An Accessible Python based Author Identification Process

SciPy 2023 – July 10-16, 2023

Anthony Breitzman, PhD
Assoc. Professor Computer Science Dept.
Data Science Program Coordinator
Rowan University, Glassboro, NJ, USA

Rowan University

Introduction

- Author identification also known as ‘author attribution’ and more recently ‘forensic linguistics’ involves identifying true authors of anonymous texts.
- The problem has garnered headlines with several high-profile cases that were made possible with computers and text mining methods. In 2017 The Discovery Channel created a TV series called Manhunt:Unabomber that showed how Forensic Linguistics was used to determine that Ted Kaczynski was the author of the Unabomber manifesto [Luu, 2017]
- In 2016 it was revealed that Christopher Marlowe credited as one of Shakespeare’s cowriters [Alberge, 2016]. It was long suspected that Shakespeare collaborated with others, but since Marlowe was always considered his biggest rival, it was quite a surprise that the two collaborated.
- See [Foster, 2000] for other examples including the best seller Primary Colors about the Clinton campaign that was published anonymously and the question of authorship of “Twas the Night Before Christmas” as well as other examples

Introduction

- While forensic linguistics may be a recent name for such attribution, the idea of using statistical modeling to identify authors goes back to at least 1963 when Mosteller and Wallace published their ground-breaking study of the Federalist Papers [Mosteller and Wallace, 1963].

- The above study and subsequent studies require a solid background in statistics to understand.

- In this project, we wish to leverage Python to do attribution analyses easily without requiring a statistics background

- All code related to this project can be found in https://github.com/AbreitzmanSr/SciPy2023-AuthorAttribution.git

Why should we care?

- Is attribution a big problem? Why should I care?

- The methods described here can be leveraged for other text-analysis problems such as:
  - Sentiment analysis
  - Gender identification
  - Genre classification

- Given the amount of text produced daily on the internet, being able to do some basic text analysis via Python is a useful skill

The Federalist Papers

- The Federalist Papers were a series of essays written by Alexander Hamilton, John Jay, and James Madison published under the pseudonym "Publius" in New York newspapers in 1787 and 1788 in support of ratification of the constitution.
- It is surmised that the authors were not anxious to claim the essays for decades because the opinions in the essays sometimes opposed positions each later supported [Mosteller and Wallace, 1963].
- Hamilton was famously killed in a duel in 1804 but he left a list of the essays he wrote with his lawyer before his death.
- Madison later generated his own list a decade later and attributed any discrepancies between the lists as "owing doubtless to the hurry in which (Hamilton's) memorandum was made out" [Adair, 1944].
- Of the 85 essays, the 5 essays written by Jay are not in dispute. Another 51 by Hamilton, 14 by Madison, and 3 joint essays coauthored by Hamilton and Madison are also not in dispute. However, 12 essays (Federalist Nos. 49-58, 62 and 63) were claimed by both Hamilton and Madison in their respective lists [Mosteller and Wallace, 1963].

The Federalist Papers (2)

- Identifying the authors of the disputed Federalist Papers is a good case-study for learning text-mining techniques.
- Most approaches use statistics and machine learning, but for this audience we will show how far we can get with easy-to-use Python tools for text.
- We'll add some easy statistics and machine learning at the end, but we can do the author identification with minimal statistics.

