Author Attribution

CS 4395 - Intro to NLP

Dr. Karen Mazidi

Prepare by Leo Nguyen - ldn190002

Instruction 1

Read in cvs file using panda.

In [1]:
```
import pandas as pd
df = pd.read_csv('federalist.csv', header=0, encoding='latin-1')
```

Convert the author column to categorical data

In [2]:
```
df = df.astype({"author":'category'})
df.dtypes
```

Out[2]:
```
author    category
text        object
dtype: object
```

In [3]:
```
df.shape
```

Out[3]:
```
(83, 2)
```

Display first few fow

In [4]:
```
df.head()
```

Out[4]:

	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

In [5]:
```
df['author'].value_counts()
```

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

Instruction 2

Divide into train and test (80/20)

In [6]:
```
from sklearn.model_selection import train_test_split

X = df.text # Predictor
y = df.author  # Target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

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

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

In [7]:
```
X_train.head()
```

Out[7]:
```
66    FEDERALIST No. 67 The Executive Department Fro...
54    FEDERALIST No. 55 The Total Number of the Hous...
68    FEDERALIST No. 69 The Real Character of the Ex...
18    FEDERALIST No. 19 The Same Subject Continued (...
45    FEDERALIST No. 46 The Influence of the State a...
Name: text, dtype: object
```

In [8]:
```
y_train.head()
```

Out[8]:
```
66               HAMILTON
54     HAMILTON OR MADISON
68               HAMILTON
18    HAMILTON AND MADISON
45                 MADISON
Name: author, dtype: category
Categories (5, object): ['HAMILTON', 'HAMILTON AND MADISON', 'HAMILTON OR MADISON',
'JAY', 'MADISON']
```

Instruction 3

Process the text by removing stop words and performing tf-idf vectorization

In [9]:
```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

stopwords = set(stopwords.words('english'))
vectorizer1 = TfidfVectorizer(stop_words=stopwords)
```

Fit to the training data only, and applied to train and test

In [10]:
```
# apply tfidf vectorizer
X_train1 = vectorizer1.fit_transform(X_train)
X_test1 = vectorizer1.transform(X_test)
```

Output the training set shape and the test set shape.

In [11]:
```
print('train size:', X_train1.shape)
print(X_train1.toarray()[:5]) # take a peek at the data

print('\ntest size:', X_test1.shape)
print(X_test1.toarray()[:5]) # take a peek at the data
```

```
train size: (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.         ... 0.03741484 0.         0.        ]]

test size: (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.        ]]
```

Instruction 4 - Bernoulli Naïve Bayes Model

Train the naive bayes classifier

In [12]:
```
from sklearn.naive_bayes import BernoulliNB

naive_bayes1 = BernoulliNB()
naive_bayes1.fit(X_train1, y_train)
```

Out[12]:
```
BernoulliNB()
```

Evaluate on the test data

In [13]:
```
# make predictions on the test data
pred = naive_bayes1.predict(X_test1)

# print confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred)
```

Out[13]:
```
array([[10,  0,  0,  0],
       [ 3,  0,  0,  0],
       [ 2,  0,  0,  0],
       [ 2,  0,  0,  0]], dtype=int64)
```

Output the result

In [14]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, l

print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred, average='micro'))
print('recall score: ', recall_score(y_test, pred, average='micro'))
print('f1 score: ', f1_score(y_test, pred, average='micro'))
```

```
accuracy score:  0.5882352941176471
precision score:  0.5882352941176471
recall score:  0.5882352941176471
f1 score:  0.5882352941176471
```

We have a very low accuracy = 0.588 as expected. It is because when the data coming into the Bernoulli classifier is not binary, the classifier will binarize it (Our data have 5 different levels). And the classifier just guessed the predominant class, Hamilton, every time.

Instruction 5 - Bernoulli Naïve Bayes Model (With modified train data)

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

In [15]:
```
vectorizer2 = TfidfVectorizer(stop_words=stopwords, max_features=1000, ngram_range=(1,
# apply tfidf vectorizer
X_train2 = vectorizer2.fit_transform(X_train)
X_test2 = vectorizer2.transform(X_test)
```

Re-train the naive bayes classifier

In [16]:
```
naive_bayes2 = BernoulliNB()
naive_bayes2.fit(X_train2, y_train)
```

Out[16]:
```
BernoulliNB()
```

Re-evaluate on the test data

In [17]:
```
# make predictions on the test data
pred = naive_bayes2.predict(X_test2)

# print confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred)
```

Out[17]:
```
array([[10,  0,  0,  0],
       [ 0,  3,  0,  0],
       [ 1,  0,  1,  0],
       [ 0,  0,  0,  2]], dtype=int64)
```

Re-Output the result

In [18]:
```
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred, average='micro'))
print('recall score: ', recall_score(y_test, pred, average='micro'))
print('f1 score: ', f1_score(y_test, pred, average='micro'))
```

```
accuracy score:  0.9411764705882353
precision score:  0.9411764705882353
recall score:  0.9411764705882353
f1 score:  0.9411764705882353
```

We now have a very good accuracy = 0.9411 which is much higher compare to 0.5882. This happen because, we do some addition step on precessing the text. We only feature 1000 most frequent words and extract both unigrams and bigrams on the text. Then use them to predict the author.

