

▼ Description of Data

The dataset that we are using is that of rotten tomatoes reviews and their classification of either a positive review or negative review. The value of 1 indicates that a review is a positive review and a value of 0 indicates that the review is a negative review. Looking at the graph the data suggests that it is an even split of 5331 positive and negative reviews. before we begin training the models however, we must make sure the data is shuffled to allow for the model to have a good mix of positive and negative reviews allowing it to learn.

```
import urllib
import os
import pandas as pd
import pathlib
import zipfile

drive_url = 'https://utdallas.box.com/shared/static/otw3umvkginpto2nckwxqy6lr9cx9jsf.zip'
file_name = 'Rotten_tomatoes.zip'

urllib.request.urlretrieve(drive_url, file_name)


os.listdir()

['.config', 'Rotten_tomatoes.zip', 'sample_data']

zip_ref = zipfile.ZipFile("Rotten_tomatoes.zip", "r")
zip_ref.extractall()
zip_ref.close()

RT_file = '/content/data_rt.csv'

data = pd.read_csv(RT_file, encoding= 'unicode_escape')
df = pd.DataFrame(data)
df
```

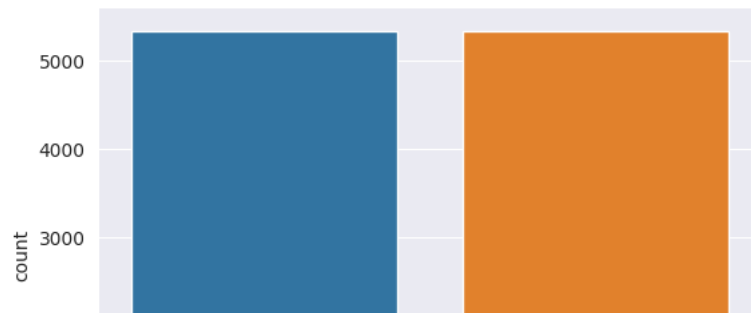
	reviews	labels	
0	simplistic , silly and tedious .	0	
1	it's so laddish and juvenile , only teenage bo...	0	
2	exploitative and largely devoid of the depth o...	0	
3	[garbus] discards the potential for pathologic...	0	
4	a visually flashy but narratively opaque and e...	0	
...	
10657	both exuberantly romantic and serenely melanch...	1	
10658	mazel tov to a film about a family's joyous li...	1	
10659	standing in the shadows of motown is the best ...	1	
10660	it's nice to see piscopo again after all these...	1	
10661	provides a porthole into that noble , tremblin...	1	

10662 rows x 2 columns

```
import seaborn as sns

sns.set_style('darkgrid')
sns.countplot(x='labels', data = df)
```

<Axes: xlabel='labels', ylabel='count'>



Naive Bayes

1000



```
from nltk.corpus.reader import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
True
```

```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
stopwords = set(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words = list(stopwords))
```

```
X = df.reviews
Y = df.labels
```

```
X.head()
```

```
0      simplistic , silly and tedious .
1  it's so laddish and juvenile , only teenage bo...
2  exploitative and largely devoid of the depth o...
3  [garbus] discards the potential for pathologic...
4  a visually flashy but narratively opaque and e...
Name: reviews, dtype: object
```

```
Y.head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: labels, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, train_size=0.75, random_state=1234, shuffle = 'True')
```

```
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
```

```
from sklearn.naive_bayes import MultinomialNB
```

```
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)
```

```
▼ MultinomialNB
MultinomialNB()
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
```

```
pred = naive_bayes.predict(X_test)
print(confusion_matrix(y_test, pred))
```

```

from sklearn.metrics import classification_report
print(classification_report(y_test, pred))

print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
[[1003  348]
 [ 274 1041]]
      precision    recall  f1-score   support

      0       0.79      0.74      0.76       1351
      1       0.75      0.79      0.77       1315

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

accuracy score:  0.7666916729182296
precision score:  0.7494600431965442
recall score:    0.7916349809885932
f1 score:        0.7699704142011835

vectorizer_b = TfidfVectorizer(stop_words=list(stopwords), binary=True)

X = vectorizer_b.fit_transform(df.reviews)
y = df.labels

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, train_size=0.75, random_state=1234, shuffle = 'True')

from sklearn.naive_bayes import BernoulliNB

naive_bayes2 = BernoulliNB()
naive_bayes2.fit(X_train, y_train)



▼ BernoulliNB
    BernoulliNB()



pred2 = naive_bayes2.predict(X_test)
print(confusion_matrix(y_test, pred2))

print(classification_report(y_test, pred2))

print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
[[1031  320]
 [ 302 1013]]
      precision    recall  f1-score   support

      0       0.77      0.76      0.77       1351
      1       0.76      0.77      0.77       1315

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

accuracy score:  0.7666916729182296
precision score:  0.7494600431965442
recall score:    0.7916349809885932
f1 score:        0.7699704142011835

```

Naive Bayes Analysis

When looking at the Naive Bayes models the two that were used were the Bernoulli Naive Bayes, which is a form of binomial naive bayes, and the multinomial naive bayes. When comparing the results of the two they are fairly similar with only a slight difference in the aspect that the Bernoulli Naive Bayes had a higher true positive rate, but a lower true negative rate in comparison to the multinomial naive bayes. Aside from the changes in the confusion matrix most of the scores stayed relatively similar including the accuracy with both achieving a score of 77%. It is interesting though to see how the two scores are fairly similar, my initial thoughts would be that the Bernoulli Naive Bayes would perform better

than the multinomial naive bayes seeing how the data is already in binary form. Regardless, both models did a fairly strong job in terms of classifying positive and negative reviews.

