authorattribution

November 9, 2022

1 1 Read in data

This cell reads in the data and displays the count of each author

[54]:
```
import pandas as pd
df = pd.read_csv("federalist.csv")
df['author'].value_counts()
```

[54]:
```
HAMILTON               49
MADISON                15
HAMILTON OR MADISON    11
JAY                     5
HAMILTON AND MADISON    3
Name: author, dtype: int64
```

2 2, 3 Preprocessing

This cell removes stopwords, splits the data into train and test splits, then performes tf-idf vectorization on the training data, and applied to train and test.

[55]:
```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
stopwords = set(stopwords.words("english"))
vectorizer = TfidfVectorizer(stop_words=stopwords)

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)

print(f"Train shape: X: {X_train.shape}, y: {y_train.shape}")
print(f"Test shape: X: {X_test.shape}, y: {y_test.shape}")

vectorizer = TfidfVectorizer(stop_words=stopwords)
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
print(f"Train shape: X: {X_train.shape}, y: {y_train.shape}")
```

```
print(f"Test shape: X: {X_test.shape}, y: {y_test.shape}")
```

```
Train shape: X: (66,), y: (66,)
Test shape: X: (17,), y: (17,)
Train shape: X: (66, 7876), y: (66,)
Test shape: X: (17, 7876), y: (17,)
```

[29]:
```
print(f"Train shape: {v_train.shape}")
print(f"Test shape: {v_test.shape}")
```

```
Train shape: (66, 7752)
Test shape: (2, 7752)
```

3 4 Naive Bayes Model

This configuration of Naive Bayes is clearly suboptimal, as there is not even 59% accuracy.

[56]:
```
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score

bernoulli_nb = BernoulliNB()
bernoulli_nb.fit(X_train, y_train)
prediction = bernoulli_nb.predict(X_test)

print(f"Accuracy on test set: {accuracy_score(y_test, prediction)}")
```

```
Accuracy on test set: 0.5882352941176471
```

4 5 Naive Bayes Take 2

This cell retries Naive Bayes with a CountVectorizer using bigrams as well as max features of 1000. This configuration is a great improvement on the previous one, as it has 94% accuracy.

[57]:
```
from sklearn.feature_extraction.text import CountVectorizer
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪train_size=0.8, random_state=1234)

vectorizer = CountVectorizer(stop_words=stopwords,
                             ngram_range=(1,2),
                             max_features=1000)
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
bernoulli_nb = BernoulliNB()
bernoulli_nb.fit(X_train, y_train)
prediction = bernoulli_nb.predict(X_test)

print(f"Accuracy on test set: {accuracy_score(y_test, prediction)}")
```

```
Accuracy on test set: 0.9411764705882353
```

5 6 Logistic Regression

In this cell, I try to predict authors using logistic regression. This leads to poor accuracy, which is curiously the exact same as the first NB accuracy. I was able to improve the accuracy to 88% by adding the parameter C=1000, which decreases regularization.

[85]:
```
from sklearn.linear_model import LogisticRegression
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪train_size=0.8, random_state=1234)

vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

classifier = LogisticRegression()
classifier.fit(X_train, y_train)
print(f"Try 1 accuracy score: {accuracy_score(y_test, classifier.
 ↪predict(X_test))}")

classifier = LogisticRegression(C=1000)
classifier.fit(X_train, y_train)
print(f"Try 2 accuracy score: {accuracy_score(y_test, classifier.
 ↪predict(X_test))}")
```

```
Try 1 accuracy score: 0.5882352941176471
Try 2 accuracy score: 0.8823529411764706
```

6 7 Neural Network

I tried 50*50 different combinations, as well as different activation functions, and found that 23 hidden layers with 5 nodes each, all using sigmoid as the activation function, gave the highest accuracy of 0.9411764705882353.

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪train_size=0.8, random_state=1234)

maxconfig = (0,0)
bestaccuracy = 0
for i in range(1,50):
    for j in range(1,50):
        config = (i,j)
        vectorizer = TfidfVectorizer(stop_words=stopwords, binary=True)
        NN = MLPClassifier(solver='lbfgs', alpha=1e-5,
                           hidden_layer_sizes=config, random_state=1,
                           max_iter=10000, activation='relu'
                          )
        pipe = Pipeline([
            ('tfidf', vectorizer),
```

```
            ('neuralnet', NN),
        ])

        pipe.fit(X_train, y_train)
        acc = np.mean(pipe.predict(X_test)==y_test)
        #print(f"{config}: acc={acc}")
        if acc > bestaccuracy:
            maxconfig = (i,j)
            bestaccuracy=acc

            print(f"New best: {maxconfig}, acc={acc}")
```

```
New best: (1, 1), acc=0.5882352941176471
New best: (1, 2), acc=0.7058823529411765
New best: (1, 13), acc=0.7647058823529411
New best: (6, 43), acc=0.8235294117647058
New best: (24, 20), acc=0.8823529411764706
```

```
c:\python39\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:559:
ConvergenceWarning: lbfgs failed to converge (status=2):
ABNORMAL_TERMINATION_IN_LNSRCH.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
c:\python39\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:559:
ConvergenceWarning: lbfgs failed to converge (status=2):
ABNORMAL_TERMINATION_IN_LNSRCH.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

[133]:
```
from sklearn.pipeline import Pipeline
from sklearn.neural_network import MLPClassifier
import numpy as np

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪train_size=0.8, random_state=1234)
vectorizer = TfidfVectorizer(stop_words=stopwords, binary=True)
NN = MLPClassifier(solver='lbfgs', alpha=1e-5,
                   hidden_layer_sizes=(5, 23), random_state=1,
                   max_iter=10000, activation='logistic'
                  )
pipe = Pipeline([
```

```
    ('tfidf', vectorizer),
    ('neuralnet', NN),
])

pipe.fit(X_train, y_train)
print(f"Accuracy: {np.mean(pipe.predict(X_test)==y_test)}")
```

```
Accuracy: 0.9411764705882353
```