```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import BernoulliNB
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler

#nltk.download("stopwords")
```

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

```
# .csv must be located in the same directory as .ipynb, or imported into colab
data = pd.read_csv('./federalist.csv')

print(data.head()[:5])
print()
print(data[['author']].value_counts())
```

```
     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...

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

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

```
train, test = train_test_split(data, test_size=0.2, random_state=1234)
print('Training data shape:', train.shape)
print('Testing data shape: ', test.shape)
```

```
Training data shape: (66, 2)
Testing data shape:  (17, 2)
```

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.

```
# Remove stop words
stop_words = stopwords.words('english')

# Transform with vectorizer
vectorizer = TfidfVectorizer(stop_words=stop_words)
X_train = vectorizer.fit_transform(train['text'])
y_train = train['author']
X_test = vectorizer.transform(test['text'])
y_test = test['author']

print('Training set shape:', X_train.shape)
print('Testing set shape: ', X_test.shape)
```

```
    Training set shape: (66, 7876)
    Testing set shape:  (17, 7876)
```

Try a Bernoulli Naïve Bayes model. What is your accuracy on the test set?

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

acc = naive.score(X_test, y_test)
print('Accuracy:', str(round(acc * 100, 2)) + '%')
```

```
    Accuracy: 58.82%
```

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 train/test vectors and compare your results.

```
vectorizer = TfidfVectorizer(stop_words=stop_words, max_features=1000, ngram_range=(1, 2))

X_train = vectorizer.fit_transform(train['text'])
y_train = train['author']
X_test = vectorizer.transform(test['text'])
y_test = test['author']

naive = BernoulliNB()
```

```
naive.fit(X_train, y_train)
acc = naive.score(X_test, y_test)
print('Accuracy:', str(round(acc * 100, 2)) + '%')
```

```
Accuracy: 94.12%
```

Try logistic regression. Adjust at least one parameter in the LogisticRegression() model to see if you can improve results over having no parameters. What are your results?

```
lr = LogisticRegression(class_weight='balanced', multi_class='multinomial', max_iter=1000)
lr.fit(X_train, y_train)

acc = lr.score(X_test, y_test)
print('Accuracy:', str(round(acc * 100, 2)) + '%')
```

```
Accuracy: 76.47%
```

Try a neural network. Try different topologies until you get good results. What is your final accuracy?

```
scaler = StandardScaler(with_mean=False)
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

regr = MLPClassifier(random_state=1234, max_iter=100, hidden_layer_sizes=(100))
regr.fit(X_train_scaled, y_train)

acc = regr.score(X_test_scaled, y_test)
print('Accuracy:', str(round(acc * 100, 2)) + '%')
```

```
Accuracy: 88.24%
```

✓ 0s completed at 7:25 PM