University of Lorraine - IDMC

M1 NLP

Author Identification for Philosophers

Recognizing the author of a philosophical text based on text data

Frédéric ASSMUS Cécile MACAIRE
Seweryn POLEC Alena YAKAVETS

Nancy, December 25, 2019

Table of Contents

1. Introduction

In the beginning of the year we were tasked to come up with a project that would lie within the domain of Natural Language Processing, applied to Written Corpora. Taking into account two important constraints – public availability of texts and copyright issues – we have decided to settle on a set of philosophical texts of XVII-XIX centuries and try to develop an approach that would allow detecting an author (or proximity to an author within our corpus) of a philosophical book, based on syntactical or semantic patterns in text data.

Authorship identification is the task of identifying the author of a given text from a set of suspects. The main challenge of this task is to define an appropriate way to characterize texts, capturing the writing style of authors. In the recent years authorship analysis has gained much importance due to its various applications in digital forensics, anti-terror investigation, literary works, intelligence, criminal law, civil law, plagiarism detection, *etc.*

Stylometry is the statistical analysis of literary style, that provides an alternative means of investigating works of doubtful provenance. It is based in the core assumption that authors have an unconscious aspect to their style, an aspect which cannot be consciously be manipulated, but which possesses features which are quantifiable and which may be distinctive. By measuring and counting these features, stylometrists hope to uncover the "characteristics" of an author.[1]

We have split our project into the following phases:

1. Project concept
2. Corpus creation
3. Texts processing
4. Author profiles compilation
5. Author attribution
6. Tests and analysis

In this report we will describe how we have selected and cleaned the corpus, what data we selected to base our profiling on, how we have approached the developments and what libraries we used, and also speak about our successes and challenges.

2. Corpus Creation

2.1 Defining the corpus

To create the corpus for our study we have decided to work with the Project Gutenberg. It contains over 60 000 eBooks in txt format that are in the public domain and free to use.
Official page of the Gutenberg Project: http://www.gutenberg.org/wiki/Main_Page
The Project Gutenberg License : https://www.gutenberg.org/wiki/Gutenberg:The_Project_Gutenberg_License

From the Gutenberg corpus, proceeded to extracting philosophical books to create our sub-corpus. Here are the criteria that we have used:

1. The book has to be written by a known philosopher and be of philosophical nature (we did not include, for instance, autobiographies of philosophers or historical books; we also excluded books that contained essays of multiple authors)
2. Time span for the books: 17-19 centuries
3. Originally written in English (we did not want to have the author style to be influenced by that of the translator or editor)
4. The author and the books have a dedicated page on Wikipedia (our original idea was to do some cross-checks with Wikipedia entries, for the topic of a book or topics pool per author, for example)

The total number of authors selected for our sub-corpus is 30, and the total number of books is 189. We have split out corpus into main and test with approximately the following ratio: 80/20. In table 2.1 you will find the list of authors we have used in our project, with the indication of the total number of books we considered and their respective living times.

№	Author	Number of books	Time period
1	John Stuart Mill	12	1806/1873
2	John Locke	3	1632/1704
3	Francis Bacon	6	1561/1626
4	Bertrand Russell	11	1872/1970
5	Mary Wollstonecraft	5	1759/1797
6	David Hume	8	1711/1776
7	Jeremy Bentham	1	1748/1832
8	Thomas Hobbes	1	1588/1679
9	Edmund Burke	14	1729/1797
10	Roger Bacon	1	1214/1294
11	Henry Sidgwick	1	1815/1864
12	Margaret Cavendish, Duchess of Newcastle	3	1623/1673
13	Harriet Martineau	15	1802/1876
14	John Dewey	16	1859/1962
15	Ralph Emerson	9	1803/1882
16	George Santayana	10	1863/1952
17	George Berkeley	7	1685/1753
18	Henry David Thoreau	5	1817/1862
19	William James	5	1842/1910
20	Emma Goldman	5	1869/1940
21	Herbert Spencer	9	1820/1903
22	Thomas Jefferson	9	1743/1826
23	Charles Darwin	9	1809/1882
24	Jane Addams	4	1860/1935
25	George Stuart Fullerton	3	1859/1925
26	David Starr Jordan	3	1851/1931
27	Adam Smith	2	1723/1790
28	William Kingdon Clifford	1	1845/1879
29	William De Wytt Hyde	2	1858/1917
30	William Goldwin	9	1756/1836

Table 2.1: Authors list

2.2 Cleaning the corpus

After the books selection have been done, we proceeded to cleaning the corpus. There was a lot of supplementary information in each book, unrelated to the authors texts, that needed to be removed to avoid noise in our data. Here are some examples of such auxiliary text elements: Project Gutenberg information, table of contents, foreword written by someone else (e.g. the publisher), chapter indications, footnote tags, ornaments.

