Text Classification

```
import numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/rotten-tomatoes-reviews-dataset/rt-polarity-no-header.csv
/kaggle/input/rotten-tomatoes-reviews-dataset/data_rt.csv
```

```
df = pd.read_csv('../input/rotten-tomatoes-reviews-dataset/data_rt.csv', header=0, usecols=[0,1], encoding='latin-1')
df = df.sample(frac = 1)
print('rows and columns:', df.shape)
print(df.head())
```

```
rows and columns: (10662, 2)
                                                 reviews  labels
3608   while it may not add up to the sum of its part...       0
527    empire can't make up its mind whether it wants...       0
8264   smart , funny and just honest enough to provid...       1
2892   you would be better off investing in the worth...       0
10006  that the e-graveyard holds as many good ideas ...       1
```

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

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

```
# Setting up X and y
X = df.reviews
y = df.labels
```

```
# Looking into X
X.head()
```

```
3608     while it may not add up to the sum of its part...
527      empire can't make up its mind whether it wants...
8264     smart , funny and just honest enough to provid...
2892     you would be better off investing in the worth...
10006    that the e-graveyard holds as many good ideas ...
Name: reviews, dtype: object
```

```
# Looking into y
y[:10]
```

```
3608     0
527      0
8264     1
2892     0
10006    1
10184    1
2148     0
2967     0
2767     0
2812     0
Name: labels, dtype: int64
```

```
# Dividing into test/train
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

X_train.shape
```

```
(8529,)
```

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

```
# Looking into the data
print('train size:', X_train.shape)
print(X_train.toarray()[:5])

print('\ntest size:', X_test.shape)
print(X_test.toarray()[:5])
```

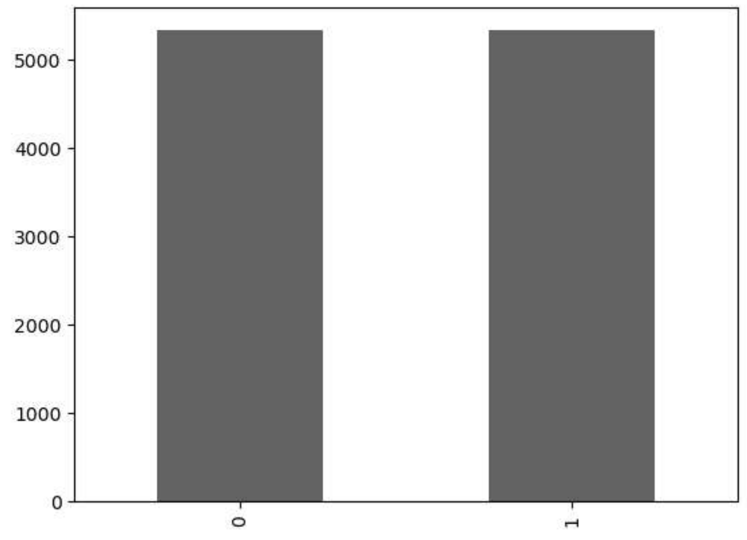
```
train size: (8529, 16296)
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

test size: (2133, 16296)
[[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.]]
```

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Creating a graph of the distribution of the target class
fig, ax = plt.subplots()
df['labels'].value_counts().plot(ax=ax, kind='bar')
```

```
<AxesSubplot:>
```



This data set consists of movie reviews from Rotten Tomatoes and whether they are positive or negative. There is an equal number of positive and negative reviews and they are organized by this attribute. This is why we shuffled the data set before using it. This model will be able to predict whether a movie review is positive or negative.

Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB

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

```
MultinomialNB()
```

```
# Priors
import math
```

```
prior_p = sum(y_train == 1)/len(y_train)
print('prior labels:', prior_p, 'log of prior:', math.log(prior_p))

naive_bayes.class_log_prior_[1]
```

```
prior labels: 0.5012310939148786 log of prior: -0.6906880189482489
-0.6906880189482489
```

```
# Log likelihood of words given the class
naive_bayes.feature_log_prob_
```

```
array([[ -9.83549646,  -9.78729538,  -8.91157172, ..., -10.28478609,
        -10.08507177, -10.07565809],
       [-10.29379823, -10.29379823,  -9.28296195, ..., -10.02512029,
        -10.29379823, -10.29379823]])
```

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

# Makes predictions on the test data
pred = naive_bayes.predict(X_test)

# Prints confusion matrix
print(confusion_matrix(y_test, pred))
```

```
[[799 278]
 [231 825]]
```

```
print('accuracy score: ', accuracy_score(y_test, pred))

print('\nprecision score (negative): ', precision_score(y_test, pred, pos_label=0))
print('precision score (positive): ', precision_score(y_test, pred))

print('\nrecall score: (negative)', recall_score(y_test, pred, pos_label=0))
print('recall score: (positive)', recall_score(y_test, pred))

print('\nf1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.7613689639006095

precision score (negative):  0.7757281553398059
precision score (positive):  0.7479601087941976

recall score: (negative) 0.7418755803156918
recall score: (positive) 0.78125

f1 score:  0.7642427049559981
```

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

```
              precision    recall  f1-score   support

           0       0.78      0.74      0.76      1077
           1       0.75      0.78      0.76      1056

    accuracy                           0.76      2133
   macro avg       0.76      0.76      0.76      2133
weighted avg       0.76      0.76      0.76      2133
```

```
print('spam size in test data:',y_test[y_test==0].shape[0])
print('test size: ', len(y_test))
baseline = y_test[y_test==0].shape[0] / y_test.shape[0]
print(baseline)
```

```
spam size in test data: 1077
test size:  2133
0.5049226441631505
```

We mentioned before that the dataset has an equal number of positive and negative reviews. Naive Bayes did perform better than if we had just guessed negative every time, which would have given us an accuracy of exactly 50%.

Logistic Regression

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


# Training
classifier = LogisticRegression(solver='lbfgs', class_weight='balanced')
classifier.fit(X_train, y_train)

# Evaluating
pred = classifier.predict(X_test)
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))
```

```
    accuracy score:  0.7566807313642757
    precision score:  0.7540208136234626
    recall score:  0.7547348484848485
    f1 score:  0.7543776620918127
    log loss:  0.5394111584905085
```

Logistic Regression was slightly less accurate than Naive Bayes but only by less than 1%. It has a better precision score and a worse recall and f1 score.

Neural Networks

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

```
    MLPClassifier(alpha=1e-05, hidden_layer_sizes=(15, 2), random_state=1,
                  solver='lbfgs')
```

```
pred = classifier.predict(X_test)
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))
```

```
    accuracy score:  0.49507735583684953
    precision score:  0.49507735583684953
    recall score:  1.0
    f1 score:  0.6622765757290687
```

The accuracy score is not great here. Let's increase the amount of layers to see if it will perform better.

```
classifier = MLPClassifier(solver='lbfgs', alpha=1e-5,
                   hidden_layer_sizes=(15, 3), random_state=1)
classifier.fit(X_train, y_train)
```

```
    MLPClassifier(alpha=1e-05, hidden_layer_sizes=(15, 3), random_state=1,
                  solver='lbfgs')
```

```
pred = classifier.predict(X_test)
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))
```

```
    accuracy score:  0.7313642756680732
    precision score:  0.723404255319149
    recall score:  0.740530303030303
    f1 score:  0.7318671034160037
```

That's better! It still isn't as accurate as the previous models but increasing the amount of layers again causes an error so that's as far as that trick will take us. The Neural Network's numbers are worse accross the board. Some data sets are just not suited for this model. In this case, our data set is smaller and relatively complex so a different model would perform better.