Instruction 6 - Logistic Regression

Logistic Regression (With no parameters)

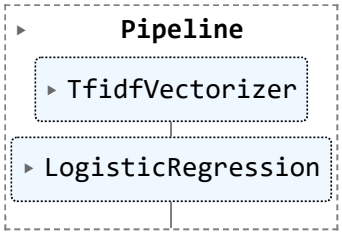
Vectorizer

In [19]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, l
from sklearn.pipeline import Pipeline

pipe1 = Pipeline([
        ('tfidf',  TfidfVectorizer(stop_words=stopwords, max_features=1000, ngram_rang
        ('logreg', LogisticRegression()),
])

pipe1.fit(X_train, y_train)
```

Out[19]:



Train the Logistic Regression classifier

Evaluate on test data

In [20]:
```
pred3 = pipe1.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred3))
```

```
print('precision score: ', precision_score(y_test, pred3, average='micro'))
print('recall score: ', recall_score(y_test, pred3, average='micro'))
print('f1 score: ', f1_score(y_test, pred3, average='micro'))
```

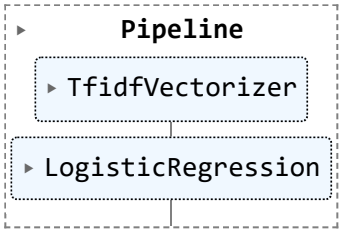
```
accuracy score:  0.5882352941176471
precision score:  0.5882352941176471
recall score:  0.5882352941176471
f1 score:  0.5882352941176471
```

Logistic Regression (With parameter adjusted)

In [21]:
```
pipe2 = Pipeline([
        ('tfidf',  TfidfVectorizer(stop_words=stopwords, max_features=1000, ngram_rang
        ('logreg', LogisticRegression(solver='lbfgs', class_weight='balanced')),
])

pipe2.fit(X_train, y_train)
```

Out[21]:



Re train and evaluate

In [22]:
```
pred4 = pipe2.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred4))
print('precision score: ', precision_score(y_test, pred4, average='micro'))
print('recall score: ', recall_score(y_test, pred4, average='micro'))
print('f1 score: ', f1_score(y_test, pred4, average='micro'))
```

```
accuracy score:  0.7647058823529411
precision score:  0.7647058823529411
recall score:  0.7647058823529411
f1 score:  0.7647058823529412
```

Compare result between 2 Logistic Regession Model

We have a big improvement on accuracy from 0.588 to 0.764. It is because we now specify the class_weight to balance which mean we will adjust the weight of each author prediction result based on the error that the model learned. If not all author will have the same weight which we know is not true as we know from previous result, the predominant class is Hamilton.

Instruction 7 - Neural Network

In [23]:
```
from sklearn.neural_network import MLPClassifier
```

A network of (15, 2)

In [24]:
```
pipe71 = Pipeline([
        ('tfidf', TfidfVectorizer(stop_words=stopwords, max_features=1000, ngram_range
```

```
        ('neuralnet', MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(15, 2), random_state=1)),
         ])

pipe71.fit(X_train, y_train)

pred = pipe71.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred, average='micro'))
print('recall score: ', recall_score(y_test, pred, average='micro'))
print('f1 score: ', f1_score(y_test, pred, average='micro'))
```

```
accuracy score:  0.6470588235294118
precision score:  0.6470588235294118
recall score:  0.6470588235294118
f1 score:  0.6470588235294118
```

A network of (100, 67, 15, 7)

In [25]:

```
pipe72 = Pipeline([
        ('tfidf', TfidfVectorizer(stop_words=stopwords, max_features=1000, ngram_range
        ('neuralnet', MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(100, 67, 15, 7), random_state=1)),
         ])

pipe72.fit(X_train, y_train)

pred = pipe72.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred, average='micro'))
print('recall score: ', recall_score(y_test, pred, average='micro'))
print('f1 score: ', f1_score(y_test, pred, average='micro'))
```

```
accuracy score:  0.7058823529411765
precision score:  0.7058823529411765
recall score:  0.7058823529411765
f1 score:  0.7058823529411765
```

A network of (21, 7)

In [26]:

```
pipe73 = Pipeline([
        ('tfidf', TfidfVectorizer(stop_words=stopwords, max_features=1000, ngram_range
        ('neuralnet', MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(21, 7), random_state=1)),
         ])

pipe73.fit(X_train, y_train)

pred = pipe73.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred, average='micro'))
print('recall score: ', recall_score(y_test, pred, average='micro'))
print('f1 score: ', f1_score(y_test, pred, average='micro'))
```

```
accuracy score:  0.8235294117647058
precision score:  0.8235294117647058
recall score:  0.8235294117647058
f1 score:  0.8235294117647058
```

After a several trial and error we can see that the network of (21, 7) got 82.3% accuracy, the network of (100, 67, 15, 7) got 70.5% accuracy and the network of (15, 2) got 64.7% accuracy. There is no suprise there, it is because even though the the network of (100, 67, 15, 7) have more nodes and hidden layers compare to other two, it is still do not have the highest accuracy among 3 topologies. One of reason is that, it failed the 3rd rules when choosen the number nodes, which is too much. It should be smaller than the 2x of input layer.