2.2.1 Automated cleaning

Some of the "noise" was quite regular in structure, which allowed us to remove it with the help of a script. The script looked at the combination of the following parameters and deleted this bits of text from all the books:

- Parts numbering:
  - Part name: "BOOK", "CHAPTER", "SECTION", "SECT.", "PART", "ESSAY", "CAP.", "LECTURE", "VOL.", "VOLUME"
  - Enumeration: "I", "V", "X", "L", ...; "1", "2", "3", ...; "A", "B", "C", ...

  - Syntax: PART I, PART I., PART I -, PART —, PART I–, PART I:

- Illustrations and decorators:
  - "[ILLUSTRATION . . .]"
  - "*****", ". . . . . ."

- Footnotes: "FOOTNOTES:", "[Footnote: text text]", "[1-9*]" Text text text

- Multiple spaces and multiple carriage returns

2.2.2 Manual cleaning

Some other "noise" in the books was less predictable or required pragmatic judgement, without a possibility to write sufficient yet simple instructions to delete it. Therefore it was more efficient to perform some manual cleaning in the books.

Manually cleaned items included the following:

- Foreword written not by the author
- Project Gutenberg references
- Publisher info
- Footnotes and comments written not by the author
- Table of contents
- Other irregularities

These both types of cleaning allowed us to remove all the unnecessary from all the books, leaving only relevant text for further stylistic analysis.

3. Linguistic features and data extraction

3.1 Linguistic features

We are more original than we think – this is what is being suggested by literary text analysis using stylometry methods. Even to convey a similar idea, each of us uses the same language in a slightly different way. Some have a broader vocabulary, others narrower, some like to use certain phrases and make mistakes, others avoid repetition and are linguistic purists. And when we write, we also differ in the way we use punctuation. Texts written by well-known authors is being studied, and if a text with a questioned authorship arrives, you can try to find the creator based on the parameters, closest to those obtained for the material being identified.[2]

To define the style, and subsequently the profile, of a philosopher, we have looked at the following statistical data in the books:

- Text length in sentences,
- Vocabulary size,
- Lexical richness,
- Punctuation signs number,
- Punctuation within a sentence,
- Average number of words per sentence,
- Average number of words (without punctuation signs) per sentence,
- Length of words,
- POS frequency: percentage of each POS tag,
- Modal verbs frequency,
- "Ego" indicator: frequency of pronouns I, me, myself, mine, my,
- Logical connectors frequency, by type: emphasis ("especially", "also", "indeed", ...), comparison ("as", "equally", "similarly", ...) , contrast ("but", "however", ...), addition ("further", "as well as", ...), illustration ("such as", "as an example", ...),
- Punctuation types frequency : comma, dot, semicolon, exclamation mark, question mark, dash, slash and colon,
- Most used words (excluding stopwords),
- Topic of a book.

Most of the items we considered, turned out to be somewhat classical markers of author attribution. We would like however make a callout about the usage of logical connectors (expressions like: hence, therefore, follows from, in the first place, on the other hand, at this point, *etc.*). When brainstorming about how we can approach the project, what features to keep and what to set aside, we thought about the specifics of our corpus and how can we make our solution leverage its peculiarities. We therefore checked some guidelines for writing texts of philosophical nature. One of the key ideas, that seemed particularly relevant for our project, was that such texts should be carefully organised: you should plan the argumentation flow, there should be a logical progression of ideas, logical connective expressions should be mindfully used when asserting your position in the sequence of points.[3] [4] You have to defend the claims you make, you have to offer reasons to believe them, you should use connective words to help your reader keep track of where your discussion is going.[5] Considering these guidelines, we have decided to add logical connectors to the pool of possible features of an author profile.

3.2 Texts processing

In this project, we have used NLTK library, Gensim library to help us extract the outlined text features, and Scikit-learn, Pandas and Matplotlib Libraries to manage the representation of statistical data obtained with principal component analysis. We will describe what libraries we used for what purpose and some behavior we observed.

3.2.1 NLTK

This library provides several tools which will be useful for the annotation of the corpus.

