Assignment 7: Text Classification

Name: Aarya Patil

Date: 03/29/23

> Data set used: [SMS Spam Collection (Text Classification)

](https://www.kaggle.com/datasets/thedevastator/sms-spam-collection-a-more-diverse-dataset)

In [ ]:
```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sb
import math

from sklearn.model_selection import train_test_split
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
```

In [ ]:
```
# Save the data file name into a variable
input_file = "text_classification_data.csv"
data = pd.read_csv(input_file, header = 0) #load in data
print(data.head())
```

```
                                                 sms  label
0  Go until jurong point, crazy.. Available only ...      0
1                      Ok lar... Joking wif u oni...\n      0
2  Free entry in 2 a wkly comp to win FA Cup fina...      1
3  U dun say so early hor... U c already then say...      0
4  Nah I don't think he goes to usf, he lives aro...      0
```

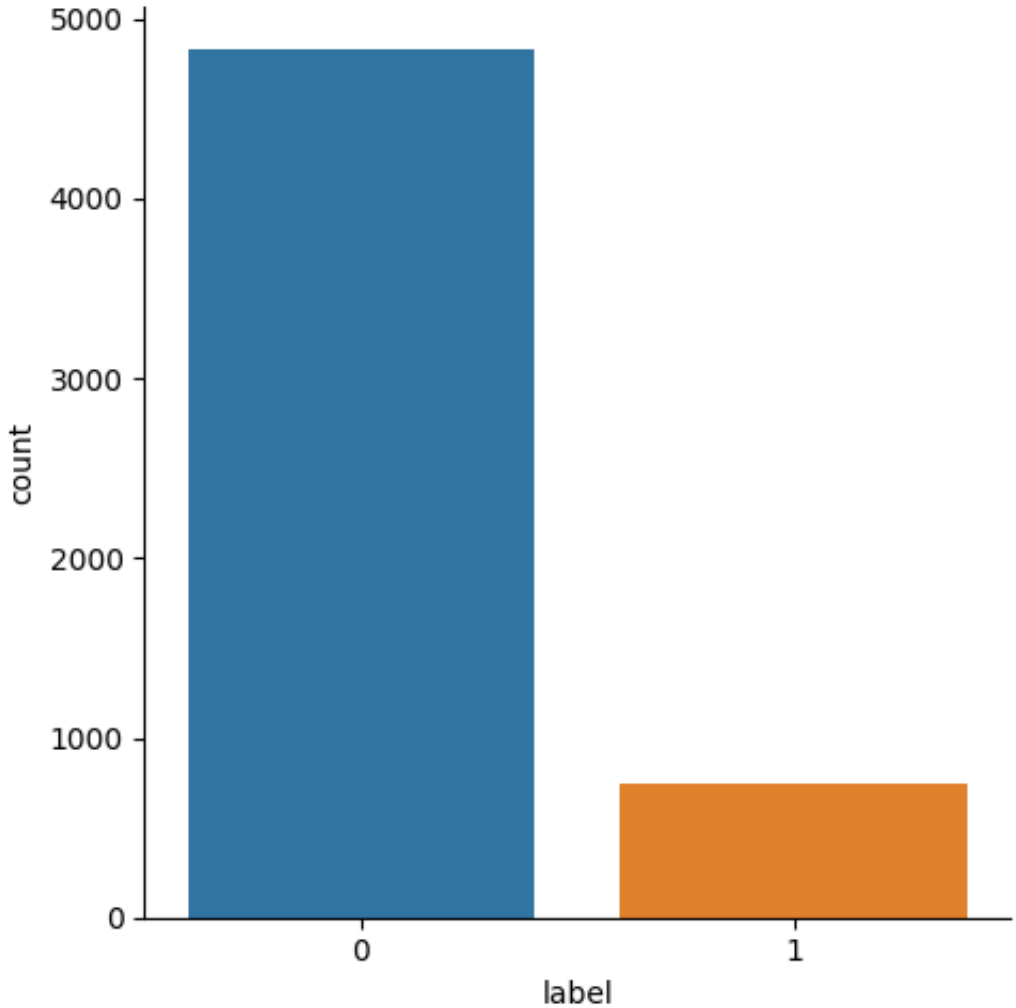
In [ ]:
```
# Split into train and test data
X = data.sms
y = data.label
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 350

# Print out training and testing data
print('X_train : ')
print(X_train.head(), "\n")

print('X_test : ')
print(X_test.head(), "\n")

print('y_train : ')
print(y_train.head(), "\n")

print('y_test : ')
print(y_test.head(), "\n")
```

```
X_train :
3969    Did u turn on the heater? The heater was on an...
1153                             Ok i go change also...\n
3563    Still chance there. If you search hard you wil...
5114    December only! Had your mobile 11mths+? You ar...
670                               Did u receive my msg?\n
Name: sms, dtype: object

X_test :
4077    87077: Kick off a new season with 2wks FREE go...
4682                           Are you staying in town ?\n
4355    important information 4 orange user 0789xxxxxx...
615     I called and said all to him:)then he have to ...
4393                      what are your new years plans?\n
Name: sms, dtype: object

y_train :
3969    0
1153    0
3563    0
5114    1
670     0
Name: label, dtype: int64

y_test :
4077    1
4682    0
4355    1
615     0
4393    0
Name: label, dtype: int64
```

Create a graph showing the distribution of the target classes

In [ ]:
```
# Convert to data frame so we can use seaborn
df_y = pd.DataFrame(y, columns = ['label'])

# Create a graph showing the distribution of the target classes
sb.catplot(x = "label", kind = 'count', data = df_y)
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x177e00a00>
```



Describe the data set and what the model should be able to predict

This data set holds SMS labeled messages that have been collected for mobile phone spam research. Messages that are not spam are labeled as '0' and messages that are spam are labeled as '1'.

