Stylometric Analysis using PCA

Group Members

- Yogant Patil 1019101]
- Harshvardhan Singh 1019161]
- Sanchita Mehetre 1019170

Department of Computer Engineering
Fr. C. Rodrigues Institute of Technology, Vashi

1 Introduction

Stylometry is the quantitative study of literary style through computational distant reading methods. It is based on the observation that authors tend to write in relatively consistent, recognisable and unique ways. For example:

- Each person has their own unique vocabulary, sometimes rich, sometimes limited. Although a larger vocabulary is usually associated with literary quality, this is not always the case. Ernest Hemingway is famous for using a surprisingly small number of different words in his writing,1 which did not prevent him from winning the Nobel Prize for Literature in 1954.
- Some people write in short sentences, while others prefer long blocks of text consisting of many clauses.
- No two people use semicolons, em-dashes, and other forms of punctuation in the exact same way.

one of the most common applications of stylometry is in authorship attribution. Given an anonymous text, it is sometimes possible to guess who wrote it by measuring certain features, like the average number of words per sentence or the propensity of the author to use !while" instead of !whilst", and comparing the measurements with other texts written by the suspected author.
For our project, we will be using Principal component analysis. Principal component analysis is another useful approach to take with stylometric analysis. It uses the same initial data as HCA (matrix of word frequencies), but we use linear algebra to get abstracted axes that show us where the variance in the data.

2 Problem Analysis

A. Analysis on Sherlock Holmes documents.

B. Our objective is to find out when are the documents written, in 1891 or 1892.

C. Since it is difficult plot the documents and attribute the time, we apply PCA because there are too many dimensions to handle.

3 Steps to Perform Stylometric Analysis

3.1 styl_basic_pca.py

Step 1: Importing PCA modal from sklearn. Decomposition.

Step 2: Setting up some information to use. In what year each of these stories are written? Step 3: Setting

colours for each year.

Step 4: Making array dense and perform PCA on the countMatrix. Step 5:

Coloring the data by year so that we can get unique items.

Step 6: Make a dictionary by using a syntax dict(zip(uniqueYears, numberForClasses))

Step 7: Make a new representation for each document and it hast to be NumPy array and make list of colours after that.

Step 8: Plot the graph

3.2 styl_fed.py

Step 1: Running the analysis again, but with the federalist papers.

Step 2: Create a dictionary with the authorship information Step 3:

Perform PCA on the countMatrix

Step 4: Finding unique author and get a number for each class.

Step 5: Make a new representation for each document and it hast to be NumPy array and make list of colours after that.

Step 6: Labelling individual points so we know which document they are.

Step 7: Plot the graph

3.3 loadings.py

In order to help with interpretation , it is often useful to plot the component loadings, which will show us how individual words influence where documents appear in the plot.

Step 1: It is same as above codes but instead of plotting the text by themselves we will add the words themselves onto the plot.

Step 2: Plot the information.

Step 3: When plotting the information we will make the dots smaller and transparent so that we can see the words better.

Step 4: Get the component loading from the PCA object not the fit transformed object called myPCA. Step 5: Get the

vocabulary from the countVectorizer and add them to the plot.

Step 7: Apply plt.legend() and plt.show().

3 Implementation

An implementation to this problem is made using Python and the NLTK library.
Code:

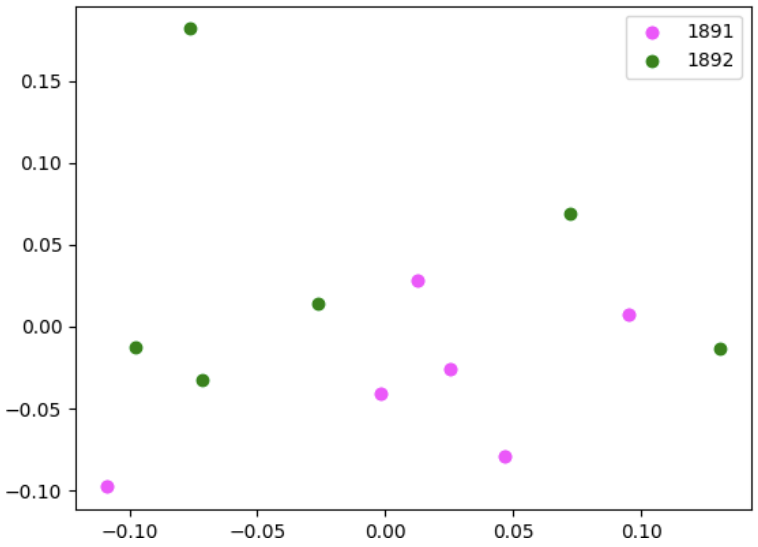
styl_basic_pca.py

```
import re, nltk, os
from pandas import DataFrame import
numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer from
sklearn.decomposition import PCA
import matplotlib.pyplot as plt
# First, lets set up some information to use. In what year was each of these #stories
written?
storyYear = {"Engineer's Thumb":1892,"Red-headed League":1891,"Twisted Lip":1891,
        "Identity":1891,"Noble Bachelor":1892,"Beryl Coronet":1892,
        "Orange Pips":1891,"Blue Carbuncle":1892,"Copper Beeches":1892, "Boscombe
        Valley":1891,"Bohemia":1891, "Speckled Band":1892}
yearColor = {1891:"magenta",1892:"green"}\
ignoreFiles = set([".DS_Store","LICENSE","README.md"])
sherlockTexts = []sherlockTitles = []
for root, dirs, files in os.walk("corpus"): for filename
   in files:
      if filename not in ignoreFiles:
         with open(os.path.join(root,filename)) as rf:
            sherlockTexts.append(rf.read().lower())
            sherlockTitles.append(filename[:-4].lower())

shortenTitle = {"the adventure of the engineer's thumb":"Engineer's Thumb",
             'the red-headed league':"Red-headed League", 'the man with the twisted lip':"Twisted
Lip",'a      of identity':"Identity", 'the adventure of the noble bachelor':"Noble Bachelor", 'the adventure of the
case         beryl coronet': "Beryl Coronet", 'the adventure of the speckled
band':"Speckled Band",
                  'the five orange pips':"Orange Pips", 'the adventure of the blue carbuncle':"Blue
```

```
Carbuncle",
                'the adventure of the copper beeches':"Copper Beeches", 'the boscombe valley
mystery':"Boscombe Valley",
                'a scandal in bohemia':"Bohemia"}
shortTitles = [shortenTitle[title] for title in sherlockTitles] countVectorizer =
TfidfVectorizer(max_features=1000, use_idf=False) countMatrix =
countVectorizer.fit_transform(sherlockTexts)
array countMatrix = countMatrix.toarray()
# Lets perform PCA on the countMatrix:
pca = PCA(n_components=2)
myPCA = pca.fit_transform(countMatrix)
uniqueYears = [1891,1892]
numberForClass = [0,1]
yearForClassNumber = dict(zip(uniqueYears,numberForClass))
textClass = np.array([yearForClassNumber[storyYear[s]] for s in shortTitles])
print(textClass)
print(textClass==0)
colors = [yearColor[year] for year in uniqueYears]
for col, classNumber, year in zip(colors, numberForClass, uniqueYears):
plt.scatter(myPCA[textClass==classNumber,0],myPCA[textClass==classNumber,1],label=year,c=col)
plt.legend() plt.show()
```

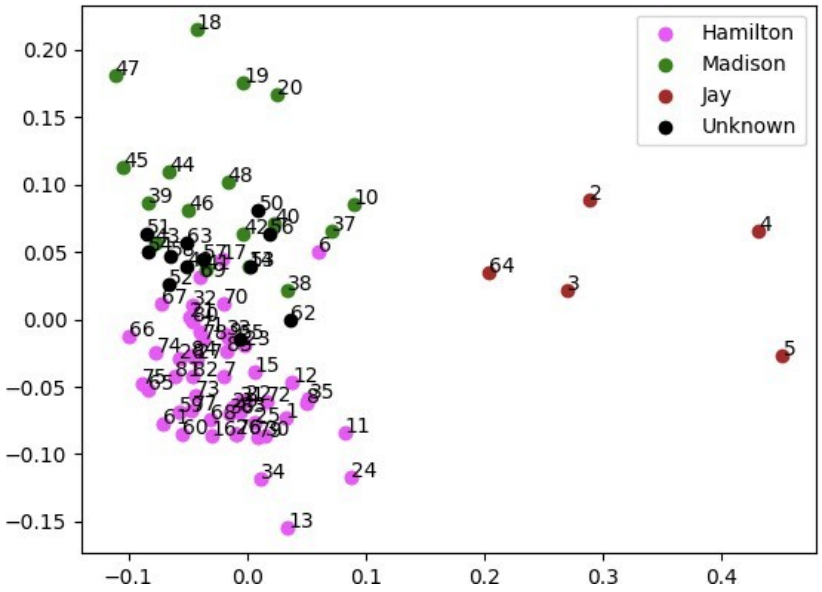
OUTPUT:



styl_fed.py

```
# Let's run the analysis again, but this time with the federalist papers import re,
nltk, os
from pandas import DataFrame import
numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer from
sklearn.decomposition import PCA
import matplotlib.pyplot as plt
rf = open("federalist.txt","r",encoding="utf8") text =
rf.read()
rf.close()
text = text[:text.rfind("End of the Project Gutenberg")]
papers = re.split(r"FEDERALIST\.? No\.? \d+", text)
papers = papers[1:]
federalistnum = [i+1 for i in range(len(papers))]
|authorship={}
for i in range(len(papers)): fno = i+1
    if fno==10 or fno==14 or (fno>=18 and fno<=20) or (fno>=37 and fno<=48):
        authorship[str(fno)] = "Madison"
    elif (fno>=2 and fno<=5) or fno==64:
        authorship[str(fno)] = "Jay"
    elif (fno>=49 and fno<=58) or fno==62 or fno==63:
        authorship[str(fno)] = "Unknown"
    else:
        authorship[str(fno)] = "Hamilton"
authorColor = {"Hamilton":"magenta","Madison":"green","Jay":"brown","Unknown":"black"}
countVectorizer = TfidfVectorizer(max_features=1000, use_idf=False)
countMatrix = countVectorizer.fit_transform(papers) countMatrix
= countMatrix.toarray()
# Lets perform PCA on the countMatrix:
pca = PCA(n_components=2)
myPCA = pca.fit_transform(countMatrix)
# Now we need the Unique Authors
uniqueAuthors = ["Hamilton", "Madison", "Jay", "Unknown"] # Let's get a
number for each class
numberForClass = [i for i in range(len(uniqueAuthors))]
# Make a dictionary! This is new sytax for us! It just makes a dictionary where # the keys are
the unique years and the values are found in numberForClass authForClassNumber =
dict(zip(uniqueAuthors,numberForClass))
# and it needs to be a numpy array
textClass = np.array([authForClassNumber[authorship[str(f)]] for f in federalistnum])
# Make a list of the colors
colors = [authorColor[auth] for auth in uniqueAuthors]
for col, classNumber, year in zip(colors, numberForClass, uniqueAuthors):
plt.scatter(myPCA[textClass==classNumber,0],myPCA[textClass==classNumber,1],label=year,c =col)
datapoint in zip(federalistnum, myPCA):
plt.annotate(str(number),xy=datapoint)
plt.legend() plt.show()
```

OUTPUT :



loadings.py

```
import re, nltk, os
from pandas import DataFrame import
numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer from
sklearn.decomposition import PCA
import matplotlib.pyplot as plt
rf = open("federalist.txt","r",encoding="utf8") text =
rf.read()
rf.close()
text = text[:text.rfind("End of the Project Gutenberg")]
papers = re.split(r"FEDERALIST\.? No\.? \d+", text)
papers = papers[1:]
federalistnum = [i+1 for i in range(len(papers))]

authorship={}
for i in range(len(papers)):
    fno = i+1
    if fno==10 or fno==14 or (fno>=18 and fno<=20) or (fno>=37 and fno<=48):
        authorship[str(fno)] = "Madison"
    elif (fno>=2 and fno<=5) or fno==64:
          authorship[str(fno)] = "Jay"
    elif (fno>=49 and fno<=58) or fno==62 or fno==63:
        authorship[str(fno)] = "Unknown"
    else:
        authorship[str(fno)] = "Hamilton"
authorColor = {"Hamilton":"magenta","Madison":"green","Jay":"brown","Unknown":"black"}
countVectorizer = TfidfVectorizer(max_features=1000, use_idf=False)
countMatrix = countVectorizer.fit_transform(papers)
countMatrix= countMatrix.toarray()
```

```
pca = PCA(n_components=2)
myPCA = pca.fit_transform(countMatrix)
uniqueAuthors = ["Hamilton", "Madison", "Jay", "Unknown"]
numberForClass = [i for i in range(len(uniqueAuthors))]
authForClassNumber = dict(zip(uniqueAuthors,numberForClass))
textClass = np.array([authForClassNumber[authorship[str(f)]] for f in federalistnum]) colors =
[authorColor[auth] for auth in uniqueAuthors]
# When we plot the information, let's make the dots smaller and make them # transparent
for col, classNumber, year in zip(colors, numberForClass, uniqueAuthors):
    plt.scatter(myPCA[textClass==classNumber,0],myPCA[textClass==classNumber,1]
                ,label=year,c=col, s=2,alpha=.5)
loadings = pca.components_
vocabulary = countVectorizer.get_feature_names()
for i, word in enumerate(vocabulary): plt.annotate(word,xy=(loadings[0,i],loadings[1,i]))
plt.legend() plt.show()
```

OUTPUT :