Indeed, we could:

- know the occurrence of a specific word,
- count vocabulary (text length, size of the vocabulary, measure of the lexical richness, ...),
- do simple statistics (frequency of each vocabulary item, average word length, ...),
- identify collocations (sequence of words that occur together unusually often),
- categorize and automatically tag words (various methods are provided in NLTK),
- analyze sentence structure (ambiguities, ...)

3.2.2 Part of speech tagging

For Part of speech (POS) tagging we used a word tokenizer from the python NLTK library. We loaded in, given the specified pathway, non tokenized books that were "cleaned" for further processing. For each file we removed whitespaces from each line, in this manner we were able to obtain individual words. The processed lines were then stored in a separate array.

After that, we proceeded to tokenize each word by extracting each sentence from the aforementioned array. This allowed the tokenizer to "decide" on the POS tag with context, making our results more accurate.

Finally, once we had the complete tokenized file, we saved it under a new name marking it as POS tagged. This processing allowed us later to do statistical analysis in the frequency of POS tags in our corpus. The script is visible in the appendix, section A.1.1.

3.2.3 Topic modelling

The topic modeling (see section A.1.2) was performed thanks to the Gensim Library. Indeed, it is a powerful machine learning tool which provide different approaches to topic modeling. Specifically to this project, we used LDA model (Latent Dirichlet Allocation). The LDA algorithm takes in entry, a corpus created with a sample of tokenized texts, and then creates the model with a specific number of topics.

To know the number of topics which represents the best the corpus, we computed the coherence value, expressed in percentage. At the end, the algorithm displays a list of words per topics. Assigned to each, a percentage. The higher the percentage associated to a word, the better it represents the topic.

We have experimented with various numbers of keyword topics: 10, 15, 20. With the corpus we had, the best number of topics was 17. The image 3.1 bellow shows you the list of topics, represented by keywords. The best Coherence value for this LDA model was 62 %.

```
Topic 0 : 0.010*"reasoning" + 0.009*"perception" + 0.008*"substance" + 0.007*"species" + 0.006*"derive" + 0.006*"conne
+ 0.005*"essence"
Topic 1 : 0.008*"doth" + 0.008*"infinite" + 0.006*"sensitive" + 0.005*"christ" + 0.005*"unto" + 0.005*"church" + 0.005
Topic 2 : 0.027*"species" + 0.016*"gland" + 0.015*"flower" + 0.012*"seed" + 0.009*"solution" + 0.008*"page" + 0.007*"c
Topic 3 : 0.005*"religious" + 0.005*"american" + 0.005*"lady" + 0.004*"slave" + 0.004*"gentleman" + 0.003*"america" +
0.002*"traveller"
Topic 4 : 0.020*"hastings" + 0.016*"company" + 0.010*"court" + 0.009*"lord" + 0.009*"government" + 0.008*"lordship" +
Topic 5 : 0.010*"government" + 0.005*"king" + 0.005*"constitution" + 0.004*"france" + 0.004*"sentiment" + 0.003*"parli
0.002*"kingdom"
Topic 6 : 0.011*"trade" + 0.011*"price" + 0.009*"labour" + 0.008*"money" + 0.008*"government" + 0.007*"revenue" + 0.00
Topic 7 : 0.006*"mile" + 0.006*"indian" + 0.006*"river" + 0.005*"night" + 0.005*"shore" + 0.004*"woods" + 0.004*"islan
Topic 8 : 0.013*"girl" + 0.007*"emerson" + 0.006*"carlyle" + 0.005*"social" + 0.005*"letter" + 0.005*"week" + 0.004*"c
Topic 9 : 0.007*"proposition" + 0.007*"phenomenon" + 0.006*"process" + 0.006*"theory" + 0.005*"physical" + 0.005*"conc
0.004*"philosophy" + 0.004*"mental"
Topic 10 : 0.016*"consciousness" + 0.012*"sensation" + 0.011*"sameness" + 0.011*"external" + 0.010*"substance" + 0.008
0.006*"infinite"
Topic 11 : 0.010*"social" + 0.007*"ideal" + 0.005*"happiness" + 0.004*"philosophy" + 0.004*"impulse" + 0.003*"theory"
0.003*"practical"
Topic 12 : 0.005*"lady" + 0.005*"king" + 0.004*"night" + 0.004*"father" + 0.003*"morning" + 0.003*"white" + 0.003*"oli
Topic 13 : 0.008*"social" + 0.007*"russia" + 0.006*"revolution" + 0.005*"government" + 0.004*"political" + 0.004*"ecor
0.003*"america"
Topic 14 : 0.007*"mother" + 0.005*"grey" + 0.005*"sister" + 0.004*"husband" + 0.004*"father" + 0.004*"hugh" + 0.004*"l
Topic 15 : 0.018*"capital" + 0.018*"labor" + 0.015*"production" + 0.015*"price" + 0.015*"money" + 0.013*"wages" + 0.01
Topic 16 : 0.033*"male" + 0.025*"female" + 0.016*"species" + 0.013*"sex" + 0.012*"bird" + 0.010*"colour" + 0.009*"sexu
```

