FLIP ROBO

RATINGS PREDICTION

Submitted by:

ARUN KUMAR M

INTRODUCTION

Business Problem Framing :

We have a client who has a website where people write different reviews for technical products. Now they are adding a new feature to their website i.e. the reviewer will have to add stars (rating) as well with the review. The rating is out 5 stars and it only has 5 options available 1 star, 2 stars, 3 stars, 4 stars, 5 stars. Now they want to predict ratings for the reviews which were written in the past and they don't have a rating. So, we have to build an application which can predict the rating by seeing the review.

We have a client who has a website where people write different reviews for technical products.

Now they are adding a new feature to their website i.e. The reviewer will have to add stars (rating)

As well with the review. The rating is out 5 stars and it only has 5 options available 1 star, 2 stars, 3 stars, 4 stars, 5 stars. Now they want to predict ratings for the reviews which were written in the past and they don't have a rating. So, we have to build an application which can predict the rating by seeing the review.

DATA COLLECTION:

Around 27 products have been scrapped for getting the data required for building the Machine Learning Model. We have scrapped from both Amazon and Flip-kart for the diversity in the data-set.

These data's are collected separately and then combined into a single file.

```
import pandas as pd
import glob
import os

file_path = r'E:\pyhton\NLP_Ratings_csv'
all_files = glob.glob(file_path+'/*.csv')
df_all_file = (pd.read_csv(f, sep=',') for f in all_files)
df_merged   = pd.concat(df_all_file, ignore_index=True)
df_merged.to_csv( "NLP_Ratings_csv_merged.csv")
```

```
df_merged['rating'].value_counts()
```

```
5.0    64927
4.0    23603
1.0    11704
3.0     8994
2.0     6151
Name: rating, dtype: int64
```

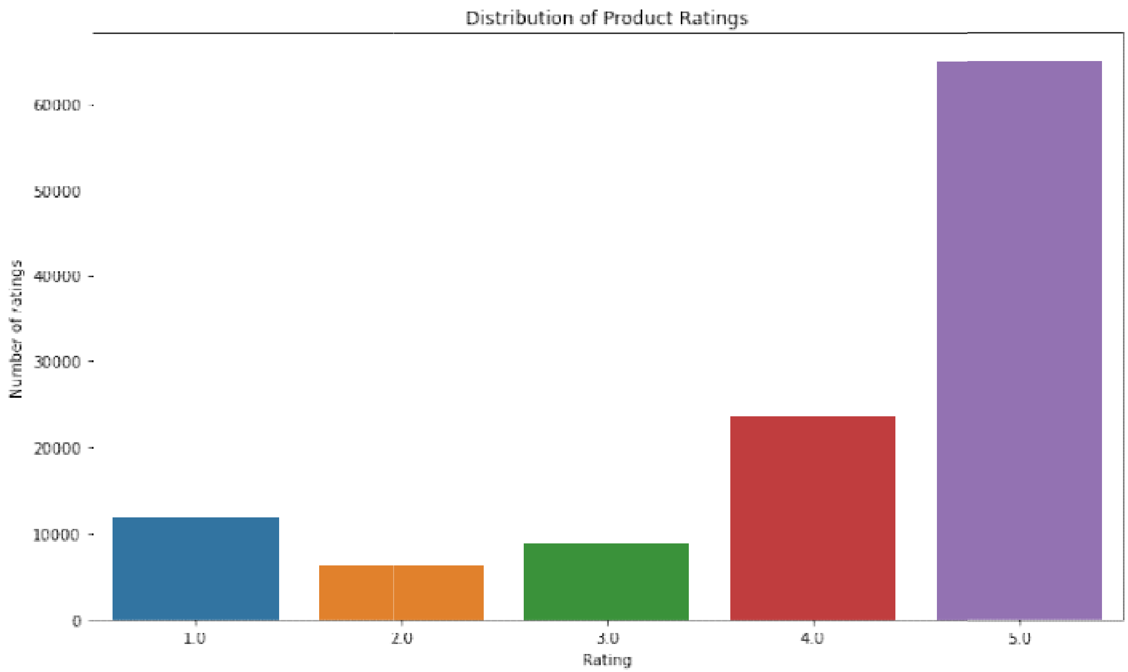
```
df_merged.shape
```

```
(115379, 2)
```

