Hamna B Mustafa

hbm170002

Author Attribution

Reading in 'federalist.csv' using numpy and converting the 'author' column to categorical data.

```
import pandas as pd
df = pd.read_csv('/content/federalist.csv')
df = df.astype({"author":'category'})
df
```

	author	text
0	HAMILTON	FEDERALIST. No. 1 General Introduction For the...
1	JAY	FEDERALIST No. 2 Concerning Dangers from Forei...
2	JAY	FEDERALIST No. 3 The Same Subject Continued (C...
3	JAY	FEDERALIST No. 4 The Same Subject Continued (C...
4	JAY	FEDERALIST No. 5 The Same Subject Continued (C...
...	...	...
78	HAMILTON	FEDERALIST No. 79 The Judiciary Continued From...
79	HAMILTON	FEDERALIST No. 80 The Powers of the Judiciary ...
80	HAMILTON	FEDERALIST. No. 81 The Judiciary Continued, an...
81	HAMILTON	FEDERALIST No. 82 The Judiciary Continued From...
82	HAMILTON	FEDERALIST No. 83 The Judiciary Continued in R...

83 rows × 2 columns

Displaying the counts by author

```
print("Total count:")
print(df.count())
print("\nHAMILTON count:")
print(df[df.author == 'HAMILTON'].count())
print("\nJAY count:")
print(df[df.author == 'JAY'].count())
print("\nMADISON count:")
print(df[df.author == 'MADISON'].count())
```

```
print("\nHAMILTON AND MADISON count:")
print(df[df.author == 'HAMILTON AND MADISON'].count())
print("\nHAMILTON OR MADISON count:")
print(df[df.author == 'HAMILTON OR MADISON'].count())
```

```
Total count:
author    83
text      83
dtype: int64

HAMILTON count:
author    49
text      49
dtype: int64

JAY count:
author    5
text      5
dtype: int64

MADISON count:
author    15
text      15
dtype: int64

HAMILTON AND MADISON count:
author    3
text      3
dtype: int64

HAMILTON OR MADISON count:
author    11
text      11
dtype: int64
```

Dividing the data into train and test

```
from sklearn.model_selection import train_test_split
x = df.text
y = df.author
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,train_size=0.8
```

```
print("Shape of train:")
print("X: " + str(X_train.shape))
print("Y: " + str(y_train.shape))

print("Shape of test:")
```

```
print("X: " + str(X_test.shape))
print("Y: " + str(y_test.shape))
```

```
Shape of train:
X: (66,)
Y: (66,)
Shape of test:
X: (17,)
Y: (17,)
```

Processing Text

```
# text preprocessing
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
import re
from sklearn.feature_extraction.text import TfidfVectorizer

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

X_train1 = vectorizer.fit_transform(X_train)
X_test1 = vectorizer.transform(X_test)

print("Shape of train:")
print("X: " + str(X_train1.shape))
print("Y: " + str(y_train.shape))

print("Shape of test:")
print("X: " + str(X_test1.shape))
print("Y: " + str(y_test.shape))
```

```
Shape of train:
X: (66, 7876)
Y: (66,)
Shape of test:
X: (17, 7876)
Y: (17,)
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Bernoulli Naive Bayes with only stopwords removed

```
from sklearn.naive_bayes import BernoulliNB

naive_bayes1 = BernoulliNB()
```

```
naive_bayes1.fit(X_train1, y_train)
```

```
BernoulliNB()
```

Printing the accuracy of bernoulli naive bayes with only stopwords removed

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

# make predictions on the test data
pred = naive_bayes1.predict(X_test1)

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

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

```
[[10  0  0  0]
 [ 3  0  0  0]
 [ 2  0  0  0]
 [ 2  0  0  0]]
accuracy score:  0.5882352941176471
```

Bernoulli Naive Bayes with stopwords removed, bigrams, and max_features

```
vectorizer_b = TfidfVectorizer(stop_words=stopwords, ngram_range=(1, 2),max_features=1

X_train2 = vectorizer_b.fit_transform(X_train)
X_test2 = vectorizer_b.transform(X_test)


print("Shape of train:")
print("X: " + str(X_train2.shape))
print("Y: " + str(y_train.shape))

print("Shape of test:")
print("X: " + str(X_test2.shape))
print("Y: " + str(y_test.shape))

naive_bayes2 = BernoulliNB()
naive_bayes2.fit(X_train2, y_train)
```

```
Shape of train:
X: (66, 1000)
```

```
Y: (66,)
Shape of test:
X: (17, 1000)
Y: (17,)
BernoulliNB()
```

Printing the accuracy of bernoulli naive bayes with stopwords removed, bigrams, and max_features set to 1000

