Reg Gonzalez | CS 4395.001

Assignment 10: Author Attribution

Step 1: Reading the csv file

```
import pandas as pd

df = pd.read_csv('/content/federalist.csv')
df.author = df.author.astype('category')  # Convert author column to categorical data
print(df.head())
print()

# Get the counts for each author and then display them
hamilton_count = 0
jay_count = 0
madison_count = 0
hamilton_or_madison_count = 0
hamilton_and_madison_count = 0

for i in range(83):
  if df.iloc[i, 0] == 'HAMILTON':
    hamilton_count += 1
  elif df.iloc[i, 0] == 'JAY':
    jay_count += 1
  elif df.iloc[i, 0] == 'MADISON':
    madison_count += 1
  elif df.iloc[i, 0] == 'HAMILTON OR MADISON':
    hamilton_or_madison_count += 1
  elif df.iloc[i, 0] == 'HAMILTON AND MADISON':
    hamilton_and_madison_count += 1

print("Hamilton count: ", hamilton_count)
print("Madison count: ", madison_count)
print("Jay count: ", jay_count)

print()
print("Hamilton or Madison count: ", hamilton_or_madison_count)
print("Hamilton and Madison count: ", hamilton_and_madison_count)
```

```
     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 count:  49
Madison count:  15
Jay count:  5

Hamilton or Madison count:  11
Hamilton and Madison count:  3
```

Step 2: Divide into train and test

```
from sklearn.model_selection import train_test_split

X = df.text     # feature
y = df.author   # target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, ranc

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

```
X_train shape:  (66,)
X_test shape:  (17,)
y_train shape:  (66,)
y_test shape:  (17,)
```

Step 3: Process the text

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

# Set up vectorizer
english_stopwords = set(stopwords.words('english'))
tfidf_vectorizer = TfidfVectorizer(stop_words=english_stopwords)

# Apply the vectorizer
# Fit/transform on training data and only transform test data
X_train = tfidf_vectorizer.fit_transform(X_train)
X_test = tfidf_vectorizer.transform(X_test)

# Display shape of train and test
```

```
print("X_train shape: ", X_train.shape)
print("X_test shape: ", X_test.shape)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
X_train shape:  (66, 7876)
X_test shape:  (17, 7876)
```

Step 4: Try a Bernoulli Naive Bayes Model

```
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score,

# Set up the Bernoulli Naive Bayes model
bernoulli_nb = BernoulliNB()
bernoulli_nb.fit(X_train, y_train)

# Evaluate on the test data
pred = bernoulli_nb.predict(X_test)
print("Confusion matrix: ")
print(confusion_matrix(y_test, pred))
print()

print('Accuracy score: ', accuracy_score(y_test, pred))
```

```
Confusion matrix:
[[10  0  0  0]
 [ 3  0  0  0]
 [ 2  0  0  0]
 [ 2  0  0  0]]

Accuracy score:  0.5882352941176471
```

Step 5: Redoing the vectorization

If we compare the results from our previous tfidf vectorization and our new vectorization, we can see that there is a much better improvement. The previous accuracy score was ~0.59, while the new accuracy is ~0.94.

```
# Redo the vectorization with max_features set to the 1000 most frequent words
# and add bigrams as a feature
english_stopwords = set(stopwords.words('english'))
tfidf_vectorizer = TfidfVectorizer(stop_words=english_stopwords, max_features=1000,
                                   min_df=2, max_df=0.5, ngram_range=(1, 2))
```

```
X = df.text     # feature
y = df.author   # target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, ranc

# Apply the vectorizer
# Fit/transform on training data and only transform test data
X_train = tfidf_vectorizer.fit_transform(X_train)
X_test = tfidf_vectorizer.transform(X_test)

# Set up the Bernoulli Naive Bayes model
bernoulli_nb = BernoulliNB()
bernoulli_nb.fit(X_train, y_train)

# Evaluate on the test data
pred = bernoulli_nb.predict(X_test)
print("New confusion matrix: ")
print(confusion_matrix(y_test, pred))
print()

print('New accuracy score: ', accuracy_score(y_test, pred))
```

```
New confusion matrix:
[[10  0  0  0]
 [ 0  3  0  0]
 [ 1  0  1  0]
 [ 0  0  0  2]]

New accuracy score:  0.9411764705882353
```