We could see that the data-set contains 115379 entries. Also, we can see that the data is imbalanced. Ratings counts differ for each rating.

```
df_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115379 entries, 0 to 115378
Data columns (total 2 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   title    115373 non-null  object
 1   rating   115379 non-null  float64
dtypes: float64(1), object(1)
memory usage: 1.8+ MB
```



BALANCING THE DATA-SET:

Now we will try to balance the unbalanced data-set. Firstly, we will try to check and remove the null values in the data-set.

```
df.isnull().sum()
```

```
Unnamed: 0    0
title         6
rating        0
dtype: int64
```

```
df.dropna(inplace=True)
df.isnull().sum()
```

```
Unnamed: 0    0
title         0
rating        0
dtype: int64
```

Since the rating-2 has 6151 entries. We will try to equally divide all the ratings to 6151 nos.

```
rating5 = df[df['rating']==5]
rating4 = df[df['rating']==4]
rating3 = df[df['rating']==3]
rating2 = df[df['rating']==2]
rating1 = df[df['rating']==1]
```

```
rating5.info()
rating4.info()
rating3.info()
rating2.info()
rating1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64925 entries, 0 to 64924
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   title   64925 non-null  object
 1   rating  64925 non-null  float64
dtypes: float64(1), object(1)
memory usage: 1.5+ MB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23601 entries, 64925 to 88525
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   title   23601 non-null  object
 1   rating  23601 non-null  float64
dtypes: float64(1), object(1)
memory usage: 553.1+ KB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8994 entries, 88526 to 97519
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   title   8994 non-null   object
 1   rating  8994 non-null   float64
dtypes: float64(1), object(1)
memory usage: 210.8+ KB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6151 entries, 97520 to 103670
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   title   6151 non-null   object
 1   rating  6151 non-null   float64
dtypes: float64(1), object(1)
memory usage: 144.2+ KB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11702 entries, 103671 to 115372
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   title   11702 non-null  object
 1   rating  11702 non-null  float64
dtypes: float64(1), object(1)
memory usage: 274.3+ KB
```

Now we will try to divide them equally to 6151 nos. each.

```
dft=pd.concat([rating1[0:6151], rating2[0:6151], rating3[0:6151], rating4[0:6151], rating5[0:6151]])
```

```
dft.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30755 entries, 103671 to 6150
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   title   30755 non-null  object
 1   rating  30755 non-null  float64
dtypes: float64(1), object(1)
memory usage: 720.8+ KB
```

```
dft['rating'].value_counts()
```

```
5.0    6151
4.0    6151
3.0    6151
2.0    6151
1.0    6151
Name: rating, dtype: int64
```

```
dft.shape
```

```
(30755, 2)
```

DATA-SET PREPROCESSING:

In this data pre-processing we will try to change the dataset to lower-case. Then we will remove the spaces, email address, web address, signs, phone number, numbers, and punctuation.

```
dft['title'] = dft['title'].str.lower()#lower case
dft['title'] = dft['title'].str.replace(r'^.+@[^\.].*\.[a-z]{2,}$', 'email')#remove email address
dft['title'] = dft['title'].str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$', 'web')#remove webaddress
dft['title'] = dft['title'].str.replace(r'£|\$|₹', 'signs')#remove signs
dft['title'] = dft['title'].str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$', 'ph_number')#remove phonenumber
dft['title'] = dft['title'].str.replace(r'\d+(\.\d+)?', 'number')#remove numbers
dft['title'] = dft['title'].str.replace(r'[^\w\d\s]', ' ')#remove punctuation
dft['title'] = dft['title'].str.replace(r'\s+', ' ')#remove spaces
dft['title'] = dft['title'].str.replace(r'^\s+|\s+?$', '')
```

We will also now use stop words to remove some meaningless words. This will help the data-set to turn into a simpler version easy for the ML Model.

```
#Removing Stop_Words
stop_words = set(stopwords.words('english') + ['u','ur','im','doin','i','so', 'ü', 'â', 'ur', '4', '2', 'dont', 'doin', 'ure','RE
dft['title'] = dft['title'].apply(lambda x: ' '.join(term for term in x.split() if term not in stop_words))
dft.head(10)
```

We will now lemmatize the dataset.

```
#Lemmetizing
WL = WordNetLemmatizer()
dft['title'] = dft['title'].apply(lambda x: ' '.join(WL.lemmatize(i) for i in x.split()))
dft.head(10)
```

We will now use word cloud to visualize the sense of words reciprocating regularly in the data-set for each rating.

