Lets first import pandas, so that we can read in our federalist.csv file. Our column count will be dependent on the author target.

```
#Importate imports of the program
import pandas as PD
import nltk
import matplotlib.pyplot as plt
import seaborn as SB
import numpy as NP

#For NLTK
from nltk import word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
nltk.download('stopwords')
from sklearn.linear_model import LogisticRegression

#For Naive Bayas
from sklearn.naive_bayes import *

#Imports of neural networks
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
#Reading in fileName
fileName = 'federalist.csv'
df = PD.read_csv(fileName)

#making the author coln to categorical data
df['author'] = df.author.astype('category')
df.head()

#Showing the author counts
print("Here are the instances of Hamilton, Madison, and Jay:\n")
df['author'].value_counts()
```

```
Here are the instances of Hamilton, Madison, and Jay:

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

Now we will from our train and test variables of the readed document. A nice 80/20 split between the train and test shape.

```
from sklearn.model_selection import train_test_split

X = df.drop('author', axis=1)
Y = df['author']

#Forming the 80% train and 20% test set
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=0.8, random_state=123)

#Results of the shaped test and train
print("Here is the train shape:",  X_train['text'].shape,
      "\nHere is the test shapes:", X_test['text'].shape)
```

```
Here is the train shape: (66,)
Here is the test shapes: (17,)
```

It is necessary that we filter our text data, so that we can fit it into the training data only.

By applying the train and test sets, we bet the following shapes.

```
stopwords = stopwords.words('english')

X_train = X_train['text']
X_test = X_test['text']

#Removing the stopwords from the text. All to fit the training data
vector = TfidfVectorizer(stop_words= stopwords)
X_train_vectored = vector.fit_transform(X_train)
X_test_vectored = vector.transform(X_test)

#making to array
X_train_vectored = X_train_vectored.toarray()
X_test_vectored = X_test_vectored.toarray()

#Displaying the results
print("The train shape after vectorization is:",X_train_vectored.shape)
print("The test shape after vectorization is:",X_test_vectored.shape)
```

```
The train shape after vectorization is: (66, 7858)
The test shape after vectorization is: (17, 7858)
```

Now we will see what the results are in Naive Bayas model.

```
#Getting MultinomialNB started
MNB = MultinomialNB()
MNB.fit(X_train_vectored, y_train)

#the MNB score
MNBscore = MNB.score(X_test_vectored, y_test)
print('Naive Bayes accuracy is:', MNBscore)
```

```
Naive Bayes accuracy is: 0.7058823529411765
```

Redoing the vectorization w/ max_fratures options.

```
vectored_limit = TfidfVectorizer(max_features= 1000, stop_words = stopwords,
                                 ngram_range= (1,1))
#forming the variables for vectored limited
X_train_vectoredLimit = vectored_limit.fit_transform(X_train)
X_test_vectoredLimit = vectored_limit.transform(X_test)

#making to array
X_train_vectoredLimit = X_train_vectoredLimit.toarray()
X_test_vectoredLimit = X_test_vectoredLimit.toarray()

#Attempt number 2 of NB w/ new train/test vectors
MNB_limited = MultinomialNB()
MNB_limited.fit(X_train_vectoredLimit, y_train)
MNB_limited_score = MNB_limited.score(X_test_vectoredLimit, y_test)

print("Limited Naive Bayes accuracy is =", MNB_limited_score)
print("Naive Bayes accuracy w/ limited vectorizer is =", MNB_limited_score / MNBscore)
```

```
Limited Naive Bayes accuracy is = 0.7058823529411765
Naive Bayes accuracy w/ limited vectorizer is = 1.0
```

Now lets use the logistical regression w/ one parmeter difference

```
LOGmodel = LogisticRegression(random_state = 101)
LOGmodel.fit(X_train_vectoredLimit, y_train)

LR_score = LOGmodel.score(X_test_vectoredLimit, y_test)
print("The logistical regression accuracy is =", LR_score)
```

```
The logistical regression accuracy is = 0.7058823529411765
```

Finally, lets try a neural network approach. We will use different topologies until there is a good result.

```
#This is for neural networks
Scaler = StandardScaler().fit(X_train_vectoredLimit)
X_train_scaled = Scaler.transform(X_train_vectoredLimit)
X_test_scaled = Scaler.transform(X_test_vectoredLimit)

#Topology attempt 1
regressor = MLPClassifier(hidden_layer_sizes= (15,3), max_iter = 6000)
regressor.fit(X_train_scaled, y_train)
regressor_score = regressor.score(X_test_scaled, y_test)
print("Here is the regressor accuracy:", regressor_score)

#Topology attempt 2
regressor = MLPClassifier(hidden_layer_sizes= (10,13), max_iter = 6000)
regressor.fit(X_train_scaled, y_train)
regressor_score = regressor.score(X_test_scaled, y_test)
print("Here is the regressor accuracy:", regressor_score)

#Topology attempt 3
regressor = MLPClassifier(hidden_layer_sizes= (11,3), max_iter = 6000)
regressor.fit(X_train_scaled, y_train)
regressor_score = regressor.score(X_test_scaled, y_test)
print("Here is the regressor accuracy:", regressor_score)

#Topology attempt 4
regressor = MLPClassifier(hidden_layer_sizes= (5,4), max_iter = 6000)
regressor.fit(X_train_scaled, y_train)
regressor_score = regressor.score(X_test_scaled, y_test)
print("Here is the regressor accuracy:", regressor_score)
```

```
Here is the regressor accuracy: 0.8235294117647058
Here is the regressor accuracy: 0.7647058823529411
Here is the regressor accuracy: 0.9411764705882353
Here is the regressor accuracy: 0.7647058823529411
```

✓ 4s completed at 9:54 PM