Step 6: Try logistic regression

Here, we try logistic regression to classify the different texts. The first model we made doesn't have any parameters in it, as sort of a baseline. The second model has 3 new parameters that we added for multi_class, solver, and class_weight.

The first model's accuracy was pretty bad, with only an accuracy score of ~0.59. The second model's accuracy, however, was much better. It had a score of ~0.99.

```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
import numpy as np
from sklearn import metrics

# Set up the logistic regression models.
# The first model will have no parameters, while the second model will
# have parameters to hopefully make our results better.
logreg_model1 = Pipeline([
```

```
        ('tfidf', TfidfVectorizer()),
        ('logreg', LogisticRegression()),
])

logreg_model2 = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('logreg', LogisticRegression(multi_class='multinomial', solver='lbfgs',class_weight=
])

# Fit the models
logreg_model1.fit(df.text, df.author)
logreg_model2.fit(df.text, df.author)

# Predict on the data
pred1 = logreg_model1.predict(df.text)
pred2 = logreg_model2.predict(df.text)

# Output the results
print("Confusion matrix for logreg_model1:\n", metrics.confusion_matrix(df.author, pred1))
print("\nAccuracy for logreg_model1: ", np.mean(pred1 == df.author))
print()
print("Confusion matrix for logreg_model2:\n", metrics.confusion_matrix(df.author, pred2))
print("\nAccuracy for logreg_model2: ", np.mean(pred2 == df.author))
```

```
Confusion matrix for logreg_model1:
 [[49  0  0  0  0]
 [ 3  0  0  0  0]
 [11  0  0  0  0]
 [ 5  0  0  0  0]
 [15  0  0  0  0]]

Accuracy for logreg_model1:  0.5903614457831325

Confusion matrix for logreg_model2:
 [[49  0  0  0  0]
 [ 0  3  0  0  0]
 [ 0  0 11  0  0]
 [ 0  0  0  5  0]
 [ 0  0  1  0 14]]

Accuracy for logreg_model2:  0.9879518072289156
```

Step 7: Try a neural network

Here, we try neural networks for classifying different texts. The first model has several parameters, which are the lbfgs solver, alpha = 1e-5, two hidden layers (one of size 15 and the other of size 2), a random state of 1234, etc. The second model has the parameters for the lbfgs solver, two hidden layers (one of size 36 and the other of size 15), max iterations = 1000, and the same random state of 1234.

The first model's accuracy wasn't good, with only a score of ~0.59. The second model's score was better, but it still could be improved; it's score was ~0.76.

```
from sklearn.neural_network import MLPClassifier

# Preprocess the text
english_stopwords = set(stopwords.words('english'))
tfidf_vectorizer = TfidfVectorizer(stop_words=english_stopwords, binary=True)

# Set up X and y & divide into train and test
X = tfidf_vectorizer.fit_transform(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, ranc

# Set up and train the NN models.
regr1 = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(15, 2), random_state=12
regr2 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(36, 15), max_iter=1000, random_stat
regr1.fit(X_train, y_train)
regr2.fit(X_train, y_train)

# Predict on the test data
pred1 = regr1.predict(X_test)
pred2 = regr2.predict(X_test)

# Output the results
print("Accuracy for regr1: ", accuracy_score(y_test, pred1))
print()
print("Accuracy for regr2: ", accuracy_score(y_test, pred2))
```

```
Accuracy for regr1:  0.5882352941176471

Accuracy for regr2:  0.7647058823529411
```

✓ 4s completed at 9:54 AM