Author's Favorite Words

	% of Fed. Papers Containing Word					Usage per 1000 words				
word	Hamilton	Madison	Joint	Disputed	Jay	Hamilton	Madison	Joint	Disputed	Jay
upon	100.0	21.4	66.6	16.6	20.0	3.012	0.161	0.312	0.112	0.107
on	98.0	100.0	100.0	100.0	100.0	3.037	6.817	6.094	7.077	4.721
very	72.5	85.7	100.0	91.6	60.0	0.583	1.040	0.937	2.209	1.394
kind	68.6	7.1	0.0	8.3	60.0	0.615	0.023	0.000	0.037	0.429
community	62.7	14.2	33.3	25.0	20.0	0.558	0.046	0.156	0.187	0.107
while	39.2	0.0	0.0	0.0	40.0	0.291	0.000	0.000	0.000	0.214
enough	35.2	0.0	33.3	0.0	0.0	0.267	0.000	0.156	0.000	0.000
nomination	13.7	0.0	0.0	0.0	0.0	0.178	0.000	0.000	0.000	0.000
consequently	5.8	57.1	0.0	41.6	40.0	0.032	0.277	0.000	0.337	0.429
lesser	3.9	35.7	0.0	16.6	20.0	0.016	0.161	0.000	0.149	0.107
whilst	1.9	57.1	66.6	50.0	0.0	0.008	0.277	0.312	0.337	0.000
although	1.9	42.8	0.0	33.3	80.0	0.008	0.161	0.000	0.149	0.536
composing	1.9	42.8	33.3	16.6	0.0	0.008	0.254	0.156	0.074	0.000
recommended	1.9	35.7	0.0	8.3	20.0	0.008	0.138	0.000	0.037	0.429
sphere	1.9	35.7	0.0	16.6	0.0	0.008	0.184	0.000	0.112	0.000
pronounced	1.9	28.5	0.0	16.6	0.0	0.008	0.115	0.000	0.074	0.000
respectively	1.9	28.5	0.0	16.6	0.0	0.008	0.138	0.000	0.074	0.000
enlarge	0.0	28.5	0.0	16.6	0.0	0.000	0.115	0.000	0.074	0.000
involves	0.0	28.5	0.0	16.6	0.0	0.000	0.092	0.000	0.074	0.000
stamped	0.0	28.5	33.3	0.0	0.0	0.000	0.092	0.156	0.000	0.000
crushed	0.0	21.4	0.0	8.3	0.0	0.000	0.069	0.000	0.037	0.000
democratic	0.0	21.4	0.0	8.3	0.0	0.000	0.069	0.000	0.037	0.000
dishonorable	0.0	21.4	0.0	8.3	0.0	0.000	0.069	0.000	0.037	0.000
precision	0.0	21.4	0.0	8.3	0.0	0.000	0.069	0.000	0.037	0.000
reform	0.0	21.4	33.3	16.6	0.0	0.000	0.161	0.156	0.074	0.000
transferred	0.0	21.4	0.0	8.3	0.0	0.000	0.069	0.000	0.037	0.000
universally	0.0	21.4	0.0	8.3	20.0	0.000	0.069	0.000	0.037	0.107
bind	0.0	14.2	0.0	8.3	20.0	0.000	0.069	0.000	0.037	0.107
derives	0.0	14.2	33.3	8.3	0.0	0.000	0.069	0.156	0.037	0.000
drawing	0.0	14.2	0.0	8.3	0.0	0.000	0.069	0.000	0.037	0.000
function	0.0	14.2	0.0	8.3	0.0	0.000	0.069	0.000	0.037	0.000
inconveniency	0.0	14.2	0.0	16.6	0.0	0.000	0.069	0.000	0.074	0.000
obviated	0.0	14.2	0.0	8.3	0.0	0.000	0.069	0.000	0.037	0.000
patriotic	0.0	14.2	0.0	25.0	20.0	0.000	0.069	0.000	0.112	0.107
speedy	0.0	14.2	0.0	8.3	0.0	0.000	0.069	0.000	0.037	0.000

- We start with a key table that will suggest that Madison is the author of most if not all of the disputed papers.
- Note that Hamilton uses 'upon' many times in place of 'on'. In the disputed papers both terms are used at the Madison rate rather than the Hamilton rate.
- Madison uses 'whilst' instead of 'while'. While is never used in the disputed papers but 'Whilst' is used in half of them.
- Several words like 'democratic', 'dishonorable', 'precision', 'inconveniency', etc. don't show up in any Hamilton documents but show up in the disputed papers and Madison documents.
- 'While', 'enough', 'nomination', 'kind' appear in Hamilton documents but either not at all in the disputed papers or at the Madison rate within the disputed papers

How did we get the Previous Table?

- Generating the previous table is an example of what Data Scientists call **Exploratory Data Analysis** which is an initial investigation on data to discover patterns and trends, spot anomalies, and generate statistical summaries which might help us check assumptions and perform hypotheses about our data.