Figure 3.1: Topics list with LDA model

The coherence value of 62% means that 62% of the corpus is well represented by these topics. This value can be improved by changing some parameters to build the LDA model, such as the number of topics or the number of passes (set by default at 10).
Moreover, even if it is a good library to help define a list of topics, the list of keywords provided per topic did not necessary help us in defining the final topic. As you can see in the figure, the topic n°1 is defined with the words: "doth", "infinite", "sensitive", "christ", ... – these words do not belong to the same lexical family.

3.2.4 Mega script

The "mega script", as we call it, consists of two separate programs: the first extracts the author names and the book titles, and the second loads authors specified by user and runs tests on them.

The first program is called "Produce_Author_File", (see section A.1.3). It works by extracting the author's name along with the book title from the specified path, it then takes this information and generates a list of author then below all the books of that author in an alphabetic order. For each author at the beginning of the entry there is a special character assigned to separate the author name from the actual book title. Finally a separate file is generated called "Profile List" that stores the entire array contents in alphabetical order.

The second program is called "Test Books" (see section A.1.4). The user selects which authors to test on, the program then loads in the file "Profile List" to extract the selected author names and the books assigned to them. Then the specific path is reconstructed which allows for loading of all the contents of all selected author's books into an array; that array then gets passed to the statistical tests mentioned at the beginning of chapter 3 such as part of speech frequency or modal verb frequency to generate a score.

At the end the results are collected and passed onto a create-csv procedure that generates a csv file with the final results for each author.

Be aware that this procedure needs to be run two times: the first time to generate the csv file (see section A.2) for the train data and the second one to generate a csv file for the text you want to test and which has an unknown author. Remember that the aim of this project is to identify to which author a text belongs (or is closest to).

3.2.5 Principal Component Analysis (PCA)

Principal component analysis is a useful tool for data visualization. It converts a set of features of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. To visualize the data stored in the csv file, previously generated by the mega script, Sci-kit learn and Matplotlib were used. After centering the data (with fit-transform method of sci-kit learn, which makes it have zero mean and unit standard error), Matplotlib is a library which displays the data in a plot. You will see in the figures 3.2, 3.3 and 3.4 bellow that each point in the plot corresponds to a specific author.

The first script called "PCA_authors.py" (see section A.1.5) shows the plot of the train data.

