notebook

November 7, 2022

1 Author Attribution

1.0.1 Objectives:

• Gain experience with machine learning using sklearn • Experiment with the NLP task author attribution

[1]:
```
# Library imports

import pathlib
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
```

2 Part 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.

[2]:
```
# read the csv file
df = pd.read_csv(pathlib.Path.cwd().joinpath("federalist.csv"))

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


# Display the first few rows
print(df.head())

# Display the counts by author
df.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…
```

[2]:
```
HAMILTON                49
MADISON                 15
HAMILTON OR MADISON     11
JAY                      5
HAMILTON AND MADISON     3
Name: author, dtype: int64
```

3 Part 2

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

[3]:
```
# feature
x = df.text
# target
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)

# Display the 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_train shape: ", y_train.shape)
```

```
x_train shape :  (66,)
x_test shape :  (17,)
y_train shape:  (66,)
y_train shape:  (66,)
```

4 Part 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.

[4]:
```
vector = TfidfVectorizer(stop_words= 'english')

# perform vectorizer
x_train = vector.fit_transform(x_train)
x_test = vector.transform(x_test)

# Display the shape of train and test
```

```
print("x_train shape: " , x_train.shape)
print("x_test shape: ", x_test.shape)
```

```
x_train shape:  (66, 7727)
x_test shape:  (17, 7727)
```

5 Part 4

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

[5]:
```
# Bernoulli Naive bayes model
naiveBayes = BernoulliNB()
naiveBayes.fit(x_train, y_train)

# evaluate the test data
guess = naiveBayes.predict(x_test)

# print acccuracy
print("The accuracy on the test set is: ", accuracy_score(y_test, guess))
```

```
The accuracy on the test set is:  0.5882352941176471
```

6 Part 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.

[6]:
```
vector = TfidfVectorizer(stop_words= 'english', max_features= 1000,␣
 ↪ngram_range= (1,2))

# feature
x = df.text
# target
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)

# perform vectorizer
x_train = vector.fit_transform(x_train)
x_test = vector.transform(x_test)

# Bernoulli Naive bayes model
naiveBayes = BernoulliNB()
```

```
naiveBayes.fit(x_train, y_train)

# evaluate the test data
guess = naiveBayes.predict(x_test)

# print acccuracy
print("The new accuracy on the test set is: ", accuracy_score(y_test, guess))
```

```
The new accuracy on the test set is:  0.9411764705882353
```

7 Part 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?

[7]:
```
# model with no parameters
classifer = LogisticRegression()
classifer.fit(x_train, y_train)

# estimate
guess = classifer.predict(x_test)

# print accuracy
print("The accuracy score(without parameters) is: ", accuracy_score(y_test,␣
 ↪guess))

# model with parameters
log = LogisticRegression(solver= 'lbfgs', class_weight= 'balanced')
log.fit(x_train, y_train)

# estimate
guess = log.predict(x_test)

# print accuracy
print("The accuracy score(with parameters) is: ", accuracy_score(y_test, guess))
```

```
The accuracy score(without parameters) is:  0.5882352941176471
The accuracy score(with parameters) is:  0.7058823529411765
```

8 Part 7

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

[8]:
```
# Neural Network training models
n_network1 = MLPClassifier(solver= 'lbfgs', alpha= 1e-5, hidden_layer_sizes=␣
 ↪(30,), random_state= 1234)
n_network2 = MLPClassifier(solver= 'lbfgs', alpha= 1e-5, hidden_layer_sizes=␣
 ↪(1000,), random_state= 1234)
n_network1.fit(x_train, y_train)
n_network2.fit(x_train, y_train)

# guess on test data
guess1 = n_network1.predict(x_test)
guess2 = n_network2.predict(x_test)

# print accuracy
print("Accuracy for n_network1: ", accuracy_score(y_test, guess1))
print("Accuracy for n_network2: ", accuracy_score(y_test, guess2))
```

```
Accuracy for n_network1:  0.6470588235294118
Accuracy for n_network2:  0.7647058823529411
```