Author Attribution

Assignment 8
CS 4395.001: Human Language Technologies
Hannah Valena: HCV180000

1. Read in the csv file using pandas. Convert the author column to categorical data. Display the first few rows. Display the counts by author.

```
import pandas as pd

# read in csv to dataframe
url = 'https://raw.githubusercontent.com/hvalena/nlp-portfolio/main/Homework8-AuthorAt
df = pd.read_csv(url)

# convert author column to categorical data
df['author'] = df.author.astype('category')

# display first few rows
df.head()
```

	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...

```
# display counts by author
df.groupby(['author'])['author'].count()
```

```
author
HAMILTON                  49
HAMILTON AND MADISON       3
HAMILTON OR MADISON       11
JAY                        5
MADISON                   15
Name: author, dtype: int64
```

2. Divide into train and test, with 80% in train. Use random state 1234. Display the shape of train

```
from sklearn.model_selection import train_test_split

X = df.text # features
y = df.author # targets

# divide data frame into train (80%) and test (20%) with random state 1234
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    train_size=0.8,
                                                    random_state=1234)

# display shape of train and test
print('Train shape:')
print(X_train.shape)
print('\nTest shape:')
print(X_test.shape)
```

```
Train shape:
(66,)

Test shape:
(17,)
```

3. Process the text by removing stop words and performing tf-idf vectorization, fit to the training data only, and applied to train and test. Output the training set shape and the test set shape.

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

# tf-idf vectorization
vectorizer = TfidfVectorizer(stop_words=set(stopwords.words('english')))

# fit to the training data only
X_train1 = vectorizer.fit_transform(X_train)

# # apply to train and test
X_test1 = vectorizer.transform(X_test)

# output train and test set shape
print('Train shape:', X_train1.shape)
print(X_train1.toarray())
print('\nTest shape:', X_test1.shape)
print(X_test1.toarray())
```

```
Train shape: (66, 7876)
```

```
[[0.         0.         0.02956872 ... 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.         ... 0.02275824 0.         0.        ]]

Test shape: (17, 7876)
[[0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.02314673 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.        ]]
```

4. Try a Bernoulli Naïve Bayes model. The accuracy on the test set is 58%.

```
from sklearn.naive_bayes import BernoulliNB

# try a Bernoulli Naïve Bayes model
naive_bayes = BernoulliNB()
naive_bayes.fit(X_train1, y_train)
```

```
BernoulliNB()
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

# make predictions on test set
pred = naive_bayes.predict(X_test1)

# print confusion matrix and accuracy
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
```

5. The results from step 4 will be disappointing. The classifier just guessed the predominant class, Hamilton, every time. Looking at the train data shape above, there are 7876 unique words in the vocabulary. This may be too much, and many of those words may not be helpful. Redo the vectorization with max_features option set to use only the 1000 most frequent

words. In addition to the words, add bigrams as a feature. Try Naïve Bayes again on the new

```
# redo vectorization with max_features=1000, include unigrams and bigrams
vectorizer2 = TfidfVectorizer(stop_words=set(stopwords.words('english')),
                              max_features=1000, ngram_range=(1,2))

# fit to the training data only
X_train2 = vectorizer2.fit_transform(X_train)

# # apply to train and test
X_test2 = vectorizer2.transform(X_test)

# output train and test set shape
print('Train shape:', X_train2.shape)
print('Test shape:', X_test2.shape)

# redo bernoulli naive bayes model
naive_bayes2 = BernoulliNB()
naive_bayes2.fit(X_train2, y_train)
```

```
Train shape: (66, 1000)
Test shape: (17, 1000)
[[10  0  0  0]
 [ 0  3  0  0]
 [ 1  0  1  0]
 [ 0  0  0  2]]
accuracy score:  0.9411764705882353
```

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

# print confusion matrix and accuracy
print(confusion_matrix(y_test, pred2))
print('accuracy score: ', accuracy_score(y_test, pred2))
```

```
[[10  0  0  0]
 [ 0  3  0  0]
 [ 1  0  1  0]
 [ 0  0  0  2]]
accuracy score:  0.9411764705882353
```

6. Try logistic regression. Adjust at least one parameter in the LogisticRegression() model to see if you can improve results over having no parameters. Results were improved by changing the class_weight to balanced.

```
from sklearn.linear_model import LogisticRegression

# no parameters
```

```
log_reg = LogisticRegression()
log_reg.fit(X_train2, y_train)

pred3 = log_reg.predict(X_test2)
print(confusion_matrix(y_test, pred3))
print('accuracy score: ', accuracy_score(y_test, pred3))
```

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

```
# adjust parameters
log_reg2 = LogisticRegression(class_weight='balanced')
log_reg2.fit(X_train2, y_train)

pred4 = log_reg2.predict(X_test2)
print(confusion_matrix(y_test, pred4))
print('accuracy score: ', accuracy_score(y_test, pred4))
```

```
[[10  0  0  0]
 [ 0  2  0  1]
 [ 1  0  1  0]
 [ 1  1  0  0]]
accuracy score:  0.7647058823529411
```

7. Try a neural network. Try different topologies until you get good results. The final accuracy is 88%.

```
from sklearn.neural_network import MLPClassifier

nn = MLPClassifier(hidden_layer_sizes=(9, 5), max_iter=500,
                   solver='lbfgs')
```

```
MLPClassifier(hidden_layer_sizes=(9, 5), max_iter=500, solver='lbfgs')
```

```
pred5 = nn.predict(X_test2)
print(confusion_matrix(y_test, pred5))
print('accuracy score: ', accuracy_score(y_test, pred5))
```

```
[[10  0  0  0  0]
 [ 0  0  0  0  0]
 [ 0  0  3  0  0]
 [ 0  1  0  0  1]
 [ 0  0  0  0  2]]
accuracy score:  0.8823529411764706
```

Colab paid products - Cancel contracts here

✓ 0s completed at 9:36 PM ● ×