Text Classification with sklearn

April 2, 2023

First, let's read the data into a data frame and observe the first few rows of the data set. The data collected text from Reddit and identifies whether the user who made the post was stressed or not. We will use this data to build a machine learning model that predicts stress. The data is from https://www.kaggle.com/datasets/kreeshrajani/human-stress-prediction.

[1]:
```
import pandas as pd

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

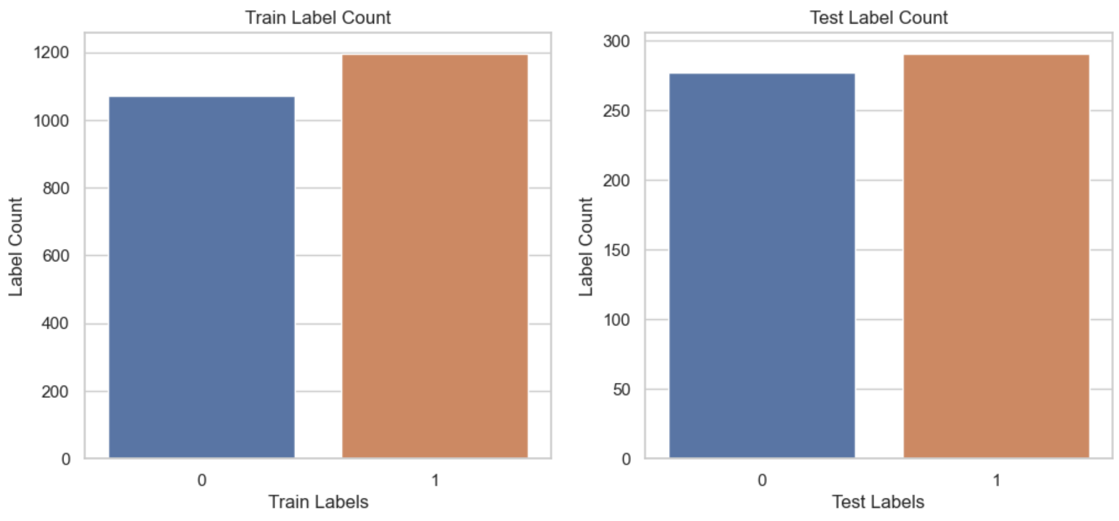
[1]:
```
                                                text  label
0  He said he had not felt that way before, sugge…      1
1  Hey there r/assistance, Not sure if this is th…      0
2  My mom then hit me with the newspaper and it s…      1
3  until i met my new boyfriend, he is amazing, h…      1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

Next, let's divide the data into an 80/20 train/test split.

[2]:
```
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 splitting our data, we can visualize the ratio of "not stressed" to "stressed" classifications in the training and testing data. A 0 corresponds to "not stressed," whereas a 1 corresponds to "stressed." This code was taken from https://stackoverflow.com/questions/70379709/visualization-data-train-and-data-test-from-train-test-split-with-seaborn.

[3]:
```
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('Label Count')
```



1 Naive Bayes

1.1 Text Pre-processing

Let's clean up the text before training our model. Firstly, we will remove stopwords and then create a tf-idf representation of the data. Then, we will take a peek at the tf-idf representation of the data. The matrices will be quite sparse due to the fact that each word does not occur in each post.

[4]:
```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk import word_tokenize
from nltk import WordNetLemmatizer
import re

# I tried doing additional text pre-processing (tokenizing, lowercasing,␣
 ↪lemmatizing, etc.) but it made my accuracy scores lower :(
# so, I commented it out
#
# for i in range(len(X_train)):
#     tokens = word_tokenize(X_train.iloc[i])
#     lemmatizer = WordNetLemmatizer()
#     processed = [word.lower() for word in tokens if word.isalnum()]
#     token_lemmas = [lemmatizer.lemmatize(token) for token in processed]
#     X_train.iloc[i] = ' '.join(token_lemmas)
#
# for i in range(len(X_test)):
#     tokens = word_tokenize(X_test.iloc[i])
#     lemmatizer = WordNetLemmatizer()
#     processed = [word.lower() for word in tokens if word.isalnum()]
#     token_lemmas = [lemmatizer.lemmatize(token) for token in processed]
#     X_test.iloc[i] = ' '.join(token_lemmas)

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.]]
```

1.2 Multinomial Naive Bayes

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

[5]:
```
from sklearn.naive_bayes import MultinomialNB

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

[5]:
```
MultinomialNB()
```

After training the model, we can evaluate how well it performs on the test data. Our model achieved an accuracy of approximately 69%, a F1 of 76%, and we can see that there is a high amount of

False Positives. In other words, our model falsely classified a 'not stressed' reddit post as 'stressed.'