- The previous table suggests Madison is the likely author of most of the disputed Federalist Papers. But the table did materialize out of nowhere.

- There are 2 key components to the previous table:
  - We need to identify words that have a high probability of being used by one author but not the other.
  - We need a way to identify usage per 1000 words for each author

- Both of those components are easily done using Python's NLTK (Natural Language Tool-kit) library

Get Federalist Papers from Project Gutenberg

[https://www.gutenberg.org/]

```
#Get Federalist Papers
from urllib import request
url = "https://www.gutenberg.org/cache/epub/1404/pg1404.txt"

response = request.urlopen(url)
raw = response.read()
text = raw.decode("utf-8-sig")
```

```
#slicing routines
def left(s, amount):
    return s[:amount]

def right(s, amount):
    return s[-amount:]
```

```
#replace multiple spaces with single space
import re
text = re.sub("\s+", " ", text)
```

```
#kill front matter
text = right(text,len(text)-text.find('FEDERALIST No.'))
```

```
#kill back matter
text = left(text,text.find('*** END OF THE PROJECT GUTENBERG'))
```

Rowan University

Create a List of Each Author's Federalist Papers

```
#Break Federalist papers up into individual texts
FedChapters = []
text5 = text
i = text5.find('FEDERALIST No.')
while(i >= 0):
    FedChapters.append((left(text5,i)).strip())
    text5 = right(text5,len(text5)-(i+14))
    i = text5.find('FEDERALIST No.')

FedChapters.append(text5.strip())
```

```
#returns the main text of a Federalist paper.
def getFedText(s):
  if (len(s)>0):
    t = s + ' PUBLIUS' #in case it's not there (but in most cases it is)
    i = t.find('PUBLIUS')
    t = left(t,i)
    i = t.find('State of New York')
    t = right(t,len(t)-(i+19))
    return t.strip()
  else:
    return ""
```

```
#Store Hamilton's Federalist papers in a Hamilton list, Madison's in a Madison list, etc. for Jay, joint, disputed papers
hamilton = []
jay = []
madison = []
joint = []
disputed = []
for i in range(len(FedChapters)):
    if (i in {2,3,4,5,64}):
        jay.append([i,[getFedText(FedChapters[i])]])
    else:
        if (i in {18,19,20}):
            joint.append([i,[getFedText(FedChapters[i])]])
        else:
            if (i in {49,50,51,52,53,54,55,56,57,58,62,63}):
                disputed.append([i,[getFedText(FedChapters[i])]])
            else:
                if (i in {10,14,37,38,39,40,41,42,43,44,45,46,47,48}):
                    madison.append([i,[getFedText(FedChapters[i])]])
                else:
                    if (i > 0):
                        hamilton.append([i,[getFedText(FedChapters[i])]])
```

Intro to NLTK tokenizers

```
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize

text_2sentences = "The quick brown fox jumped over the lazy dog. A short sentence."

sentences = sent_tokenize(text_2sentences)

for x in sentences:
    print(word_tokenize(x))
```

```
['The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy', 'dog', '.']
['A', 'short', 'sentence', '.']
```

- NLTK makes it easy to make lists of sentences, lists of words, count sentences, count words in sentences etc.
- Next slide shows code for making a list of word frequency dictionaries for each author

Make List of Word Frequency Dictionaries for each Author

```
hamiltonDicts = [] #list of dictionaries containing word freq for each of Hamilton's Federalist Papers
madisonDicts = []
jayDicts = []
disputedDicts = []
jointDicts = []

def getDocDict(str1):
#returns a dictonary containing frequencies of any word in string
#e.g. str1 = 'quick brown fox is quick.'
# returns {quick:2, brown:1, fox:1, is:1}
  x = {}
  words = word_tokenize(str1.lower().strip())
  for b in words:
        if b in x:
            x[b]+=1
        else:
            x[b]=1
  return(x)

for a in hamilton:
    hamiltonDicts.append(getDocDict(a[1][0]))

for a in madison:
    madisonDicts.append(getDocDict(a[1][0]))

for a in jay:
    jayDicts.append(getDocDict(a[1][0]))

for a in joint:
    jointDicts.append(getDocDict(a[1][0]))

for a in disputed:
    disputedDicts.append(getDocDict(a[1][0]))

print(len(hamiltonDicts),len(madisonDicts),len(jayDicts),len(jointDicts),len(disputedDicts))
```

```
51 14 5 3 12
```

Rowan University

SciPy 2023

What we know so far….

