author_attribution

November 6, 2022

1 Author Attribution

In this notebook I will predict the author of a given text from federalist.csv. It is a csv file of text and it's given author whom can be Hamilton, Jay, or Madison. Sometimes a mixture of the authors as well. I will utlize pandas for the data processing, NLP for the word processing, and then sklearn to perform Bernoulli Naive Bayes, Logistic Regression, and Neural Network.

1.1 Reading and Processing the Data

1.1.1 With Pandas

[1]:
```
# Importing Pandas
import pandas as pd
# Reading in the csv file with pandas
df = pd.read_csv('federalist.csv')
# Converting the author column to categorical data
df.author = df.author.astype('category')
# Displaying the counts of each author
print("Counts: ")
print(df.author.value_counts())
# Displaying the first few rows of the data frame
print("\nFirst few rows of data frame:")
df.head()
```

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

First few rows of data frame:
```

[1]:
```
     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…
```

1.1.2 Utilizing sklearn to create train/test data frames

[2]:
```
# Import sklearn's train_test_split
from sklearn.model_selection import train_test_split
# Divide into train and test (80/20 with seed 1234 for replicable results)
# X contains the predictor columns and y contains the target column
X_train, X_test, y_train, y_test = train_test_split(
    df.text, df.author, test_size=0.2, random_state=1234,
    stratify=df.author)

# Outputting the dimensions of train and test
print("Dimensions of train data: ", X_train.shape)
print("Dimensions of test data: ", X_test.shape)
```

```
Dimensions of train data:  (66,)
Dimensions of test data:  (17,)
```

1.1.3 Removing stop words

[3]:
```
# Importing the nltk stopwords
from nltk.corpus import stopwords
# This is our set of stopwords, it will be used during vectorization
stopwords = set(stopwords.words('English'))
```

1.1.4 Performing tf-idf Vectorization

[4]:
```
# Importing our tf-idf vectorizer from sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
# Setting up our stopwords for our vectorizer
vectorizer = TfidfVectorizer(stop_words=stopwords)
# Perform tf-idf vectorization and fit to training data
X_train_vect = vectorizer.fit_transform(X_train)
# Transforming the test data with the fitted tf-idf vectorization
X_test_vect = vectorizer.transform(X_test)
# Outputting the dimensions of train and test
print("Dimensions of the vectorized train data: ", X_train_vect.shape)
print("Dimensions of the vectorized test data: ", X_test_vect.shape)
```

```
Dimensions of the vectorized train data:  (66, 7678)
Dimensions of the vectorized test data:  (17, 7678)
```

1.2 Benoulli Naive Bayes

1.2.1 The First Model

[5]:
```
# Importing the Benoulli Naive Bayes model from sklearn
from sklearn.naive_bayes import BernoulliNB
# Creating Benoulli Naive Bayes model
b_nb_1 = BernoulliNB()
b_nb_1.fit(X_train_vect, y_train)
# Printing the accuracy on the training data
print("Accuracy on Training Data: ", b_nb_1.score(X_train_vect, y_train))

# Testing and evaluating
b_nb_1_pred = b_nb_1.predict(X_test_vect)

# Importing tools from sklearn for evaluation
from sklearn.metrics import confusion_matrix, classification_report
print("\nResults on testing data:\n")
print("Confusion matrix:")
print(confusion_matrix(y_test, b_nb_1_pred))
print("\nClassification Report:")
print(classification_report(y_test, b_nb_1_pred))
```

```
Accuracy on Training Data:  0.803030303030303

Results on testing data:

Confusion matrix:
[[10  0  0  0  0]
 [ 1  0  0  0  0]
 [ 2  0  0  0  0]
 [ 1  0  0  0  0]
 [ 3  0  0  0  0]]

Classification Report:
                      precision    recall  f1-score   support

            HAMILTON       0.59      1.00      0.74        10
HAMILTON AND MADISON       0.00      0.00      0.00         1
 HAMILTON OR MADISON       0.00      0.00      0.00         2
                 JAY       0.00      0.00      0.00         1
             MADISON       0.00      0.00      0.00         3

            accuracy                           0.59        17
           macro avg       0.12      0.20      0.15        17
        weighted avg       0.35      0.59      0.44        17


C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
```

```
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

1.2.2 Performing tf-idf Vectorization with Adjusted Parameters

[6]:
```python
# Limiting the number of frequent words and adding bigrams
# to improve vectorization for train and test
# Importing our tf-idf vectorizer from sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
# Setting up our maximum number of words as 1000
# stopwords and bigrams for our vectorizer
vectorizer = TfidfVectorizer(stop_words=stopwords, max_features=1000,␣
 ↪ngram_range=(1,2))
# Perform tf-idf vectorization and fit to training data
X_train_vect2 = vectorizer.fit_transform(X_train)
# Transforming the test data with the fitted tf-idf vectorization
X_test_vect2 = vectorizer.transform(X_test)
# Outputting the dimensions of train and test
print("Dimensions of the vectorized train data: ", X_train_vect2.shape)
print("Dimensions of the vectorized test data: ", X_test_vect2.shape)
```

```
Dimensions of the vectorized train data:  (66, 1000)
Dimensions of the vectorized test data:  (17, 1000)
```

