Text Classification SkLearn

First we will read in the dataset, and look at the first few rows. This is a data set I found on Kaggle about Stress.

```
import pandas as pd

df= pd.read_csv('Stress.csv', usecols=[3,4])
df.head()
```

	text	label
0	He said he had not felt that way before, sugge...	1
1	Hey there r/assistance, Not sure if this is th...	0
2	My mom then hit me with the newspaper and it s...	1
3	until i met my new boyfriend, he is amazing, h...	1
4	October is Domestic Violence Awareness Month a...	1

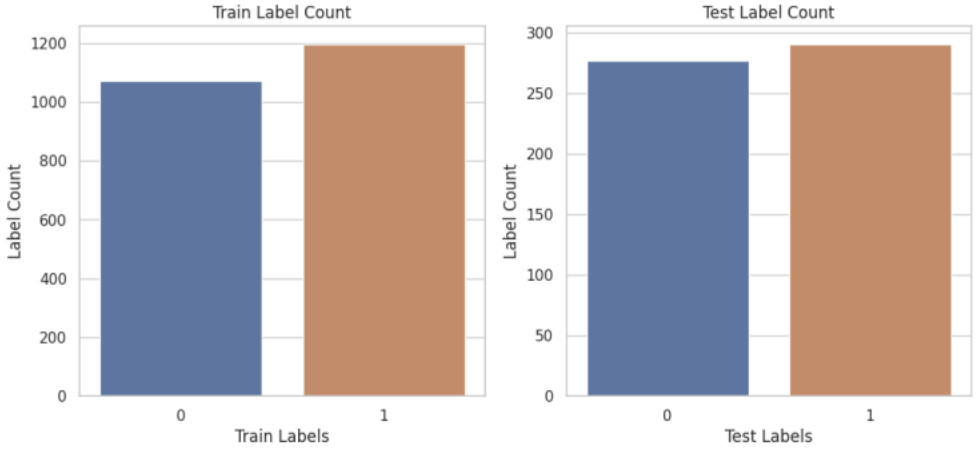
Dividing the Data 80/20

Here we divide the data into a 80/20 split.

```
from sklearn.model_selection import train_test_split
X= df.text
Y= df.label

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, train_size=0.8, test_size=0.2, random_state=1234)
```

After we split the data we can see the ratio if "stressed" to "not stressed" classifications. 0- not stressed. 1- stressed. We will plot the data using a bar graph.

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_theme(style="whitegrid")

fig, ax = plt.subplots(1,2, figsize=(12,5))
for index, group in enumerate([('Train',Y_train),('Test',Y_test)]):
  data = group[1].value_counts()
  sns.barplot(ax=ax[index], x=data.index, y= data.values)
  ax[index].set_title(f'{group[0]} Label Count')
  ax[index].set_xlabel(f'{group[0]} Labels')
  ax[index].set_ylabel(f'Label Count')
```



Naive Bayes

Here we will clean up the data before training. We will also remove the stopwords and make a tf-idf representation of the data.

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

```
from nltk import wordnet.lemmatizer
import re

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

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

print('train size:', X_train.shape)
print(X_train.toarray()[:5])

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

```
train size: (2270, 10212)
[[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: (568, 10212)
[[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.]]
```

Multinomial Naive Bayes

First we we will use the data to train a model that uses a Multinomial Naive Bayes classifier.

```
from sklearn.naive_bayes import MultinomialNB

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

```
▾ MultinomialNB
MultinomialNB()
```

Our model achieved a 69% accuracy.