All the ratings are serialised as rating-1, rating-2, rating-3, rating-4,rating-4.

Rating-2

Rating-3

Ratings-3

Ratings-4

Feature Extraction:

```
tfidf = TfidfVectorizer(max_features = 20000, ngram_range = (1,5), analyzer = 'char')
```

```
x = tfidf.fit_transform(dft['title'])
y = dft['rating']
```

```
#Creating train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=0,test_size=.20)
```

```
x.shape,y.shape
```

```
((30755, 20000), (30755,))
```

```
x_train.shape
```

```
(24604, 20000)
```

```
y_train.shape
```

```
(24604,)
```

This will convert the data-set to vector values.

MODEL BUILIDING :

```
#Importing all the model library

from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB

#Importing Boosting models
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier

#Importing error metrics
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,roc_curve,auc
from sklearn.model_selection import GridSearchCV,cross_val_score
```

```
KNN=KNeighborsClassifier(n_neighbors=6)
DT=DecisionTreeClassifier(random_state=6)
RF=RandomForestClassifier()
ADA=AdaBoostClassifier()
MNB=MultinomialNB()
GBC=GradientBoostingClassifier()
BC=BaggingClassifier()
ETC=ExtraTreesClassifier()
```

```
models= []
models.append(('KNeighborsClassifier', KNN))
models.append(('DecisionTreeClassifier', DT))
models.append(('RandomForestClassifier', RF))
models.append(('AdaBoostClassifier', ADA))
models.append(('MultinomialNB', MNB))
models.append(('GradientBoostingClassifier', GBC))
models.append(('BaggingClassifier', BC))
models.append(('ExtraTreesClassifier', ETC))
```

```
Model= []
score= []
cvs=[]
for name,model in models:

    Model.append(name)
    model.fit(x_train,y_train)
    print(model)
    pre=model.predict(x_test)
    AS=accuracy_score(y_test,pre)
    score.append(AS*100)
    sc= cross_val_score(model, x, y, cv=10, scoring='accuracy').mean()
    cvs.append(sc*100)
    print('\n')
    print('classification_report\n',classification_report(y_test,pre))
    print('\n')
```

```
KNeighborsClassifier(n_neighbors=6)


classification_report
               precision    recall  f1-score   support

         1.0       0.72      0.71      0.71      1253
         2.0       0.60      0.65      0.62      1263
         3.0       0.68      0.76      0.72      1222
         4.0       0.86      0.72      0.78      1199
         5.0       0.98      0.94      0.96      1214

    accuracy                           0.75      6151
   macro avg       0.77      0.76      0.76      6151
weighted avg       0.76      0.75      0.76      6151
```

```
DecisionTreeClassifier(random_state=6)


classification_report
               precision    recall  f1-score   support

         1.0       0.69      0.71      0.70      1253
         2.0       0.63      0.60      0.61      1263
         3.0       0.69      0.77      0.73      1222
         4.0       0.79      0.75      0.77      1199
         5.0       0.98      0.94      0.96      1214

    accuracy                           0.75      6151
   macro avg       0.76      0.75      0.75      6151
weighted avg       0.76      0.75      0.75      6151


RandomForestClassifier()


classification_report
               precision    recall  f1-score   support

         1.0       0.73      0.77      0.75      1253
         2.0       0.64      0.67      0.66      1263
         3.0       0.74      0.75      0.74      1222
         4.0       0.84      0.77      0.80      1199
         5.0       0.99      0.94      0.96      1214

    accuracy                           0.78      6151
   macro avg       0.78      0.78      0.78      6151
weighted avg       0.78      0.78      0.78      6151
```

```
AdaBoostClassifier()


classification_report
              precision    recall  f1-score   support

         1.0       0.47      0.72      0.57      1253
         2.0       0.52      0.39      0.45      1263
         3.0       0.76      0.67      0.71      1222
         4.0       0.66      0.64      0.65      1199
         5.0       0.98      0.82      0.89      1214

    accuracy                           0.65      6151
   macro avg       0.68      0.65      0.65      6151
weighted avg       0.67      0.65      0.65      6151
```

```
MultinomialNB()


classification_report
              precision    recall  f1-score   support

         1.0       0.72      0.76      0.74      1253
         2.0       0.66      0.60      0.63      1263
         3.0       0.82      0.65      0.73      1222
         4.0       0.69      0.84      0.76      1199
         5.0       0.91      0.94      0.93      1214

    accuracy                           0.76      6151
   macro avg       0.76      0.76      0.75      6151
weighted avg       0.76      0.76      0.75      6151
```

```
GradientBoostingClassifier()


classification_report
              precision    recall  f1-score   support

         1.0       0.72      0.76      0.74      1253
         2.0       0.61      0.65      0.63      1263
         3.0       0.74      0.74      0.74      1222
         4.0       0.83      0.76      0.79      1199
         5.0       0.98      0.94      0.96      1214

    accuracy                           0.77      6151
   macro avg       0.78      0.77      0.77      6151
weighted avg       0.77      0.77      0.77      6151
```

```
BaggingClassifier()


classification_report
              precision    recall  f1-score   support

         1.0       0.70      0.76      0.73      1253
         2.0       0.63      0.64      0.64      1263
         3.0       0.72      0.74      0.73      1222
         4.0       0.83      0.76      0.79      1199
         5.0       0.98      0.94      0.96      1214

    accuracy                           0.77      6151
   macro avg       0.77      0.77      0.77      6151
weighted avg       0.77      0.77      0.77      6151


ExtraTreesClassifier()


classification_report
              precision    recall  f1-score   support

         1.0       0.72      0.78      0.75      1253
         2.0       0.66      0.67      0.66      1263
         3.0       0.74      0.75      0.74      1222
         4.0       0.84      0.77      0.80      1199
         5.0       0.98      0.94      0.96      1214

    accuracy                           0.78      6151
   macro avg       0.79      0.78      0.78      6151
weighted avg       0.78      0.78      0.78      6151
```

```python
result = pd.DataFrame({'Model': Model, 'Accuracy_score': score,'Cross_val_score': cvs})
result
```