▼ Neural Networks

```
X = vectorizer.fit_transform(df.reviews)
y = df.labels

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, train_size=0.75, random_state=1234, shuffle = 'True')
```

```
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(activation = 'tanh', solver='adam', alpha=1e-5, hidden_layer_sizes=(15, 2), random_state=1)
# activation = tanh, solver = adam
# activation = tanh, solver = lbfgs
# activation = relu, solver = adam
# activatin = relu, solver = lbfgs
classifier.fit(X_train, y_train)
```

```
▼                                MLPClassifier
MLPClassifier(activation='tanh', alpha=1e-05, hidden_layer_sizes=(15, 2),
              random_state=1)
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score
```

```
pred = classifier.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
probs = classifier.predict_proba(X_test)
print('log loss: ', log_loss(y_test, probs))
```

```
[[ 963  388]
 [ 308 1007]]
      precision    recall  f1-score   support

      0       0.76      0.71      0.73       1351
      1       0.72      0.77      0.74       1315

 accuracy         0.74
  macro avg       0.74      0.74      0.74
 weighted avg     0.74      0.74      0.74
```

```
accuracy score:  0.7389347336834209
precision score: 0.7218637992831541
recall score:    0.7657794676806083
f1 score:        0.7431734317343174
log loss:        1.1538155852850784
```

Neural Network Analysis

When looking at the neural network model, it did fairly similar to that of the naive bayes which was surprising to me as I thought it may do better than that of the Naive Bayes. In fact, the accuracy for the neural networks model was lower than that of the Naive Bayes classifier in almost every iteration of different solvers and activation functions. The two activation functions that I utilized were the relu and tanh functions. The two solvers that I used were the adam and lbfgs solvers. The lowest accuracy came from when the activation function was relu and the solver was lbfgs where the accuracy was 49%. The highest accuracy came from when the activation function was tanh and the solver was lbfgs with an accuracy of 74.9%. There are probably many other parameters that could have been fine tuned to make it so that the accuracy was much higher such as using different solvers and different activation functions or even different layer sizes.

▼ Logistic Regression

```

from sklearn.linear_model import LogisticRegression

X = df.reviews
Y = df.labels

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, train_size=0.75, random_state=1234, shuffle = 'True')

vectorizer = TfidfVectorizer(binary=True)
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

classifier = LogisticRegression(solver='saga', class_weight='balanced')
classifier.fit(X_train, y_train)

```

```

▼
LogisticRegression
LogisticRegression(class_weight='balanced', solver='saga')

```

```

from sklearn.metrics import log_loss

pred = classifier.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))

[[ 963  388]
 [ 308 1007]]

```

	precision	recall	f1-score	support
0	0.76	0.71	0.73	1351
1	0.72	0.77	0.74	1315
accuracy			0.74	2666
macro avg	0.74	0.74	0.74	2666
weighted avg	0.74	0.74	0.74	2666

```

accuracy score: 0.7389347336834209
precision score: 0.7218637992831541
recall score: 0.7657794676806083
f1 score: 0.7431734317343174

```

▼ Logistic Regression using Pipes

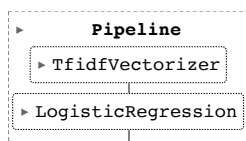
```

X = df.reviews
Y = df.labels

from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, train_size=0.75, random_state=1234, shuffle = 'True')

pipel = Pipeline([('tfidf', TfidfVectorizer(binary=True)), ('logreg', LogisticRegression(solver='sag', class_weight='balanced'))])
pipel.fit(X_train, y_train)

```



```

pred = pipel.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))

```

```

probs = model.predict_proba(X_test)
print('log loss: ', log_loss(y_test, probs))
[[1009  342]
 [ 270 1045]]

```

	precision	recall	f1-score	support
0	0.79	0.75	0.77	1351
1	0.75	0.79	0.77	1315
accuracy			0.77	2666
macro avg	0.77	0.77	0.77	2666
weighted avg	0.77	0.77	0.77	2666

```

accuracy score: 0.7704426106526632
precision score: 0.7534246575342466
recall score: 0.7946768060836502
f1 score: 0.7735011102886751
log loss: 0.5202432024937964

```

Logistic Regression Analysis

When doing the logistic regression, we used two different methods to create a model and classify the values. The first method that we used was utilizing the base logistic regression model and building a classifier off of that. The second model was built using pipelines and fitting the model to that. With the first base logistic regression model we utilized three different solvers of lbfgs, liblinear, and saga. The lbfgs solver got us an accuracy score of 77%, the liblinear solver got us an accuracy score of 77% as well, and the saga solver got us an accuracy of 77% but the confusion matrix of the true positive did one score better. The pipeline we used saga and sag solvers. Both the saga and sag solvers got a score of 77% with the sag score getting a slightly better true positive rate. Overall both the logistic regression with pipeline and basic classifier had a similar score regardless of what solver was being used.

Overall Analysis

Looking at the overall analysis of all the models, they all have fairly similar scores with the logistic regression analysis having the highest accuracy score, but the only issue is that the log loss value for both the pipeline and basic classifier are very high. However even the naive bayes and neural networks had a fairly high log loss score. In comparison the one that did better overall was the logistci regression classifier. One thing that I may do in the future is try to clean up the data to remove any outliers or data points that may cause for the log loss to be very high. This may even include trying to figure out what certain slang words are to figure out if they are good or bad in certain values. On top of this I could also hyper tune the parameters better to allow for certain changes in the log loss and accuracy.