```
from sklearn.metrics._plot.confusion_matrix import confusion_matrix
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))
print('\naccuracy score: ', accuracy_score(Y_test,pred))
print('\nprecision score(not stressed): ', precision_score(Y_test,pred,pos_label=0))
print('\nprecision score(stressed): ', precision_score(Y_test,pred))
print('\nrecall score(not stressed): ',recall_score(Y_test, pred, pos_label=0))
print('recall score(stressed): ', recall_score(Y_test,pred))

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

```
[[115 162]
 [ 12 279]]

accuracy score:  0.6936619718309859

precision score(not stressed):  0.905511811023622

precision score(stressed):  0.6326530612244898

recall score(not stressed):  0.4151624548736462
recall score(stressed):  0.9587628865979382

f1 score:  0.7622950819672131
```

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

```
              precision    recall  f1-score   support

           0       0.91      0.42      0.57       277
           1       0.63      0.96      0.76       291

    accuracy                           0.69       568
   macro avg       0.77      0.69      0.67       568
weighted avg       0.77      0.69      0.67       568
```

```
print(Y_test[Y_test != pred])
```

```
1744    0
251     0
2140    0
316     0
2789    0
       ..
483     0
431     0
130     0
281     0
1951    0
Name: label, Length: 174, dtype: int64
```

Binomial Naive Bayes

Now we train a model that uses binomial naive bayes and see if we get a different accuracy. From doing this we can see we achieved a accuracy of 75%.

```
vectorizer_b = TfidfVectorizer(stop_words=list(stops), binary=True)

X = vectorizer_b.fit_transform(df.text)
y = df.label

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,train_size=0.8, random_state=1234)
```

```
from sklearn.naive_bayes import BernoulliNB

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

```
▾ BernoulliNB
BernoulliNB()
```

```
pred = naive_bayes2.predict(X_test)

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))
```

```
[[172 105]
 [ 35 256]]
accuracy score:  0.7535211267605634
precision score:  0.7091412742382271
recall score:  0.8797250859106529
f1 score:  0.7852760736196319
```

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

```
              precision    recall  f1-score   support

           0       0.83      0.62      0.71       277
           1       0.71      0.88      0.79       291

    accuracy                           0.75       568
   macro avg       0.77      0.75      0.75       568
weighted avg       0.77      0.75      0.75       568
```

Logistic Regression

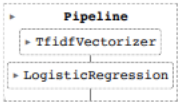
Next, let's use our data to train a model using Logistic Regression.

```
from sklearn.linear_model import LogisticRegression

X = df.text
y = df.label

X_train,X_test,y_train,y_test= train_test_split(df['text'], df['label'],test_size=0.2, train_size=0.8,random_state=1234)
```

Now... using a pipeline.

```
from sklearn.pipeline import Pipeline

pipe = Pipeline ([
    ('tfidf', TfidfVectorizer(binary=True)),
    ('logreg', LogisticRegression(solver='lbfgs',class_weight='balanced')),
])
```

```
pipe.fit(X_train, y_train)
```



We now see how the model performs on the test data and see that it had an accuracy score of 75%

```
from sklearn.metrics import log_loss

pipe.fit(X_train, y_train)

pred = pipe.predict(X_test)

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))

probs = pipe.predict_proba(X_test)
print('log loss: ', log_loss(y_test,probs))
```

```
[[202  75]
 [ 66 225]]
accuracy score:  0.7517605633802817
precision score:  0.75
recall score:  0.7731958762886598
f1 score:  0.7614213197969544
log loss:  0.5265801511065751
```

```
pipe.set_params(tfidf__min_df=3, logreg__C=2.0).fit(X_train, y_train)
pred = pipe.predict(X_test)
print("accuracy: ", accuracy_score(y_test,pred))
probs= pipe.predict_proba(X_test)
print("log loss: ", log_loss(y_test,probs))
```

```
accuracy:  0.7588028169014085
log loss:  0.5034540600918729
```

Neural Networks

Next, let's use our data to train a model using Neural Networks.

```
X = vectorizer.fit_transform(df.text)
y = df.label

vectorizer = TfidfVectorizer(stop_words=list(stops), binary=True)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,train_size=0.8,random_state=1234)


from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(32,2),random_state=1)

classifier.fit(X_train,y_train)
```

```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(32, 2), random_state=1,
              solver='lbfgs')
```

We again achieve an accuracy of about 75%

```
pred = classifier.predict(X_test)

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))
```

```
[[197  80]
 [ 67 224]]
accuracy score:  0.7411971830985915
precision score:  0.7368421052631579
```

```
recall score:  0.7697594501718213
f1 score:  0.7529411764705883
```