CS4395 Portfolio Author Attribution Assignment

Shane Arwood

November 13, 2022

This notebook aims to correctly predict the authors of a given text, the NLP task of author attribution. The corpus used is the Federalist Papers written by Alexander Hamilton, James Madison, and John Jay. Sklearn's naive bayes, neural network, and logistic regression, as well as pandas, were used to predict which author wrote each paper.

```
import pandas as pd
import math
from sklearn.naive_bayes import MultinomialNB
from sklearn.neural_network import MLPClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
nltk.download('stopwords')
nltk.download('punkt')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

Step 1: Pandas

```
# read file and convert author column to categorical
fed_papers = pd.read_csv('federalist.csv')
author_data = fed_papers[['author']].astype('category')
print("First few rows of author column: ")
print(author_data.head())
print("\n")
print("Counts by author: ")
print(author_data.value_counts())
```

```
First few rows of author column:
     author
```

```
0  HAMILTON
1       JAY
2       JAY
3       JAY
4       JAY


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

Step 2: Train and Test Split

```
papers = fed_papers.text  # feature
authors = fed_papers.author # target

papers_train, papers_test, authors_train, authors_test = train_test_split(papers,
              authors, test_size=0.2, train_size=0.8, random_state=1234)
print("Train shape: ", papers_train.shape)
print("Test shape: ", papers_test.shape)
```

```
Train shape:  (66,)
Test shape:  (17,)
```

Step 3: tfidf Vectorization

```
stopwords = stopwords.words('english')


# remove stopwords
for i in range(0, 83):
  tokens = word_tokenize(fed_papers.loc[i, 'text'])
  tokens = [t.lower() for t in tokens if t.isalpha()]
  tokens = [t for t in tokens if t not in stopwords]
  fed_papers.loc[i, 'text'] = ' '.join(tokens)


# tf-idf vectorization
vectorizer = TfidfVectorizer()
vectorizer = TfidfVectorizer(stop_words=stopwords)
papers_train_nb = vectorizer.fit_transform(papers_train)
papers_test_nb = vectorizer.transform(papers_test)
```

```
print("Train shape: ", papers_train_nb.shape)
print("Test shape: ", papers_test_nb.shape)
```

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

Step 4: Bernoulli Naïve Bayes model (stopwords removed)

```
# naive bayes model
naive_bayes = MultinomialNB()
naive_bayes.fit(papers_train_nb, authors_train)
pred = naive_bayes.predict(papers_test_nb)
print("Accuracy: ", accuracy_score(authors_test, pred))
```

```
Accuracy:  0.5882352941176471
```

Step 5: Bernoulli Naïve Bayes model (max_features=1000)

```
# tfidf vectorization
vectorizer_bigrams = TfidfVectorizer(max_features=1000)
papers_train_maxft = vectorizer_bigrams.fit_transform(papers_train)
papers_test_maxft = vectorizer_bigrams.transform(papers_test)
print("Train shape: ", papers_train_maxft.shape)
print("Test shape: ", papers_test_maxft.shape)
```

```
Train shape:  (66, 1000)
Test shape:  (17, 1000)
```

```
# naive bayes model
naive_bayes.fit(papers_train_maxft, authors_train)
pred = naive_bayes.predict(papers_test_maxft)
print("Accuracy: ", accuracy_score(authors_test, pred))
```

```
Accuracy:  0.5882352941176471
```

The accuracy from Bernoulli Naïve Bayes on the text with just stopwords removed was the same as from that on the text with max_features set to 1000.

Step 6: Logistic Regression

```
# No parameters for LogisticRegression()
```

```
pipeline_logreg = Pipeline([('tfidf', TfidfVectorizer()),
        ('logreg', LogisticRegression())])

pipeline_logreg.fit(papers_train, authors_train)

pred_logistic = pipeline_logreg.predict(papers_test)

print("Accuracy: ", accuracy_score(authors_test, pred_logistic))
```

```
    Accuracy:  0.5882352941176471
```

```
# Balance parameter for LogisticRegression()
balanced_logreg = LogisticRegression(class_weight='balanced')

pipeline_logreg = Pipeline([('tfidf', TfidfVectorizer()),
        ('logreg', balanced_logreg)])

pipeline_logreg.fit(papers_train, authors_train)

pred_logistic = pipeline_logreg.predict(papers_test)

print("Accuracy: ", accuracy_score(authors_test, pred_logistic))
```

```
    Accuracy:  0.9411764705882353
```

Adding a parameter to the LogisticRegression() model improved the accuracy immensely, an increase from about 0.58 to 0.94. The data for authors was unbalanced, so adding this parameter made the results improve over the unparameterized model.

Step 7: Neural Network trying different topologies

```
topology = [200, 47, 10]
pipeline_nn = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('neuralnet', MLPClassifier(solver='lbfgs', alpha=1e-5,
                   hidden_layer_sizes=topology, random_state=1)),])

pipeline_nn.fit(papers_train, authors_train)

pred_nn = pipeline_nn.predict(papers_test)

print("Accuracy with 200, 47, 10: ", accuracy_score(authors_test, pred_nn))
```

```
    Accuracy with 200, 47, 10:  0.7058823529411765
```

```
topology = [100, 47, 10]
```

```
pipeline_nn = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('neuralnet', MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=topology, random_state=1)),])

pipeline_nn.fit(papers_train, authors_train)

pred_nn = pipeline_nn.predict(papers_test)

print("Accuracy with 100, 47, 10: ", accuracy_score(authors_test, pred_nn))
```

```
     Accuracy with 100, 47, 10:  0.7647058823529411
```

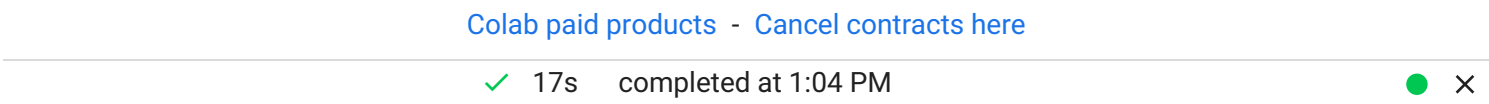
```
topology = [100, 67, 15, 2]
pipeline_nn = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('neuralnet', MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=topology, random_state=1)),])

pipeline_nn.fit(papers_train, authors_train)

pred_nn = pipeline_nn.predict(papers_test)

print("Accuracy with 100, 67, 15, 2: ", accuracy_score(authors_test, pred_nn))
```

```
     Accuracy with 100, 67, 15, 2:  0.5882352941176471
```

```
topology = [60, 45, 10]
pipeline_nn = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('neuralnet', MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=topology, random_state=1)),])

pipeline_nn.fit(papers_train, authors_train)

pred_nn = pipeline_nn.predict(papers_test)

print("Accuracy with 60, 45, 10: ", accuracy_score(authors_test, pred_nn))
```

```
     Accuracy with 60, 45, 10:  0.8823529411764706
```

Final accuracy with neural network: 0.88


Colab paid products - Cancel contracts here
17s completed at 1:04 PM