1.2.3 The Second Model

[7]:
```python
# Creating Benoulli Naive Bayes model with different vectorization
b_nb_2 = BernoulliNB()
b_nb_2.fit(X_train_vect2, y_train)
# Printing the accuracy on the training data
print("Accuracy on Training Data: ", b_nb_2.score(X_train_vect2, y_train))

# Testing and evaluating
b_nb_2_pred = b_nb_2.predict(X_test_vect2)

print("\nResults on testing data:\n")
print("Confusion matrix:")
print(confusion_matrix(y_test, b_nb_2_pred))
print("\nClassification Report:")
print(classification_report(y_test, b_nb_2_pred))
```

```
Accuracy on Training Data:  1.0

Results on testing data:

Confusion matrix:
[[10  0  0  0  0]
 [ 1  0  0  0  0]
 [ 0  0  1  0  1]
 [ 1  0  0  0  0]
 [ 2  0  0  0  1]]

Classification Report:
                      precision    recall  f1-score   support

            HAMILTON       0.71      1.00      0.83        10
HAMILTON AND MADISON       0.00      0.00      0.00         1
 HAMILTON OR MADISON       1.00      0.50      0.67         2
                 JAY       0.00      0.00      0.00         1
             MADISON       0.50      0.33      0.40         3

            accuracy                           0.71        17
           macro avg       0.44      0.37      0.38        17
        weighted avg       0.63      0.71      0.64        17

C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

1.3 Logistic Regression

1.3.1 With lbfgs

[8]:
```
# Importing our Model
from sklearn.linear_model import LogisticRegression
# Training our model, using lbfgs solver
logreg1 = LogisticRegression(solver='lbfgs', multi_class='multinomial',␣
 ↪random_state=1234, class_weight='balanced')
logreg1.fit(X_train_vect2, y_train)

# Testing and evaluating
logreg1_pred = logreg1.predict(X_test_vect2)

print("\nResults on testing data:\n")
print("Confusion matrix:")
print(confusion_matrix(y_test, logreg1_pred))
print("\nClassification Report:")
print(classification_report(y_test, logreg1_pred))
```

```
Results on testing data:

Confusion matrix:
[[10  0  0  0  0]
 [ 1  0  0  0  0]
 [ 0  0  1  0  1]
 [ 0  0  0  1  0]
 [ 1  0  0  0  2]]

Classification Report:
                      precision    recall  f1-score   support

            HAMILTON       0.83      1.00      0.91        10
HAMILTON AND MADISON       0.00      0.00      0.00         1
 HAMILTON OR MADISON       1.00      0.50      0.67         2
                 JAY       1.00      1.00      1.00         1
             MADISON       0.67      0.67      0.67         3

            accuracy                           0.82        17
           macro avg       0.70      0.63      0.65        17
        weighted avg       0.78      0.82      0.79        17

C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[9]:
```
# Training our model, using liblinear solver
logreg2 = LogisticRegression(solver='liblinear', random_state=1234,␣
 ↪class_weight='balanced', C=10)
logreg2.fit(X_train_vect2, y_train)
# Printing the accuracy on the training data
print("Accuracy on Training Data: ", logreg2.score(X_train_vect2, y_train))

# Testing and evaluating
logreg2_pred = logreg2.predict(X_test_vect2)

print("\nResults on testing data:\n")
print("Confusion matrix:")
print(confusion_matrix(y_test, logreg2_pred))
print("\nClassification Report:")
print(classification_report(y_test, logreg2_pred))
```

```
Accuracy on Training Data:  1.0

Results on testing data:

Confusion matrix:
[[10  0  0  0  0]
 [ 1  0  0  0  0]
 [ 0  0  1  0  1]
 [ 0  0  0  1  0]
 [ 1  0  0  0  2]]

Classification Report:
                      precision    recall  f1-score   support

            HAMILTON       0.83      1.00      0.91        10
HAMILTON AND MADISON       0.00      0.00      0.00         1
 HAMILTON OR MADISON       1.00      0.50      0.67         2
                 JAY       1.00      1.00      1.00         1
             MADISON       0.67      0.67      0.67         3

            accuracy                           0.82        17
```

```
           macro avg       0.70      0.63      0.65        17
        weighted avg       0.78      0.82      0.79        17

C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

1.3.2 With liblinear

1.4 Neural Networks

1.4.1 Neural Network 1

[10]:
```
# Importing the multiclassifier neural network from sklearn
from sklearn.neural_network import MLPClassifier

# Train our model
neural_net1 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(2,1),␣
 ↪random_state=1234)
neural_net1.fit(X_train_vect2, y_train)