- It is now straightforward to identify word usage for each author
- That is, given a word such as ‘upon’ it is easy to identify the percent of each author’s Federalist papers that mention ‘upon’
- It’s also easy to identify the usage of ‘upon’ per thousand words for each author

- What we haven’t addressed is how to find words that are favorites of Hamilton but not Madison and vice-versa.

- What we will be doing is building a Naïve Bayes dictionary for each author, but we will assume no prior knowledge of Naïve Bayes to do so

Build a Dictionary of all Words and Identify Useful Words

```
completeDict = {} #dictionary containing every word along with
                  #documeny frequency
                  #(e.g. # of fed papers containing word)

kills = [',','.',"'",'',';','-',')','(']
authDicts = [hamiltonDicts,madisonDicts,jointDicts,disputedDicts]
for authDict in authDicts:
  for a in authDict:
    for x in a:
        if (x not in kills):
         if x in completeDict:
            completeDict[x]+=1
         else:
            completeDict[x]=1

trimDict = set()  #subset of completeDict that contains useful words
for a in completeDict:
    x = completeDict[a]
    if (x >= 3 and x < 80):
        trimDict.add(a)

print(len(completeDict),len(trimDict))
```

```
8492 3967
```

- The code on the left creates a frequency distribution of every word mentioned in the Federalist Papers
- We then remove any word that is only used once or twice because it will have no discriminating value

- We also remove any word that is mentioned in all the documents. This will get rid of words like 'is', 'and', 'the', but preserve words like 'while' and 'whilst' that are used by Hamilton and not Madison or vice-versa.

Rowan University

SciPy 2023

Finding Interesting Words

- At this point:
  - completeDict contains word frequencies for the 8,492 unique words in all the Federalist papers.
  - trimDict contains the subset of 3,967 potentially useful words.

- We now need to find words that are much more likely to be used by Hamilton than Madison and vice-versa

- For each word in trimDict we will compute the probability that Hamilton or Madison used it.

- The words where Hamilton's probability is 5+ times more likely than Madison (or vice-versa) is an interesting word that gets selected for the previously shown table

Finding Interesting Words (2)

```
#build Naive Bayes Dictionaries for Hamilton, Madison
hamiltonNBwordDicts = {}
madisonNBwordDicts = {}

#do Laplace Smoothing first.
for word in trimDict:
    hamiltonNBwordDicts[word]=1
    madisonNBwordDicts[word]=1

for dictionary in hamiltonDicts:
    for word in dictionary:
        if (word in trimDict):
            hamiltonNBwordDicts[word]+=dictionary[word]

for dictionary in madisonDicts:
    for word in dictionary:
        if (word in trimDict):
            madisonNBwordDicts[word]+=dictionary[word]


hamiltonNBdenom = madisonNBdenom = 0
for x in hamiltonNBwordDicts:
    hamiltonNBdenom += hamiltonNBwordDicts[x]
for x in madisonNBwordDicts:
    madisonNBdenom += madisonNBwordDicts[x]
```

- Again, we don't need to know about Naïve Bayes or Laplace smoothing. We use the terms in the code to the left for those that are familiar

- For those unfamiliar with Naïve Bayes we are just doing the following:
  - Computing word frequencies of the potentially useful terms (trimDict) for each author
  - Making sure no word probability is 0. (This is called Laplace Smoothing, but essentially we're trying to avoid cases where Hamilton uses a word very few times but Madison uses it 0 times (or vice-versa) because that will pollute our table with a bunch of useless words
  - We need a denominator (consisting of counts of all words) in order to compute a probability of an author using the word