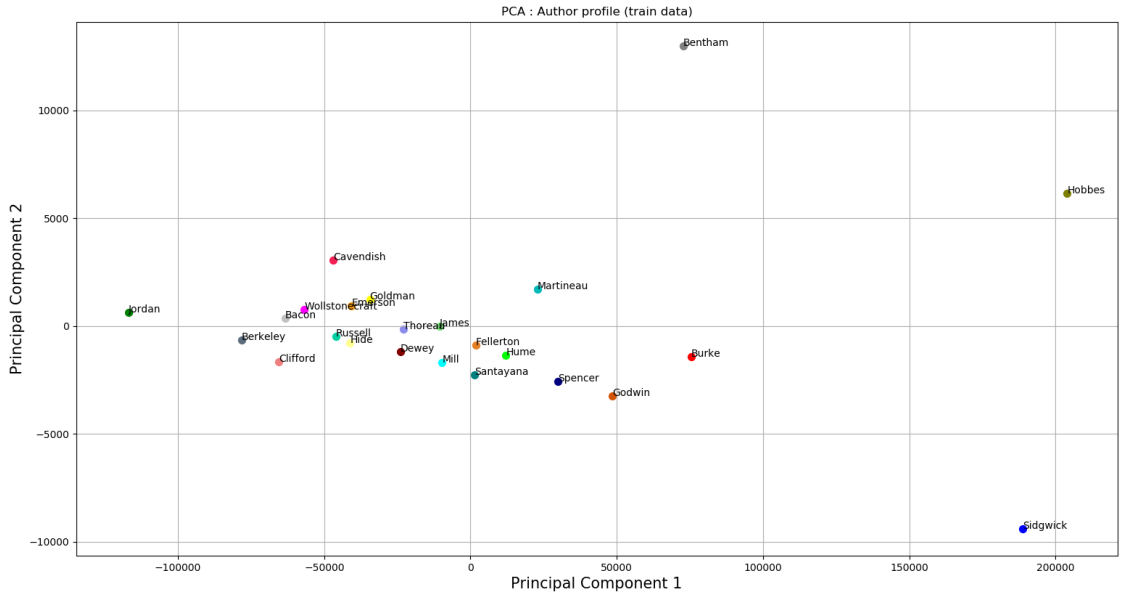

Figure 3.2: PCA plot of the train data (author profile)

Figure 3.2 identifies which authors have similarities in terms of statistic markers. Three authors are quite different from the others: Hobbes, Sidgwick and Bentham. At this point it is hard to say, whether this is due to their very distinct style or because they had very small books representation in our corpus. With the features we set in the previous steps, it is more difficult to distinguish between Goldman (anarchist political philosopher) and Emerson (transcendentalist movement), for example. We will explain in the conclusion part how could we improve this visualization.

The second script "PCA_new_text.py" (see section A.1.6) creates the PCA model and then shows the results on a plot for a single text you want to test in relation to the train data. In fact, this script uses the csv file from the train data and the second csv file which contains the data of the tested text.

Here is an example of a test text, as shown in figure 3.3:

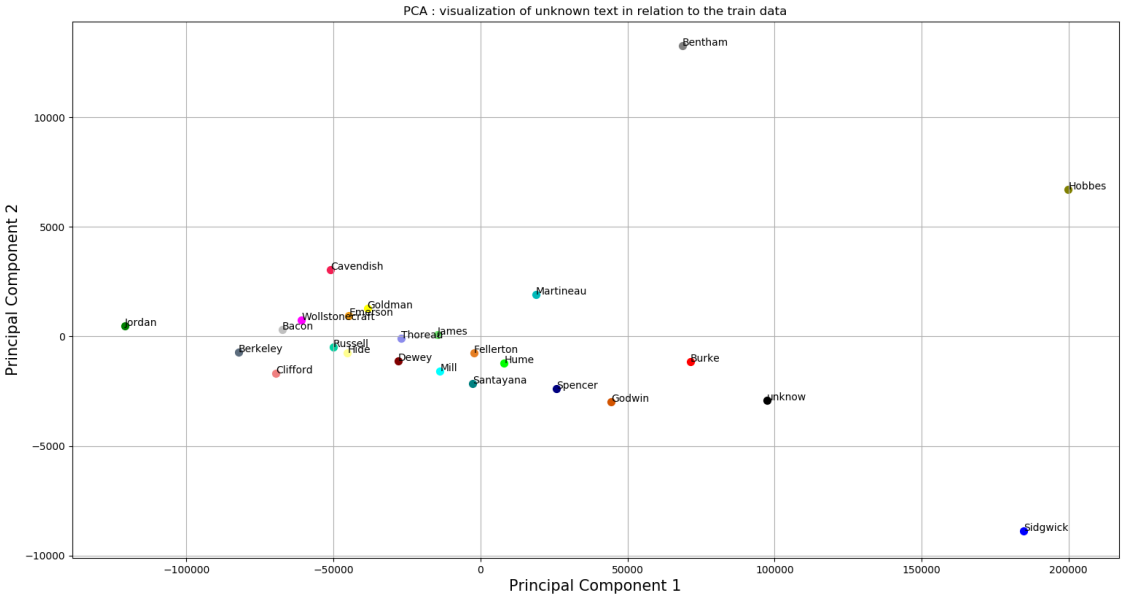

Figure 3.3: PCA plot of a test text with a new author profile

This text does not present similarities with an author from the train data. We can deduce that this text is from a new author. Indeed, this text was written by John Locke.

Another test, but with a text from an author present in the train data. Let us see in figure 3.4 if the model correctly identified the author.