The model should be able to predict whether a message is spam or not.

Naïve Bayes

In [ ]:

```
# Text preprocessing
stopwords = list(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words = stopwords)

# Use the tfidf vectorizer
X_train = vectorizer.fit_transform(X_train) #fit and transform the train dat
X_test = vectorizer.transform(X_test) #transform the test data

# Print out data after applying vectorizer
print('Train data size:', X_train.shape)
print(X_train.toarray()[:5])

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

```
Train data size: (4459, 7628)
[[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 data size: (1115, 7628)
[[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.]]
```

In [ ]:
```
# Run Multinomial Naïve Bayes
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)

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

# Check to see that the value from the code above and the code below are the
naive_bayes.class_log_prior_[1]
```

```
prior spam: 0.13321372505045975 log of prior: -2.0158004852648315
```

Out[ ]:
```
-2.0158004852648315
```

In [ ]:
```
# Plot out log values
naive_bayes.feature_log_prob_
```

Out[ ]:
```
array([[-9.75022321, -9.75022321, -9.5385121 , ..., -9.60125056,
        -9.75022321, -9.45630552],
       [-8.23799449, -7.32198145, -9.20087591, ..., -9.20087591,
        -9.00000199, -9.20087591]])
```

In [ ]:
```
# Evaluate data
pred = naive_bayes.predict(X_test) #make predictions on the

# Print results
print(confusion_matrix(y_test, pred))

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

print('\nprecision score (not spam): ', precision_score(y_test, pred, pos_la
print('precision score (spam): ', precision_score(y_test, pred))

print('\nrecall score: (not spam)', recall_score(y_test, pred, pos_label = 0
print('recall score: (spam)', recall_score(y_test, pred))

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

```
[[962   0]
 [ 30 123]]
accuracy score:  0.9730941704035875

precision score (not spam):  0.969758064516129
precision score (spam):  1.0

recall score: (not spam) 1.0
recall score: (spam) 0.803921568627451

f1 score:  0.891304347826087
```

In [ ]:
```
# Print out a classification report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       962
           1       1.00      0.80      0.89       153

    accuracy                           0.97      1115
   macro avg       0.98      0.90      0.94      1115
weighted avg       0.97      0.97      0.97      1115
```

Logistic Regression

In [ ]:
```
# Redefine values
vectorizer = TfidfVectorizer(ngram_range = (1, 2), max_features = 50000, mir

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 350
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
```

In [ ]:
```
# Run logistic regression
clf = LogisticRegression(C = 2.5, n_jobs = 4, solver = 'lbfgs', random_state
clf.fit(X_train, y_train)
```

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =        12011     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  3.09074D+03    |proj g|=  1.63550D+03

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
12011     42     49      1     0     0   1.168D-02   5.694D+02
  F =   569.35836465798229

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
 This problem is unconstrained.
[Parallel(n_jobs=4)]: Done   1 out of   1 | elapsed:    0.8s finished
```

Out[ ]:

▾ LogisticRegression

LogisticRegression(C=2.5, n_jobs=4, random_state=30, verbose=1)

In [ ]:

```python
# Evaluate data
pred = clf.predict(X_test)

# Print results
print(confusion_matrix(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))
```

```
[[962   0]
 [ 21 132]]
accuracy score: 0.9811659192825112
precision score: 1.0
recall score: 0.8627450980392157
f1 score: 0.9263157894736842
```

Neural Networks

In [ ]:

```
# Redefine values
vectorizer = TfidfVectorizer(stop_words = stopwords, binary = True)

X = vectorizer.fit_transform(data.sms)
y = data.label

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 350
```

In [ ]:

```
# Run Neural Networks
classifier = MLPClassifier(solver = 'lbfgs', alpha = 1e-5, hidden_layer_size
classifier.fit(X_train, y_train)
```

Out[ ]:

```
▾ MLPClassifier
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(16, 2), random_state
=1,
              solver='lbfgs')
```

In [ ]:

```
# Evaluate data
pred = classifier.predict(X_test)

# Print results
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: 0.9820627802690582
precision score: 0.9854014598540146
recall score: 0.8823529411764706
f1 score: 0.9310344827586207
```

Final Analysis

Out of all the approaches, Neural Networks returned the highest accuracy score, approximately 0.982, followed very closely by Logistic Regression, which was 0.981, and finally Naïve Bayes which had an accuracy score of 0.973. This order stays consistent when comparing recall and f1 scores. The precision score for Logistic Regression was a perfect 1.0 which was odd but other than this the order also stayed consistent for the precision scores.

I think that Neural Networks performed better than the other 2 algorithms because of its unique ability to automatically learn important features from raw data and also the fact that it can handle high-dimensional data. The other 2 algorithms struggle especially with the second part due to their issues with overfitting.