Finding Interesting Words (3)

```
interesting = []
for i,a in enumerate(trimDict):
    h1 = hamiltonNBwordDicts[a]/hamiltonNBdenom
    m1 = madisonNBwordDicts[a]/madisonNBdenom
    if (m1/h1 > 5 or h1/m1 > 5):
      print(a,h1/m1,m1/h1)
      interesting.append(a)
```

upon 17.940597001516863 0.05573950520796219
kind 11.22486550362285 0.0890879271272558
while 8.202786329570543 0.1219097950162448
intended 7.986923531423951 0.12520465434100816
community 7.663129334204061 0.1304949918483747
enough 7.555197935130765 0.1323592060176371 8
democracy 0.07195426604886443 13.897716631851905
whilst 0.0925126277771114 10.809335158107038
assumed 0.09593902139848591 10.423287473888928
reform 0.09593902139848591 10.423287473888928
composing 0.09962898375996612 10.03723978967082
coin 0.10158249324545568 9.844215947561764
indirectly 0.10793139907329664 9.265144421234604
.
.
.

- Here we see that Hamilton is 18 times more likely to use ‘upon’ as Madison and Madison is 11 times more likely to use ‘whilst’ than Hamilton.
- Finally, after 17 slides we have everything needed to generate the table in slide 7
- At this point, we’ve built the infrastructure needed to do our exploratory data analysis just using Python and some basic probability.
- Next step is to build a predictive model

Rowan University

SciPy 2023

Introduction to Naïve Bayes

- The interested reader can refer to [Jurafsky and Martin, 2023] or many other text-books for a derivation of Naïve Bayes.
- For our purposes we only care that:

  P(Author|word1,word2,…,wordN) = P(word1|Author)*P(word2|Author)*…*P(wordN|author)/k
- That is the conditional probability that a text (word1 through wordN) is authored by 'Hamilton' or 'Madison' is equal to the product of the probabilities of each word belonging to the authors then divided by a constant k. (The equality is only true if the words are independent. Since we don't care about the actual probabilities, but only which author has the larger value, we don't need independence.)
- The constant k is actually another probability that is hard to compute, but since it's the same for both authors all we really need is the following pseudocode:

```
Text = [word1, word2, …, wordN]
if (P(word1|Hamilton)*P(word2|Hamilton)*…*P(wordN|Hamilton) >
    P(word1|Madison)*P(word2|Madison)*…*P(wordN|Madison))
            return(Hamilton)
else
            return(Madison)
```

Rowan University

SciPy 2023

Introduction to Naïve Bayes

- Since we are computing the product of thousands of very small values there is a risk of underflow so instead of the product of many small constants we compute the sum of the logs of many small constants (e.g. Log(a*b) = Log(a) + Log(b)). Thus, the Python code looks like the following:
- We check the accuracy by looking at the predictions for all the known Hamilton papers and all the known Madison papers.
- When we check using the reduced dictionary trimDict (shown in slide 14) we see that it correctly predicts the known Federalist papers.
- The next thing to check is the 12 disputed papers and see if they are attributed to Madison as in [Mosteller and Wallace, 1963].
- For those familiar with machine learning or data mining we call the known Federalist papers, the 'training' set and the disputed papers the 'test' set.

```
#given a document return 'hamilton' if NaiveBayes prob
#suggests Hamilton authored it. similarly return
#'madison' if he is the likely author
def NB_federalist_predict(docDict,vocab1=trimDict):
  h_pr = m_pr = 0
  for word in docDict:
    if (word in vocab1):
        h_pr += float(docDict[word])*(math.log(
           hamiltonNBwordDicts[word]/hamiltonNBdenom))
        m_pr += float(docDict[word])*(math.log(
           madisonNBwordDicts[word]/madisonNBdenom))

  if (h_pr > m_pr):
        return('hamilton')
  else:
        return('madison')

def check_accuracy(vocab1=trimDict):
    right = wrong = 0
    for a in hamiltonDicts:
        if NB_federalist_predict(a,vocab1)=='hamilton':
            right+=1
        else:
            wrong+=1

    for a in madisonDicts:
        if NB_federalist_predict(a,vocab1)=='madison':
            right+=1
        else:
            wrong+=1
    return([100*right/(right+wrong),right,wrong])

print('% correct:',check_accuracy()[0])
```

```
% correct: 100.0
```