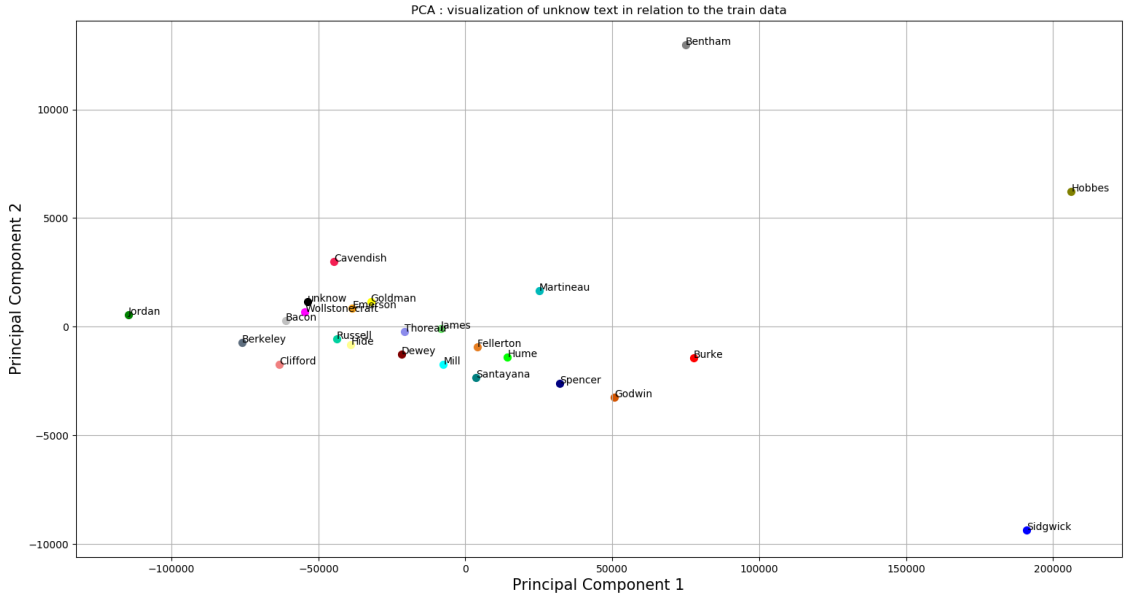

Figure 3.4: PCA plot of a test text with an existing author profile

As you can see above, the tested text (labeled "unknow" in the plot) is really close to author Wollstonecraft. This test is correct. The text we choose was written by this specific author.

4. Conclusion

This project was a great opportunity for all to try our hand in Natural Language Processing, to explore and understand specific libraries involved in NLP applications, and how linguistic data can be applied in digital solutions.

4.0.1 What challenges we had

Our first difficulty was to well identify what we wanted to do and stay focused on one main objective to avoid scope creep. After defining and fine-tuning the scope, we had to explore available libraries and decide what were the best methods to apply to achieve good results in author identification, based on the parameters that we selected.

During the corpus cleaning, even if the script removed automatically a lot of "noise", sometimes it would not capture all the things identified in the algorithm, so we had to do an additional post-cleaning check. For example, it would bug when it encountered section names starting with "=" or special characters. There is a chance that the corpus is not perfectly clean, but the "noise" part should be minimal.

One other thing we were not quite sure about, was how the system would tokenize mathematical formulas, that were present in at least one of the books. One the other hand, we considered that the actual correctness of POS tags would be less important for the final authors comparison, as long as the system would approach the formulas in a unified way.

Originally we planned to use a different source for our corpus: Corpus of English Philosophy Texts (CEPhiT), link to CEPhiT. It looked very promising, as it already had a pre-selection of philosophical books, it also had a metadata file containing information about the text sampled and its author's sociolinguistic background. However it required special software to open it, because of unusual extension of the files, so we had to drop this idea and switch to Project Gutenberg instead.

4.0.2 What went great

Even if some improvement can be done (see next section), we managed to successfully:

- create a corpus thanks to Gutenberg Project
- clean the corpus both manually and automatically with a script
- get acquainted with libraries useful for our project
- identify the interesting operations to identify a philosophical book
- compute statistical data on the corpus
- identify topics for our corpus and attribute a topic per book
- come up with an algorithm to create a profile per author (with statistical methods)
- visualize data which represent an author profile thanks to a PCA model with 2 principal components
- visualize a text which not belong to the train texts on the PCA model

It has been a great experience, with lots of things learned and some good (amazing!) team work, and it is very rewarding to see that the algorithm works.

