Homework 9

Author: Anthony Martinez | amm180005
Date: 11/10/21

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.

In [38]:
```
import pandas as pd

# Read in the csv file using pandas.
df = pd.read_csv('federalist.csv')

# Convert the author column to categorical data.
df['author'] = df.author.astype('category')
print("\nChecking to make sure author column is now of category type\n")
print(df.dtypes)

# Display the first few rows.
print("\nDisplay first few rows using head()\n")
print(df.head())

# Display the counts by author
print("\nDisplay the counts by author using value_counts()")
df['author'].value_counts()
```

```
Checking to make sure author column is now of category type

author    category
text        object
dtype: object

Display first few rows using 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 the counts by author using value_counts()
```

Out[38]:
```
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.

In [4]:
```
# Set up x and y
X = df.text
y = df.author

# Divide into train and test, with 80% in train
from sklearn.model_selection import train_test_split
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 the shape of train and test
print("\nShape of the training set before pre-processing text", X_train.shape,
y_train.shape)
print("Shape of the test set before pre-processing text", X_test.shape, y_test
.shape)
```

```
Shape of the training set before pre-processing text (66,) (66,)
Shape of the test set before pre-processing text (17,) (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

In [5]:
```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

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

# apply tfidf vectorizer
X_train = vectorizer.fit_transform(X_train)  # fit and transform the train dat
a
X_test = vectorizer.transform(X_test)        # transform only the test data

# output training set shape and test set shape
print("\nShape of the training set after processing", X_train.shape)
print("Shape of the test set after processing", X_test.shape)
```

```
Shape of the training set after processing (66, 7876)
Shape of the test set after processing (17, 7876)
```

Naïve Bayes

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

Commentary: The accuracy for this Naïve Bayes model is only 58.8%. After modifying the vectorization Naïve Bayes scored an impressive 94.1% accuracy.

In [6]:
```
from sklearn.naive_bayes import BernoulliNB

# Building a Bernoulli Naive Bayes model
naive_bayes = BernoulliNB()
naive_bayes.fit(X_train, y_train)

# evaluate on test data
from sklearn.metrics import accuracy_score, confusion_matrix
pred = naive_bayes.predict(X_test)

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

# accuracy
print("Accuracy of Bernoulli Naïve Bayes model ", accuracy_score(y_test, pred
))
```

```
[[10  0  0  0]
 [ 3  0  0  0]
 [ 2  0  0  0]
 [ 2  0  0  0]]
Accuracy of Bernoulli Naïve Bayes model  0.5882352941176471
```

5. The results from step 4 is 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.

Naïve Bayes Model 2

In [13]:
```
# redo vectorization with max_features set to 1000 most frequent word
# also add bigrams as a feature
vectorizer_b = TfidfVectorizer(stop_words=stopwords,max_features=1000,min_df=2
,max_df=.5,ngram_range=(1,2))

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, random_state=1234)

# apply tfidf vectorizer
X_train = vectorizer_b.fit_transform(X_train)
X_test = vectorizer_b.transform(X_test)

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

pred2 = naive_bayes2.predict(X_test)
print(confusion_matrix(y_test,pred2))
print("Accuracy after modifying vectorization: ", accuracy_score(y_test,pred2
))
```

```
[[10  0  0  0]
 [ 0  3  0  0]
 [ 1  0  1  0]
 [ 0  0  0  2]]
Accuracy after modifying vectorization:  0.9411764705882353
```

Logistic Regression

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?

Commentary: Running a logistic regression model with no parameters only had an accuracy of 58.8%. Similar to the accuracy score of the Naive Bayes model before modifying the vectorization. I decided to add three parameters on the 2nd logistic regression model, multi_class, solver, and class_weight. Multi_class specifies multiple classes. We have 4 different classes in the data so it makes sense to use this parameter. I also decided to specify the lbfgs solver due to the fact that it is known to be a good choice for multiclass problems. Lastly, I decided to set the weight class to 'balanced' since it is used to ensure that the data is evenly distributed by class.

Logistic Regression Model 1 (No Parameters)

In [18]:
```
X = df.text
y = df.author

# divide into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train
_size=0.8, random_state=1234)

X_train = vectorizer_b.fit_transform(X_train)  # fit and transform the train d
ata
X_test = vectorizer_b.transform(X_test)        # transform only the test data

from sklearn.linear_model import LogisticRegression
glm1 = LogisticRegression()
glm1.fit(X_train, y_train)

# Evaluate on test data
pred_log_reg_1 = glm1.predict(X_test)
print("Accuracy for logistic regression model with no parameters: ", accuracy_
score(y_test,pred_log_reg_1))
```

```
Accuracy for logistic regression model with no parameters:  0.588235294117647
1
```

Logistic Regression Model 2 W/ Adjusted Parameters

In [19]:
```
# Building 2nd logistic regression model adding additional parameters in order
to improve results
# multi class specifies multiple classes
# the ibfgs solver is a good choice for multiclass problems which is the case
 we have here.
glm2 = LogisticRegression(multi_class='multinomial', solver='lbfgs', class_wei
ght='balanced')
glm2.fit(X_train, y_train)
pred_log_reg_2 = glm2.predict(X_test)

print("Accuracy for Logistic Regression Model with Parameters ", accuracy_scor
e(y_test,pred_log_reg_2))
```

```
Accuracy for Logistic Regression Model with Parameters  0.7647058823529411
```

Neural Networks

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

Commentary: The highest accuracy for the Neural Network models is only about 71%. Much lower than the 94% accuracy from the 2nd Naive Bayes model.

In [39]:
```
# set x and y
X = vectorizer_b.fit_transform(df.text)
y = df.author

# test train split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train
_size=0.8, random_state=1234)
```

Neural Network 1: Topology = 100

In [43]:
```
from sklearn.neural_network import MLPClassifier
nn = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(100), max_i
ter=1000, random_state=1)

# Neural Network 1 Results
nn.fit(X_train, y_train)
pred_nn = nn.predict(X_test)
print("Neural Network 1 accuracy ", accuracy_score(y_test,pred_nn))
```

```
Neural Network 1 accuracy  0.7058823529411765
```

Neural Network 2: Topology = 15,2

In [30]:
```
nn2 = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(15,2), max
_iter=1000, random_state=1)
nn2.fit(X_train, y_train)

# Neural Network 2 Results
pred_nn2 = nn2.predict(X_test)
print("Neural Network 2 accuracy ", accuracy_score(y_test,pred_nn2))
```

```
Neural Network 2 accuracy  0.47058823529411764
```

Neural Network 3: Topology = 4,3

In [31]:
```
nn3 = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(4,3), max_
iter=1000, random_state=1)
nn3.fit(X_train, y_train)

# Neural Network 3 Results
pred_nn3 = nn3.predict(X_test)
print("Neural Network 3 accuracy ", accuracy_score(y_test,pred_nn3))
```

```
Neural Network 3 accuracy  0.6470588235294118
```

Neural Network 4: Topology = 100,75,50,25,5,2

In [32]:
```
nn4 = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(100,75,50,
25,5,2), max_iter=1000, random_state=1)
nn4.fit(X_train, y_train)

# Neural Network 4 Results
pred_nn4 = nn4.predict(X_test)
print("Neural Network 4 accuracy ", accuracy_score(y_test,pred_nn4))
```

```
Neural Network 4 accuracy  0.5882352941176471
```