```
# make predictions on the test data
pred2 = naive_bayes2.predict(X_test2)

# print confusion matrix
print(confusion_matrix(y_test, pred2))
print(X_test2.shape)

print('accuracy score: ', accuracy_score(y_test, pred2))
```

```
[[10  0  0  0]
 [ 0  3  0  0]
 [ 1  0  1  0]
 [ 0  0  0  2]]
(17, 1000)
accuracy score:  0.9411764705882353
```

As you can see, adding bigrams and max_features as parameters bumped the accuracy from 58% to 94%

Logistic Regression

Logistic regression without any parameters

```
from sklearn.linear_model import LogisticRegression

classifier1 = LogisticRegression()
classifier1.fit(X_train2, y_train)

# make predictions on the test data
pred3 = classifier1.predict(X_test2)

# print confusion matrix
print(confusion_matrix(y_test, pred3))
print(X_test2.shape)

print('accuracy score: ', accuracy_score(y_test, pred3))
```

```
[[10  0  0  0]
 [ 3  0  0  0]
 [ 2  0  0  0]
 [ 2  0  0  0]]
(17, 1000)
accuracy score:  0.5882352941176471
```

Logistic regression with parameters

```
classifier2 = LogisticRegression(multi_class='multinomial', solver='lbfgs', class_weig
classifier2.fit(X_train2, y_train)
print(X_train2.shape)

# make predictions on the test data
pred4 =classifier2.predict(X_test2)

# print confusion matrix
print(confusion_matrix(y_test, pred4))

print('accuracy score: ', accuracy_score(y_test, pred4))
```

```
(66, 1000)
[[10  0  0  0]
 [ 0  2  0  1]
 [ 1  0  1  0]
 [ 1  1  0  0]]
accuracy score:  0.7647058823529411
```

As you can see, adding the multi_class, solver, and class_weight parameters bumped the accuracy from 58% to 76%

Neural Networks

Neural Network model without any parameters

```
from sklearn.neural_network import MLPClassifier

NNclassifier = MLPClassifier()
NNclassifier.fit(X_train2, y_train)

predNN = NNclassifier.predict(X_test2)
print(confusion_matrix(y_test, predNN))

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

```
[[10  0  0  0]
```

```
 [ 0  2  0  1]
 [ 2  0  0  0]
 [ 1  0  0  1]]
accuracy score:  0.7647058823529411
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_percep
  ConvergenceWarning,
```

Neural network model with alpha, hidden_layer_sizes, and random_state set

```
NNclassifier = MLPClassifier( alpha=1e-5,
                   hidden_layer_sizes=(15, 7), random_state=1)
NNclassifier.fit(X_train2, y_train)

predNN = NNclassifier.predict(X_test2)
print(confusion_matrix(y_test, predNN))

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

```
[[10  0  0  0]
 [ 0  3  0  0]
 [ 2  0  0  0]
 [ 1  0  0  1]]
accuracy score:  0.8235294117647058
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_percep
  ConvergenceWarning,
```

As you can see, adding the alpha, hidden_layer_sizes, and random_state parameters bumped the accuracy from 76% to 82%

Trying different topoologies

```
NNclassifier = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter = 500)
NNclassifier.fit(X_train2, y_train)

predNN = NNclassifier.predict(X_test2)
print(confusion_matrix(y_test, predNN))

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

```
[[10  0  0  0]
 [ 0  3  0  0]
 [ 1  0  1  0]
 [ 1  0  0  1]]
accuracy score:  0.8823529411764706
```

```
NNclassifier = MLPClassifier(hidden_layer_sizes=(200, 150))
NNclassifier.fit(X_train2, y_train)
```

```
predNN = NNclassifier.predict(X_test2)
print(confusion_matrix(y_test, predNN))

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

```
[[10  0  0  0]
 [ 0  3  0  0]
 [ 1  0  1  0]
 [ 1  0  0  1]]
accuracy score:  0.8823529411764706
```

By trying different topologies, I was able to get an accuracy of 88% using Neural Networks. My final accuracy is 88%. However, Naive Bayes still performed the best and had an accuracy of 94%.