4.0.3 Possible improvements

First of all, to create and display the data of each author for the PCA, the mega script takes in entry, a sample of text per author and then computes the mean of each statistic features of a group of texts that belong to the same author to create, at the end, his profile. The mean is not necessary the best operation. In fact, an author may have written texts that are diametrically opposed in terms of construction, punctuation elements, *etc.* It could be a better solution to not have the mean but a scale.

Secondly, PCA model has some limitations. PCA is not scale invariant. We need to scale our data first. PCA is only based on the mean vector and covariance matrix. Some distributions (multivariate normal) are characterized by this, but some are not. If the variables are correlated, PCA can achieve dimension reduction. If not, PCA just orders them according to their variances.

Moreover, we only include statistical data to create the authors' profile. Topic modeling algorithm does not create statistic data but just returns the ID of the best topic that represent a text. It could be interesting to find another data visualization tool to take into account this other type of data.

As for logical connectors, we are currently only checking the frequency of expressions from a limited list. It is possible to improve this part by either expanding the list to account for more expressions or look into discourse parsing to try to identify and catch all logical connectors used in the corpus by our philosophers.

4.0.4 What stayed out of scope

There are certain ideas that we considered using, but that stayed out of scope for this project. Some things were clearly "nice-to-haves", so we de-prioritized the features that were not directly related to the core objective of the project. Below you will find some of them:

1. Cross-checking results with Wikipedia. Originally, one of our ideas was to cross-check the data we receive through our analysis with that available on Wikipedia for a given author and book. That could be helpful to enrich the topics list an author would be famous for or ideas represented in a particular book, or test the pertinence of our results. It could also allow us to extract some meta information about an author – from the dedicated section on the right side of the page.

2. UI/Visual representation. Ideally it would be nice to have a dedicated app or webpage with a nice front-end interface to facilitate the interaction with the program as opposed to dealing with the raw code.

3. Interactive wordclouds. One of the visually appealing items could be wordclouds. By default they would show the most frequently used words. You could drill down a selected word, and see all its occurrences in the given text. It could also be possible to apply a custom or pre-set filter (e.g. ego, by POS, bi-grams) to fish for more precise sentences.

4. Filtering authors or books by theme. It could be possible for a user to select a topic from the ones that exist in the corpus, and the program would show you, which authors used it in what books.

5. Interactive comparison criteria. A possibility to enable/disable certain statistical components on the fly and re-build the comparison data.

6. Usage of rare words. This is kind of embedded in the lexical richness and vocabulary size components. We did not manage to find an easy way to identify the words that would be considered as rare back in the 17-19 centuries.

7. Number of paragraphs per text. After certain consideration, we decided not to proceed with this data, as it did not seem the most relevant component.

Bibliography

[1] David I Holmes and Judit Kardos. Who was the author? an introduction to stylometry. *Chance*, 16(2):5–8, 2003. 1

[2] Stanisław Drożdż. Texts like networks: How many words are sufficient to recognize the author? https://press.ifj.edu.pl/en/news/2019/04/11/, 2019. Accessed: 2019-12-20. 5

[3] Peter Horban. Writing a philosophy paper. http://www.sfu.ca/philosophy/resources/writing.html, 1993. Accessed: 2019-10-15. 5

[4] Michael O'Connor. Writing a philosophy essay. https://advice.writing.utoronto.ca/types-of-writing/philosophy/, 1993. Accessed: 2019-12-15. 5

[5] James Pryor. Guidelines on writing a philosophy paper. *Creative Commons*, 6:09–12, 2001. 5

A. Appendix

A.1 Python Codes

A.1.1 Part-of-Speech Tagging

```
POS_Tag_Script.py       Mon Nov 25 15:04:47 2019       1
from nltk import sent_tokenize, word_tokenize, pos_tag
import os
def main():

    path = 'C:\\Users\\Utilisateur\\Documents\\NLP\\NLP Project\\ProcessedAuthors'
    name = 'POStagged_'
    for f in os.walk(path):
        if __name__ == "__main__":
            for filename in os.listdir(path):
                with open('C:\\Users\\Utilisateur\\Documents\\NLP\\NLP Project\\ProcessedAu
thors\\' + filename , "r", encoding="utf-8") as file:
                    fl = file.readlines()
                    file.close
                    booklist = []
                    for line in fl:
                        booklist.append(line.rstrip())
                    booklist = list(filter(None, booklist))

                    fil= open('C:\\Users\\Utilisateur\\Documents\\NLP\\NLP Project\\POStagg
ed\\' + name + filename, "w+", encoding="utf-8")
                    for i in range(0, len(booklist)):
                        text=booklist[i]
                        text=[pos_tag(word_tokenize(sent)) for sent in sent_tokenize(text)]
                        fil.write(str(text))

                fil.close
    print("here")

main()
```