Naïve Bayes Results for Several Word Subsets

```
#the following checks accuracy on the training set and then
#identifies how many of the disputed papers are by each author
def report1(words=trimDict):
    if (len(words)<10):
        print(words)
    print(str(check_accuracy(words)[0])+'% accuracy')
    madison = hamilton = 0
    for a in disputedDicts:
        if (NB_federalist_predict(a,words)=='madison'):
            madison+=1
        else:
            hamilton+=1
    print("disputed papers: madison:"+str(madison)+
          ', hamilton:'+str(hamilton)+'\n')

report1(interesting)
report1(['although','composing','involves',
         'confederation','upon'])
report1(['although','obviated','composing',
         'whilst','consequently','upon'])
report1(['against','within','inhabitants',
         'whilst','powers','upon','while'])
report1(['against','upon','whilst',
         'inhabitants','within'])
report1(['against','within','inhabitants',
         'whilst','upon'])
report1(['against','while','whilst','upon','on'])
report1(['concurrent','upon','on',
         'very','natural'])
report1(['while','upon','on','inconveniency'])
```

```
100.0% accuracy
disputed papers: madison:12, hamilton:0

['although', 'composing', 'involves', 'confederation', 'upon']
100.0% accuracy
disputed papers: madison:12, hamilton:0

['although', 'obviated', 'composing', 'whilst', 'consequently', 'upon']
96.92307692307692% accuracy
disputed papers: madison:12, hamilton:0

['against', 'within', 'inhabitants', 'whilst', 'powers', 'upon', 'while']
100.0% accuracy
disputed papers: madison:12, hamilton:0

['against', 'upon', 'whilst', 'inhabitants', 'within']
96.92307692307692% accuracy
disputed papers: madison:12, hamilton:0

['against', 'within', 'inhabitants', 'whilst', 'upon']
96.92307692307692% accuracy
disputed papers: madison:12, hamilton:0

['against', 'while', 'whilst', 'upon', 'on']
96.92307692307692% accuracy
disputed papers: madison:12, hamilton:0

['concurrent', 'upon', 'on', 'very', 'natural']
98.46153846153847% accuracy
disputed papers: madison:12, hamilton:0

['while', 'upon', 'on', 'inconveniency']
95.38461538461539% accuracy
disputed papers: madison:12, hamilton:0
```

- We notice for the so-called ‘interesting’ vocabulary from slide 17 as well as several subsets of the author favorite words from slide 7 that Naïve Bayes correctly identifies the known Federalist Papers with 95%+ accuracy and suggests Madison for the 12 disputed papers

Other Models from Sci-Kit Learn [Pedregosa, 2011]

- Python's Sci-Kit Learn library allows programmers to use machine learning models without much knowledge of machine learning.

- On the next slide we create an array of multiple models and run 5-fold cross-validation on the training set (known Federalist papers) before running each model on the test set of disputed papers.

- We include this to show how easy it is with Python to run multiple models that are more sophisticated than our hand-built Naïve Bayes model in only about 30 lines of code.

- We see on the next slide that all the models predict Madison to be the author of the disputed Federalist Papers