# Testing and evaluating
neural_net1_pred = neural_net1.predict(X_test_vect2)

print("\nResults on testing data:\n")
print("Confusion matrix:")
print(confusion_matrix(y_test, neural_net1_pred))
print("\nClassification Report:")
print(classification_report(y_test, neural_net1_pred))
```

```
Results on testing data:

Confusion matrix:
[[10  0  0  0  0]
 [ 1  0  0  0  0]
 [ 2  0  0  0  0]
 [ 1  0  0  0  0]
 [ 3  0  0  0  0]]

Classification Report:
                      precision    recall  f1-score   support

            HAMILTON       0.59      1.00      0.74        10
HAMILTON AND MADISON       0.00      0.00      0.00         1
 HAMILTON OR MADISON       0.00      0.00      0.00         2
                 JAY       0.00      0.00      0.00         1
             MADISON       0.00      0.00      0.00         3

            accuracy                           0.59        17
           macro avg       0.12      0.20      0.15        17
        weighted avg       0.35      0.59      0.44        17

C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

1.4.2 Neural Network 2

[11]:
```python
# Importing the multiclassifier neural network from sklearn
from sklearn.neural_network import MLPClassifier

# Train our model
neural_net2 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(2),␣
 ↪random_state=1234)
neural_net2.fit(X_train_vect2, y_train)

# Testing and evaluating
neural_net2_pred = neural_net2.predict(X_test_vect2)

print("\nResults on testing data:\n")
print("Confusion matrix:")
print(confusion_matrix(y_test, neural_net2_pred))
print("\nClassification Report:")
print(classification_report(y_test, neural_net2_pred))
```

```
Results on testing data:

Confusion matrix:
[[10  0  0  0  0]
 [ 0  0  0  1  0]
 [ 0  0  0  0  2]
 [ 0  0  0  0  1]
 [ 0  0  0  2  1]]

Classification Report:
                      precision    recall  f1-score   support

            HAMILTON       1.00      1.00      1.00        10
HAMILTON AND MADISON       0.00      0.00      0.00         1
 HAMILTON OR MADISON       0.00      0.00      0.00         2
                 JAY       0.00      0.00      0.00         1
             MADISON       0.25      0.33      0.29         3

            accuracy                           0.65        17
           macro avg       0.25      0.27      0.26        17
        weighted avg       0.63      0.65      0.64        17

C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

1.4.3 Neural Network 3

[12]:
```python
# Importing the multiclassifier neural network from sklearn
from sklearn.neural_network import MLPClassifier

# Train our model
neural_net3 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(1,1),␣
  ↪random_state=1234)
neural_net3.fit(X_train_vect2, y_train)

# Testing and evaluating
neural_net3_pred = neural_net3.predict(X_test_vect2)

print("\nResults on testing data:\n")
print("Confusion matrix:")
print(confusion_matrix(y_test, neural_net3_pred))
print("\nClassification Report:")
print(classification_report(y_test, neural_net3_pred))
```

```
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:559:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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)


Results on testing data:

Confusion matrix:
[[10  0  0  0  0]
 [ 0  1  0  0  0]
 [ 0  0  1  0  1]
 [ 0  0  0  0  1]
 [ 0  2  1  0  0]]

Classification Report:
                      precision    recall  f1-score   support

            HAMILTON       1.00      1.00      1.00        10
HAMILTON AND MADISON       0.33      1.00      0.50         1
 HAMILTON OR MADISON       0.50      0.50      0.50         2
                 JAY       0.00      0.00      0.00         1
             MADISON       0.00      0.00      0.00         3

            accuracy                           0.71        17
           macro avg       0.37      0.50      0.40        17
```

```
        weighted avg       0.67      0.71      0.68        17

C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bridg\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

1.5 Analysis

Overall, this task was very challenging to get a good accuracy as the data set is very small with only about 400 observations.

Starting off with the inital Naive Bayes it only had a 59% accuracy which is quite poor. By limiting the number of words used by tf-idf to 1000 and adding bigrams, I was able to get a 71% accuracy. This accuracy isn't great but still a 10% improvement.

Next, the initial Logistic Regression with solver lbfgs initially had the 59% accuracy as well, but adding the 'balanced' and 'multinomial' parameters made it jump to 82%. I think this is because our data set is very unbalanced, with majority of the targets being Hamilton as seen by the Naive Bayes predictions.

Then the second Logistic Regression with solver liblinear was even more tricky to get the accuracy from 59% to 82%. This is because the 'balanced' parameter didn't help very much. I then utilized the C parameter which controls the normalization, the higher the C, the less normalization. This helped significantly and after trying a few values the highest accuracy was with C=10.

Finally, the Neural Networks, the initally topology with (2,1) got the terrible accuracy of 59%. The second I tried to further simplify it to prevent overfitting by using topology (2) and it slightly helped but still only had an accuracy of 65%. Then in the final topology I attempted splitting the layer into to with topology (1,1) which gave me the highest accuracy of 71% for neural networks. This wasn't surprising that it didn't do as well as the logistic regression because neural networks typically need a lot more data and likely more balanced data to perform well.