```
import pandas
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

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.

```
# Read in csv file as data frame (DF)
dataFrame = pandas.read_csv('federalist.csv')

# Convert author column to categorical data
dataFrame['author'] = dataFrame.author.astype('category')

# Display first 10 rows of DF
count = 0
authorCounts = {}
print(dataFrame.head())
print("\n")

# Display counts by author
dataFrame['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...


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

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

```
# Divide data into training (80%) and testing (20%)
X = dataFrame.text
y = dataFrame.author
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, ra

# Display shape of train and test
print("Train Rows, Columns: ", X_train.shape)
print("Test Rows, Columns: ", X_test.shape)
```

```
Train Rows, Columns:  (66,)
Test Rows, Columns:  (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.

```
# Get stopwords and create corpus from training data
stopWords = stopwords.words('english')

# Perform tf-idf vectorization; fit to training data only
vectorizer = TfidfVectorizer(stop_words=stopWords)
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

# Display shape of train and test
print("train size: ", X_train.shape)
print("test size: ", X_test.shape)
```

```
train size:  (66, 7876)
test size:  (17, 7876)
```

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

```
# Perform NB algorithm
```

```
naive_bayes = BernoulliNB()
naive_bayes.fit(X_train, y_train)
pred = naive_bayes.predict(X_test)

# Display accuracy of test set
pred = naive_bayes.predict(X_test)
accuracy = accuracy_score(y_test, pred)
print("NB Accuracy: " + str(accuracy))
print("train size: ", X_train.shape)
print("test size: ", X_test.shape)
```

```
NB Accuracy: 0.5882352941176471
train size:  (66, 7876)
test size:  (17, 7876)
```

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 train/test vectors and compare your results.

```
# Redo vectorization: max_features=1000, add bigrams
vectorizer_2 = TfidfVectorizer(max_features=1000, ngram_range=(1,2), stop_words=stopWords)

# Re-Divide and Re-fit data
X = dataFrame.text
y = dataFrame.author
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, ra
X_train = vectorizer_2.fit_transform(X_train)
X_test = vectorizer_2.transform(X_test)

# Perform NB algorithm
naive_bayes = BernoulliNB()
naive_bayes.fit(X_train, y_train)

# Display accuracy of test set
pred_2 = naive_bayes.predict(X_test)
accuracy_2 = accuracy_score(y_test, pred_2)
print("NB Accuracy: " + str(accuracy_2))
```

```
print("train size: ", X_train.shape)
print("test size: ", X_test.shape)

# It looks like the score has increased significantly, going from ~60% accuracy to ~94%.
```

```
NB Accuracy: 0.9411764705882353
train size:  (66, 1000)
test size:  (17, 1000)
```

6. 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?

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score, log_loss
from sklearn import metrics

# Re-Divide and Re-fit data
X = dataFrame.text
y = dataFrame.author
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, ra
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

# Perform Logistic Regression
logReg = LogisticRegression(class_weight='balanced')
logReg.fit(X_train, y_train)

# Display results
pred = logReg.predict(X_test)
print('LR Accuracy: ', accuracy_score(y_test, pred))

# The accuracy by performing Logistic Regression when modifying the class_weight parameter
# when using bigrams with a maximum_feature of 1000.
```

```
LR Accuracy:  0.9411764705882353
```

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

```
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Re-Divide and Re-fit data
X = dataFrame.text
y = dataFrame.author
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, ra
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

# Create Neural Network(s)
labelEnc = LabelEncoder()
classifier = MLPClassifier(hidden_layer_sizes=(3,3), max_iter=500, random_state=1234)
regressor = MLPRegressor(hidden_layer_sizes=(3,3,2), max_iter=500, random_state=1234)

# Display results using MLPClassifier
print("__Using MLP Classifier:__")
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print('Classifier Accuracy: ', accuracy_score(y_test, y_pred))

# Display results of MLPRegressor
print("\n__Using MLP Regressor:__")
regressor.fit(X_train, labelEnc.fit_transform(y_train))
y_pred = regressor.predict(X_test)
print("Mean Squared Error: ", mean_squared_error(labelEnc.fit_transform(y_test), y_pred))
print("Correlation: ", r2_score(labelEnc.fit_transform(y_test), y_pred))

# Having 2 layers of 3 nodes each seems to get the highest accuracy when using MLPClassifie
# the first 2 with 3 nodes and the 3rd with 2.
```

```
__Using MLP Classifier:__
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.
  ConvergenceWarning,
Classifier Accuracy:  0.7058823529411765

__Using MLP Regressor:__
Mean Squared Error:  0.8878118833907458
Correlation:  0.20809372129652604
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.
  ConvergenceWarning,
```

Colab paid products - Cancel contracts here