OUTSTANDING PROJECT 3

Mudit Mathur mm7692@srmist.edu.in

Problem: Twitter Based Gender Classification Model - Natural Language Processing and Neural Network

DESCRIPTION OF DATASET:

Dataset contains gender, gender:confidence, text, name, favourite_no, profile_yn, description, created, timestamp, etc.

Total number of Entries are 20050 and Number of Features are 26.

LIBRARIES USED:

Pandas, Numpy, Matplotlib, Seaborn, Sci-kit learn

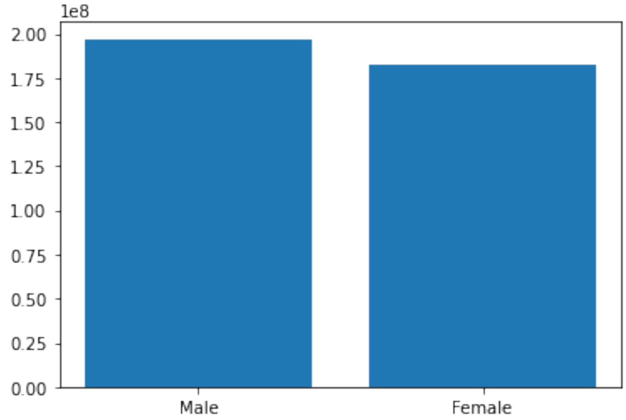
INDEPENDENT AND DEPENDENT FEATURES:

Independent Features: text, gender:confidence
Dependent / Target Feature: gender

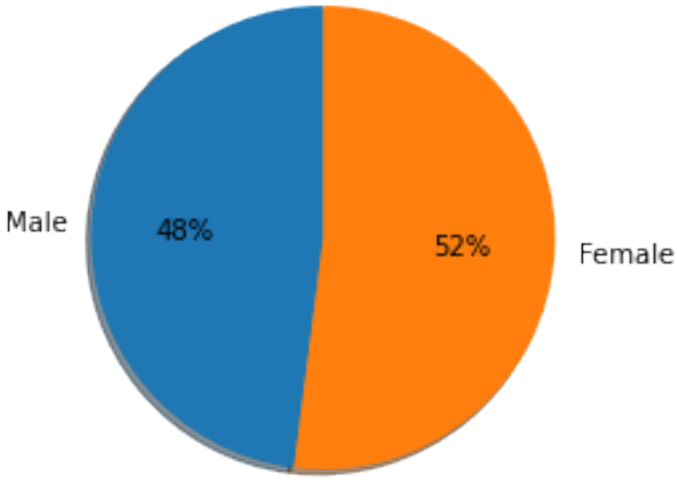
DATA CLEANING:

1. Feature selection of text, gender and gender:confidence.
2. Dropping the NULL values.
3. Taking all the rows where gender:confidence > 0.9 .
4. Resetting Index for the dataset to avoid index issues.
5. Converting gender to 0 or 1 from categorical values.
6. Converting gender to type integer.
7. Importing Stopwords from nltk and removing them from text.
8. Applying CountVectorizer on text.

EXPLORATORY DATA ANALYSIS (EDA):



Distribution of Tweet Counts : Genders



Gender Distribution on Twitter Dataset

MACHINE LEARNING AND ALGORITHMS:

1. Logistic Regression
2. Multinomial NB
3. Random Forest
4. KNN
5. SVM
6. Gradient Boosting
7. Neural Network (Multilayer Perceptron)

LOGISTIC REGRESSION (f1_score=63.36%)

Logistic Regression

In [196]:
```
from sklearn.linear_model import LogisticRegression
```

In [197]:
```
lr=LogisticRegression()
```

In [198]:
```
lr.fit(X_train,Y_train)
```

Out[198]:
```
LogisticRegression()
```

In [199]:
```
predlr=lr.predict(X_test)
```

In [200]:
```
print(classification_report(Y_test,predlr))
print('\n')
print(confusion_matrix(Y_test,predlr))
print('\n')
print(f1_score(Y_test,predlr))
```

```
              precision    recall  f1-score   support

           0       0.55      0.51      0.53      1362
           1       0.62      0.65      0.63      1645

    accuracy                           0.59      3007
   macro avg       0.58      0.58      0.58      3007
weighted avg       0.59      0.59      0.59      3007


[[ 700  662]
 [ 575 1070]]


0.6336985490079953
```

MULTINOMIAL NB(f1_score=66.94%)

Multinomial NB

In [176]:
```
from sklearn.naive_bayes import MultinomialNB
```

In [177]:
```
mb=MultinomialNB()
```

In [178]:
```
mb.fit(X_train,Y_train)
```

Out[178]:
```
MultinomialNB()
```

In [179]:
```
predmb=mb.predict(X_test)
```

In [180]:
```
print(classification_report(Y_test,predmb))
print('\n')
print(confusion_matrix(Y_test,predmb))
print('\n')
print(f1_score(Y_test,predmb))
```

```
              precision    recall  f1-score   support

           0       0.58      0.47      0.52      1362
           1       0.62      0.73      0.67      1645

    accuracy                           0.61      3007
   macro avg       0.60      0.60      0.59      3007
weighted avg       0.60      0.61      0.60      3007


[[ 636  726]
 [ 452 1193]]


0.6694725028058361
```

RANDOM FOREST(f1_score=64.40%)

