label']==1]
ht_pos = sum(ht_pos,[])
ht_neg = sum(ht_neg,[])
word_freq_positive = nltk.FreqDist(ht_pos)
word_freq_negative = nltk.FreqDist(ht_neg)
df_positive = pd.DataFrame({'Hashtags':list(word_freq_positive.keys()),'Count':list(word_freq_positive.values())})
df_negative = pd.DataFrame({'Hashtags':list(word_freq_negative.keys()),'Count':list(word_freq_negative.values())})
df_positive_plot = df_positive.nlargest(20,columns='Count')
df_negative_plot = df_negative.nlargest(20,columns='Count')
fig, [ax1, ax2] = plt.subplots(1,2,tight_layout=True)
ax1.set_title('Positive Hashtags')
ax2.set_title('Negative Hashtags')
sns.barplot(data=df_positive_plot,y='Hashtags',x='Count', ax=ax1)
sns.barplot(data=df_negative_plot,y='Hashtags',x='Count', ax=ax2)
sns.despine()
```



Creating Bag of Word model, TF-IDF model

In [307…

```
# Bag of Words Model
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features = 1000, stop_words = 'english')
BOW = cv.fit_transform(data['Tidy_Tweets'])
df_BOW = pd.DataFrame(BOW.todense())
df_BOW.head()
```

Out[307…

	0	1	2	3	4	5	6	7	8	9	…	990	991	992	993	994	995	996	997	998	999
0	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0

5 rows × 1000 columns

In [308…

```
# TF-IDF Model
from sklearn.feature_extraction.text import TfidfVectorizer
tf = TfidfVectorizer(max_features = 1000, stop_words = 'english')
TF_IDF = tf.fit_transform(data['Tidy_Tweets'])
df_TFIDF = pd.DataFrame(TF_IDF.todense())
df_TFIDF.head()
```

Out[308…

	0	1	2	3	4	5	6	7	8	9	…	990	991	992	993	994	995	996	997	998	999
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 1000 columns

Splitting the dataset into Train & Test sets

In [309…

```
train_BOW = BOW[:31962]
train_BOW.todense()

from sklearn.model_selection import train_test_split
x_train_BOW, x_test_BOW, y_train_BOW, y_test_bow = train_test_split(train_BOW,train['Label'],test_size=0.3,random_state=2)
```

In [310…

```
train_TFIDF = TF_IDF[:31962]
train_TFIDF.todense()

from sklearn.model_selection import train_test_split
x_train_TFIDF, x_test_TFIDF, y_train_TFIDF, y_test_TFIDF = train_test_split(train_TFIDF,train['Label'],test_size=0.3,random_state=2)
```

Training the Machine Learning Models on Training Set

Logistic Regression

In [311…

```
# For Bag of Words
from sklearn.linear_model import LogisticRegression
Log_Reg = LogisticRegression(random_state=0,solver='lbfgs')
Log_Reg.fit(x_train_BOW,y_train_BOW)
prediction_bow = Log_Reg.predict_proba(x_test_BOW)

from sklearn.metrics import f1_score, confusion_matrix, accuracy_score
prediction_int = prediction_bow[:,1]>=0.3
prediction_int = prediction_int.astype(np.int) # converting the results to integer type
log_bow = f1_score(y_test_bow, prediction_int) # calculating f1 score
cm = confusion_matrix(y_test_bow, prediction_int) # calculating confusion matrix
log_bow_acc = accuracy_score(y_test_bow, prediction_int) # calculating accuracy score
```

In [312…

```
# For TF-IDF
Log_Reg = LogisticRegression(random_state=0,solver='lbfgs')
Log_Reg.fit(x_train_TFIDF,y_train_TFIDF)
prediction_TFIDF = Log_Reg.predict_proba(x_test_TFIDF)
prediction_int = prediction_TFIDF[:,1]>=0.3
prediction_int = prediction_int.astype(np.int) # converting the results to integer type
log_TFIDF = f1_score(y_test_TFIDF, prediction_int) # calculating f1 score
log_TFIDF_acc = accuracy_score(y_test_TFIDF, prediction_int)
```

Naive-Baye's

In [313…

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
# For Bag of Words
x_train_BOW = x_train_BOW.toarray()
x_test_BOW = x_test_BOW.toarray()
```

In [314…

```
classifier.fit(x_train_BOW, y_train_BOW)
prediction_BOW = classifier.predict_proba(x_test_BOW)
prediction_int = prediction_BOW[:,1]>=0.3
prediction_int = prediction_int.astype(np.int)
NB_BOW = f1_score(y_test_bow,prediction_int)
NB_bow_acc = accuracy_score(y_test_bow, prediction_int)
```