[6]:
```
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('precision 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
```

[7]:
```
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
```

[8]:
```
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
```

1.3 Binomial Naive Bayes

Let's see if we can get a better performance by using our data to train a model that uses a Binomial Naive Bayes classifier instead. This may work better since it deals with the presence and absence or words, rather than word count. A 'stressed' post may have the presence of some words that hint that the user is stressed. As we can see below, we got a higher accuracy score of approximately 75% and a higher F1 score of 79%. This model had less False Positives, but also had a few more False Negatives.

[9]:
```python
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)
```

[10]:
```python
from sklearn.naive_bayes import BernoulliNB

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

[10]:
```
BernoulliNB()
```

[11]:
```python
# make predictions on the test data
pred = naive_bayes2.predict(X_test)

# print confusion matrix
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
```

[12]:
```
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
```

2 Logistic Regression

Next, let's use our data to train a model that using Logistic Regression. First, we need to divide the data into an 80/20 train/test split.

[13]:
```
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)
```

Next, we will train our data for Logistic Regression using a pipeline.

[14]:
```
from sklearn.pipeline import Pipeline

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

pipe.fit(X_train, y_train)
```

[14]:
```
Pipeline(steps=[('tfidf', TfidfVectorizer(binary=True)),
                ('logreg', LogisticRegression(class_weight='balanced'))])
```

Finally, we will evaluate how well our model performs on the test data. Our model had a higher accuracy score of 75%, a F1 score of 76%, and an okay log loss of 0.53. Admittedly, the log loss can be lower.

[15]:
```
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:  0.5265801511065752
```

By tweaking some of the parameters of the pipeline, we can get a slightly more accurate model with a lower log loss. Still, the log loss is not great.

[18]:
```
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:  0.5034540600918362
```

3 Neural Networks

Finally, we will use our data to train a neural network for classification. First, we need to divide the data into an 80/20 train/test split.

[19]:
```
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)
```

Next, we will build our neural network. I played around with the hidden layers and ultimately settled on (32,2). (16,2) was not as accurate as (32,2) and (64,2) did not converge.

[20]:
```
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)
```

[20]:
```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(32, 2), random_state=1,
              solver='lbfgs')
```

Lastly, we will evaluate how well our model performed on test data. As we can see, we got an accuracy of about 75% and a F1 score of 76%, which is comparable to the Logistic Regression model.

[21]:
```
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))
```

```
[[198  79]
 [ 65 226]]
accuracy score:  0.7464788732394366
precision score:  0.740983606557377
recall score:  0.7766323024054983
f1 score:  0.7583892617449665
```

4 Analysis

In this notebook, we performed classification with Naive Bayes, logistic regression, and neural networks to determine whether ths user of a reddit post was stressed or not based off of their post. For the Naive Bayes classifiers, we tried Multinomial Naive Bayes and Binomial Naive Bayes. The model that used Binomial Naive Bayes performed better with an accuracy of 75% and a f1 score of 79%. The Binomial Naive Bayes model performed very similarly to the model that used logistic regression. This model had an accuracy of 75% and a slightly lower f1 score of 76%. Our log loss for the logistic regression model should ideally be lower, though. The logistic regression model also performed similar to our neural network, which had an accuracy of 75% and a f1 score of 76%. Considering all three models performed similarly, I think better text pre-processing is the only thing that would increase the accuracy. While building these models, I tried thinking about how I could pre-process the text to distinguish stress from no stress. I experimented with capitalization thinking that stressed reddit posts may use more capital words. I also experimented with punctuation, reasoning that stressed reddit posts might use more punctuation. I tried lemmatizing each reddit post because I thought it may help with the word count. However, any additional steps I took decreased my accuracy. Those who work with this data set in the future will need to consider what additional steps they can take to pre-process the text to distinguish stressed reddit posts from those that aren't stressed.

```
[ ]:
```