Random Forest

In [181]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [182]:
```
rf=RandomForestClassifier()
```

In [183]:
```
rf.fit(X_train,Y_train)
```

Out[183]:
```
RandomForestClassifier()
```

In [184]:
```
predrf=rf.predict(X_test)
```

In [185]:
```
print(classification_report(Y_test,predrf))
print('\n')
print(confusion_matrix(Y_test,predrf))
print('\n')
print(f1_score(Y_test,predrf))
```

```
              precision    recall  f1-score   support

           0       0.55      0.47      0.51      1362
           1       0.61      0.68      0.64      1645

    accuracy                           0.59      3007
   macro avg       0.58      0.58      0.58      3007
weighted avg       0.58      0.59      0.58      3007

[[ 645  717]
 [ 523 1122]]

0.6440872560275545
```

KNN(f1_score=56.66%)

KNN

In [191]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [192]:
```
knn=KNeighborsClassifier()
```

In [193]:
```
knn.fit(X_train,Y_train)
```

Out[193]:
```
KNeighborsClassifier()
```

In [194]:
```
predknn=knn.predict(X_test)
```

In [195]:
```
print(classification_report(Y_test,predknn))
print('\n')
print(confusion_matrix(Y_test,predknn))
print('\n')
print(f1_score(Y_test,predknn))
```

```
              precision    recall  f1-score   support

           0       0.46      0.43      0.44      1362
           1       0.55      0.58      0.57      1645

    accuracy                           0.51      3007
   macro avg       0.50      0.50      0.50      3007
weighted avg       0.51      0.51      0.51      3007



[[581 781]
 [686 959]]


0.5666174298375185
```

SVM (f1_score=66.72%)

Support Vector Machine

In [169]:
```
from sklearn.svm import SVC
```

In [170]:
```
sv=SVC()
```

In [171]:
```
sv.fit(X_train,Y_train)
```

Out[171]:
```
SVC()
```

In [172]:
```
predsv=sv.predict(X_test)
```

In [173]:
```
print(classification_report(Y_test,predsv))
```

```
              precision    recall  f1-score   support

           0       0.58      0.45      0.50      1362
           1       0.61      0.73      0.67      1645

    accuracy                           0.60      3007
   macro avg       0.60      0.59      0.59      3007
weighted avg       0.60      0.60      0.59      3007
```

In [174]:
```
print(confusion_matrix(Y_test,predsv))
```

```
[[ 608  754]
 [ 444 1201]]
```

In [175]:
```
print(f1_score(Y_test,predsv))
```

```
0.6672222222222222
```

GRADIENT BOOSTING (f1_score=69.92%)

Gradient Boosting Classifier

In [186]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [187]:
```
gb=GradientBoostingClassifier()
```

In [188]:
```
gb.fit(X_train,Y_train)
```

Out[188]:
```
GradientBoostingClassifier()
```

In [189]:
```
predgb=gb.predict(X_test)
```

In [190]:
```
print(classification_report(Y_test,predgb))
print('\n')
print(confusion_matrix(Y_test,predgb))
print('\n')
print(f1_score(Y_test,predgb))
```

```
              precision    recall  f1-score   support

           0       0.60      0.17      0.26      1362
           1       0.57      0.91      0.70      1645

    accuracy                           0.57      3007
   macro avg       0.58      0.54      0.48      3007
weighted avg       0.58      0.57      0.50      3007



[[ 226 1136]
 [ 150 1495]]


0.6992516370439663
```

NEURAL NETWORK (MLP) (f1_score=61.22%)

Neural Network

In [201]:
```
import sklearn.neural_network
```

In [202]:
```
from sklearn.neural_network import MLPClassifier
```

In [203]:
```
clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, Y_train)
```

In [204]:
```
prednn=clf.predict(X_test)
```

In [205]:
```
print(classification_report(Y_test,prednn))
print('\n')
print(confusion_matrix(Y_test,prednn))
```

```
              precision    recall  f1-score   support

           0       0.53      0.51      0.52      1362
           1       0.60      0.62      0.61      1645

    accuracy                           0.57      3007
   macro avg       0.57      0.56      0.56      3007
weighted avg       0.57      0.57      0.57      3007



[[ 693  669]
 [ 624 1021]]
```

In [206]:
```
f1_score(Y_test,prednn)
```

Out[206]:
```
0.6122938530734633
```

WHICH ACTIVATION FUNCTION WAS CHOOSEN AND WHY?

Multilayer perceptron (MLP) uses a ReLU activation function.
The **rectified linear activation function** or **ReLU** for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

WHICH OPTIMIZER WAS CHOOSEN AND WHY?

This model optimizes the log-loss function using LBFGS or stochastic gradient descent.
It is widely used [in machine learning] because it is more memory-efficient than plain vanilla BFGS.

WHICH NEURAL NETWORK AND WHY? DESCRIBE YOUR NEURAL STRUCTURE.

Multilayer Perceptrons, or MLPs for short, are the classical type of neural network.
They are comprised of one or more layers of neurons. Data is fed to the input layer, there may be one or more hidden layers providing levels of abstraction, and predictions are made on the output layer, also called the visible layer.
MLPs are suitable for classification prediction problems where inputs are assigned a class or label.

CONCLUSION:

The Best Machine Learning Model is **Gradient Boosting Classifier with accuracy of 69.99%**