	Model	Accuracy_score	Cross_val_score
0	KNeighborsClassifier	75.483661	74.238386
1	DecisionTreeClassifier	75.288571	74.853027
2	RandomForestClassifier	77.792229	76.852700
3	AdaBoostClassifier	64.916274	63.693809
4	MultinomialNB	75.581206	74.602510
5	GradientBoostingClassifier	76.898065	75.903227
6	BaggingClassifier	76.735490	75.958532
7	ExtraTreesClassifier	78.052349	77.054287

HYPERPARAMETER TUNING

```
#RandomForestClassifier
parameters={'n_estimators':[1,10,100]}
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rfc=RandomForestClassifier(random_state=96)
rfc=GridSearchCV(rfc,parameters,cv=3,scoring='accuracy')
rfc.fit(x_train,y_train)
print(rfc.best_params_)
print(rfc.best_score_)
```

```
{'n_estimators': 100}
0.7665422743451303
```

```
#Using the best parameters obtained
gbc=GradientBoostingClassifier(random_state=96,n_estimators=100)
gbc.fit(x_train,y_train)
pred=gbc.predict(x_test)
print("Accuracy score: ",accuracy_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(gbc,x,y,cv=3,scoring='accuracy').mean()*100)
print('Classification report: \n')
print(classification_report(y_test,pred))
print('Confusion matrix: \n')
print(confusion_matrix(y_test,pred))
```

```
Accuracy score:  76.84929279791903
Cross validation score:  75.45442213549543
Classification report:

              precision    recall  f1-score   support

         1.0       0.72      0.75      0.74      1253
         2.0       0.61      0.65      0.63      1263
         3.0       0.74      0.74      0.74      1222
         4.0       0.83      0.76      0.80      1199
         5.0       0.98      0.94      0.96      1214

    accuracy                           0.77      6151
   macro avg       0.78      0.77      0.77      6151
weighted avg       0.77      0.77      0.77      6151
```

Final Model :

```
rating_prediciton=rfc.predict(x)
Ratings_Prediction = pd.DataFrame({'Prediciton' : rating_prediciton})
Ratings_Prediction
```

	Prediciton
0	5.0
1	5.0
2	5.0
3	5.0
4	5.0
5	5.0
6	5.0
7	5.0
8	5.0
9	5.0
10	5.0

Saving the Model

```
#Saving the model
import pickle
filename='NLP_Ratings_Prediction.pkl'
pickle.dump(rfc,open(filename,'wb'))
```

END

CONCLUSION:

Thus we have predicted ratings with help of the Machine Learning Model. The Machine Learning Model that has been selected is **Random forest Classifier** because of its good accuracy score and cross value score.