Other Models from Sci-Kit Learn [Pedregosa, 2011]

```
def mPercent(results):
 mcount = 0
 tcount = 0
 for a in results:
    if (a == 'm'):
        mcount+=1
    tcount+=1
 print('% Disputed attributed to Madison:',100.0*mcount/tcount)
 print()
```

```
"""
Build and test multiple models via SKlearn.
X is a dataframe consisting of known Hamilton and Madison papers.
y is a data frameconsisting of author labels.
X_test is a dataframe consisting of disputed
papers
"""

smallVocab5 = ['against','within','inhabitants','whilst','upon']
tfidf = sklearn.feature_extraction.text.TfidfVectorizer(analyzer="word",
                                                        binary=False,
                                                        min_df=2,
                                                        vocabulary=smallVocab5)

X_transformed = tfidf.fit_transform(X)
lb = sklearn.preprocessing.LabelEncoder()
y_transformed = lb.fit_transform(y)
X_test_transformed = tfidf.transform(X_test)

models = [
  KNeighborsClassifier(3),
  DecisionTreeClassifier(max_depth=5),
  RandomForestClassifier(n_estimators=25,max_depth=3),
  LinearSVC(),
  SVC(gamma=2, C=1),
  ComplementNB(),
  AdaBoostClassifier()
]
CV = 5
cv_df = pd.DataFrame(index=range(CV * len(models)))
for model in models:
  model_name = model.__class__.__name__
  accuracies = cross_val_score(model, X_transformed,y_transformed,scoring='accuracy',cv=CV)
  avgAccur = 0
  for fold_idx, accuracy in enumerate(accuracies):
    print(model_name,"fold:",fold_idx,"accuracy:",str(accuracy)[:5])
  print(model_name,"avg accuracy:",str(accuracies.mean())[:5])
  model.fit(X_transformed, y_transformed)
  y_final_predicted = model.predict(X_test_transformed)
  y_final_predicted_labeled =lb.inverse_transform(y_final_predicted)
  mPercent(y_final_predicted_labeled)
```

```
KNeighborsClassifier fold: 0 accuracy: 1.0
KNeighborsClassifier fold: 1 accuracy: 1.0
KNeighborsClassifier fold: 2 accuracy: 1.0
KNeighborsClassifier fold: 3 accuracy: 1.0
KNeighborsClassifier fold: 4 accuracy: 1.0
KNeighborsClassifier avg accuracy: 1.0
% Disputed attributed to Madison: 100.0

DecisionTreeClassifier fold: 0 accuracy: 1.0
DecisionTreeClassifier fold: 1 accuracy: 0.846
DecisionTreeClassifier fold: 2 accuracy: 1.0
DecisionTreeClassifier fold: 3 accuracy: 1.0
DecisionTreeClassifier fold: 4 accuracy: 1.0
DecisionTreeClassifier avg accuracy: 0.969
% Disputed attributed to Madison: 100.0

RandomForestClassifier fold: 0 accuracy: 1.0
RandomForestClassifier fold: 1 accuracy: 0.923
RandomForestClassifier fold: 2 accuracy: 1.0
RandomForestClassifier fold: 3 accuracy: 1.0
RandomForestClassifier fold: 4 accuracy: 1.0
RandomForestClassifier avg accuracy: 0.984
% Disputed attributed to Madison: 100.0

LinearSVC fold: 0 accuracy: 1.0
LinearSVC fold: 1 accuracy: 1.0
LinearSVC fold: 2 accuracy: 1.0
LinearSVC fold: 3 accuracy: 1.0
LinearSVC fold: 4 accuracy: 1.0
LinearSVC avg accuracy: 1.0
% Disputed attributed to Madison: 100.0

SVC fold: 0 accuracy: 1.0
SVC fold: 1 accuracy: 1.0
SVC fold: 2 accuracy: 1.0
SVC fold: 3 accuracy: 1.0
SVC fold: 4 accuracy: 1.0
SVC avg accuracy: 1.0
% Disputed attributed to Madison: 100.0

ComplementNB fold: 0 accuracy: 0.923
ComplementNB fold: 1 accuracy: 1.0
ComplementNB fold: 2 accuracy: 1.0
ComplementNB fold: 3 accuracy: 1.0
ComplementNB fold: 4 accuracy: 1.0
ComplementNB avg accuracy: 0.984
% Disputed attributed to Madison: 100.0

AdaBoostClassifier fold: 0 accuracy: 1.0
AdaBoostClassifier fold: 1 accuracy: 0.846
AdaBoostClassifier fold: 2 accuracy: 1.0
AdaBoostClassifier fold: 3 accuracy: 1.0
AdaBoostClassifier fold: 4 accuracy: 1.0
AdaBoostClassifier avg accuracy: 0.969
% Disputed attributed to Madison: 100.0
```