In [315…

```
# For TF-IDF
x_train_TFIDF = x_train_TFIDF.toarray()
x_test_TFIDF = x_test_TFIDF.toarray()
```

In [316…

```
classifier.fit(x_train_TFIDF,y_train_TFIDF)
prediction_TFIDF = classifier.predict_proba(x_test_TFIDF)
prediction_int = prediction_TFIDF[:,1]>=0.3
prediction_int = prediction_int.astype(np.int)
NB_TFIDF = f1_score(y_test_TFIDF, prediction_int)
NB_TFIDF_acc = accuracy_score(y_test_TFIDF, prediction_int)
```

XGBoost

In [317…

```
from xgboost import XGBClassifier

# For Bag of Words
model_bow = XGBClassifier(random_state=22,learning_rate=0.9)
model_bow.fit(x_train_BOW, y_train_BOW)
xgb = model_bow.predict_proba(x_test_BOW)
xgb=xgb[:,1]>=0.3
xgb_int=xgb.astype(np.int)
xgb_bow=f1_score(y_test_bow,xgb_int)
xgb_bow_acc = accuracy_score(y_test_bow, xgb_int)
```

In [318…

```
# For TF-IDF
model_TFIDF = XGBClassifier(random_state=22,learning_rate=0.9)
model_TFIDF.fit(x_train_TFIDF, y_train_TFIDF)
xgb = model_TFIDF.predict_proba(x_test_TFIDF)
xgb=xgb[:,1]>=0.3
xgb_int=xgb.astype(np.int)
xgb_TFIDF=f1_score(y_test_TFIDF,xgb_int)
xgb_TFIDF_acc = accuracy_score(y_test_TFIDF, xgb_int)
```

Decision Tree

In [319…

```
from sklearn.tree import DecisionTreeClassifier
DT = DecisionTreeClassifier(criterion='entropy', random_state=1)

# For Bag of Words
DT.fit(x_train_BOW,y_train_BOW)
DT_bow = DT.predict_proba(x_test_BOW)
DT_bow = DT_bow[:,1]>=0.3
DT_int_bow = DT_bow.astype(np.int)
DT_f1_BOW = f1_score(y_test_bow,DT_int_bow)
DT_bow_acc = accuracy_score(y_test_bow, DT_int_bow)
```

In [320…

```
# For TF-IDF
DT.fit(x_train_TFIDF,y_train_TFIDF)
DT_TFIDF = DT.predict_proba(x_test_TFIDF)
```

```
DT_TFIDF = DT_TFIDF[:,1]>=0.3
DT_int_TFIDF = DT_TFIDF.astype(np.int)
DT_f1_TFIDF = f1_score(y_test_TFIDF,DT_int_TFIDF)
DT_TFIDF_acc = accuracy_score(y_test_TFIDF, DT_int_TFIDF)
```

In [321…

```
col_labels = ['Logistic Regression', "Naive Baye's", 'XGBoost', 'Decision Tree']
row_labels = ['Bag of Words', 'TF-IDF']
DF_Values = [[log_bow, NB_BOW, xgb_bow, DT_f1_BOW],[log_TFIDF, NB_TFIDF, xgb_TFIDF, DT_f1_TFIDF]]
f1_score_DF = pd.DataFrame(data=DF_Values, index=row_labels, columns = col_labels)
f1_score_DF
```

Out[321…

	Logistic Regression	Naive Baye's	XGBoost	Decision Tree
Bag of Words	0.570492	0.213269	0.572990	0.488835
TF-IDF	0.576300	0.219512	0.560995	0.538279

In [322…

```
col_labels = ['Logistic Regression', "Naive Baye's", 'XGBoost', 'Decision Tree']
row_labels = ['Bag of Words', 'TF-IDF']
DF_Values = [[log_bow_acc, NB_bow_acc, xgb_bow_acc, DT_bow_acc],[log_TFIDF_acc, NB_TFIDF_acc, xgb_TFIDF_acc, DT_TFIDF_acc]]
accuracy_score_DF = pd.DataFrame(data=DF_Values, index=row_labels, columns = col_labels)
accuracy_score_DF
```

Out[322…

	Logistic Regression	Naive Baye's	XGBoost	Decision Tree
Bag of Words	0.945354	0.521431	0.942955	0.911670
TF-IDF	0.948170	0.539472	0.941078	0.938993