One Last Model

```
#return usage rate per 1000 words of a target word
#e.g. if target=='upon' appears 3 times in a 1500
#word essay, we return a rate of 2 per 1000 words.
def rate_per_1000(docDict,target):
    if (target in docDict):
        wordCount=0
        for a in docDict:
            wordCount+=docDict[a]
        return(1000*docDict[target]/wordCount)
    else:
        return(0)

def federalist_decison_tree(docDict):
    if ('while' in docDict):
        return('hamilton')
    else:
        if (rate_per_1000(docDict,'whilst') >= .25):
                return('madison')
        if (rate_per_1000(docDict,'upon') >= .9):
                return('hamilton')
        else:
                return('madison')


right = wrong = 0
for a in hamiltonDicts:
    if (federalist_decison_tree(a)=='hamilton'):
        right+=1
    else:
        wrong+=1

for a in madisonDicts:
    if (federalist_decison_tree(a)=='madison'):
        right+=1
    else:
        wrong+=1

madisonCount = 0
for a in disputedDicts:
    if (federalist_decison_tree(a)=='madison'):
       madisonCount += 1

print('accuracy:'+str(right/(right+wrong))+
      ', % disputed attributed to Madison:'+
     str(100*madisonCount/12.0))
```

```
accuracy:1.0, % disputed attributed to Madison:100.0
```

- Some readers may wonder just how small we can make the word list and get 100% accuracy on the training set
- At the risk of overfitting, we build a simple decision tree using just the words ‘while’, ‘whilst’, and ‘upon’
- Usually, we know our model is overfit if the training accuracy exceeds the testing accuracy however this model has 100% accuracy for both if we believe previous authors results that suggest Madison wrote the 12 disputed papers
- We include this one for fun and do not recommend anyone attempt author attribution using just 3 words

Summary

- We show multiple methods of author attribution using Python tools and no prior knowledge of any advanced statistics
- We start by showing how to do Exploratory Data Analysis using NLTK and Python to identify favorite words of an author
- We then show how to build an author dictionary and build a simple Naïve Bayes model
- We introduce the notion of a training and testing set and then show how easy it is with Sci-Kit Learn to run multiple models that are more complex than our hand-built Naïve Bayes model
- In each case we confirm the [Mosteller and Wallace, 1963] result that Madison is the likely author of the Federalist papers
- All code related to this project can be found in https://github.com/AbreitzmanSr/SciPy2023-AuthorAttribution.git

References

1. C. Luu, “Fighting words with the unabomber,” JSTOR.org, 2017.
2. D. Alberge, “Christopher marlowe credited as one of shakespeare’s cowriters,” The Guardian, 2016.
3. F. Mosteller and D. L. Wallace, “Inference in an authorship problem,” Journal of the American Statistical Association, pp. 275–309, 1963.
4. D. Adair, “The authorship of the disputed federalist papers,” The William and Mary Quarterly, pp. 97–122, 1944.
5. D. Adair, “The authorship of the disputed federalist papers: Part ii,” The William and Mary Quarterly, p. 235–264, 1944.
6. F. Mosteller, “A statistical study of the writing styles of the authors of "the federalist" papers,” Proceedings of the American Philosophical Society, 1987.
7. E. Loper and S. Bird, “NLTK: The natural language toolkit,” 2002.
8. J. Grimmer, M. Roberts, and B. Stewart, Text as Data: A New Framework for Machine Learning and the Social Sciences. Princeton University Press, 2022.
9. D. Jurafsky and J. Martin, Speech and Language Processing (Draft of 3rd Ed.). 2023.
10. A. Hamilton, J. Jay, and J. Madison, “The project gutenberg ebook of the federalist papers.” Available at https://www.gutenberg.org/cache/epub/1404/pg1404.txt .
11. F. Pedregosa et al., “Scikit-learn: Machine learning in python,” Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
12. J. Rennie, L. Shih, J. Teevan, and D. Karger, “Tackling the poor assumptions of naive bayes text classifiers,” ICML, vol. 3, pp. 616–623, 2023.
13. Foster, D., “Author Unknown: On the Trail of Anonymous,” Henry Holt and Company, New York, 2000.

Rowan University