A.1.2 Topic Modeling

```
topicmodeling.py        Tue Dec 24 16:52:54 2019        1
import os
import nltk
from gensim.models import CoherenceModel
from nltk.collocations import *
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords


# Gensim
import gensim
import gensim.corpora as corpora
from nltk.corpus import wordnet as wn
from nltk.stem.wordnet import WordNetLemmatizer

# Stop words
stop_words = stopwords.words('english')
numbers = [str(i) for i in range(0, 1000)]
stop_words.extend(['.', '-', '--', ':', '_', 'â\200\224', ',', '&', '(', ')', '[', ']', '{'
, '}', '=', '+', '/', ';', '"\"',
                   "\\", '``', '`', '\'', "\'\'", "â\200\231", 'would', 'could', 'must', 'h
ath', 'thou', 'locke', 'emeric'])
stop_words.extend(numbers)


def get_lemma(word):
    lemma = wn.morphy(word)
    if lemma is None:
        return word
    else:
        return lemma


def get_lemma2(word):
    return WordNetLemmatizer().lemmatize(word)


def tokens(text):
    """tokens of a text with stopwords"""
    tokens_1 = word_tokenize(text)
    tokens_2 = [i.lower() for i in tokens_1 if i != 'I']
    # To delete uderscore inside the string
    tokens_3 = []
    for j in tokens_2:
        if j[0] == '_' and j[-1] == '_':
            j = j[1:-1]
        elif j[0] == '_':
            j = j[1:]
        elif j[-1] == '_':
            j = j[:-1]
        tokens_3.append(j)
    tokens_4 = [i for i in tokens_3 if i not in stop_words and len(i) > 3]
    tokens_final = [get_lemma(token) for token in tokens_4]
    return tokens_final

########################" GENSIM MODEL ############################

def make_data_model(file_path):
    text_data = []
    for file in os.listdir(file_path):
        with open(file_path + file, 'r') as f:
            lines = f.readlines()
            string = ''
            for i in lines:
                string += i
            t = tokens(string)
            text_data.append(t)
    return text_data
```

```
def make_lda_model(token_list, num_topics, passes=10):
    """
    Function to generate LDA models
    Args:
        token_list: a list of lists, one list for each docs
        num_topics: number of topics
        passes: passes for the LDA model
    Returns:
        dictionary
        corpus: document-term matrix, a list of vectors equal to the number of docs.
        lda_model
    """
    dictionary = corpora.Dictionary(token_list)
    dictionary.filter_extremes(no_below=15, no_above=0.8)
    dictionary.save('dictionary.gensim')

    # doc2bow() converts the dictionary into a bag of words.
    corpus = [dictionary.doc2bow(text) for text in token_list]

    lda_model = gensim.models.ldamodel.LdaModel(corpus, num_topics=num_topics,
                                                id2word=dictionary, passes=passes, per_word
_topics=True)
    lda_model.save('model' + str(num_topics) + '.gensim')  # to save te model
    return lda_model, corpus, dictionary


def model_coherence(token_list, model, dictionary):
    """Compute the Coherence complexity of the lda model"""
    coherence_model_lda = CoherenceModel(model=model, texts=token_list, dictionary=dictiona
ry, coherence='c_v')
    coherence_lda = coherence_model_lda.get_coherence()
    return coherence_lda


def model_perplexity(lda_model, corpus):
    """Compute perplexity of the lda model"""
    # a measure of how good the model is. lower the better.
    return lda_model.log_perplexity(corpus)


def lda_model_topics(lda_model):
    """Return topics identified from the lda model"""
    topics = lda_model.print_topics(num_words=10)
    return topics


def load_lda_model(dict_path, model, token_list):
    """Load Lda model"""
    dictionary = corpora.Dictionary.load(dict_path)
    corpus = [dictionary.doc2bow(text) for text in token_list]
    lda_model = gensim.models.ldamodel.LdaModel.load(model)
    return lda_model, corpus, dictionary


def topic_doc(doc, lda_model, dictionary):
    """find the topic of a text"""
    doc_l = open(doc, 'r')
    lines = doc_l.readlines()
    string = ''
    for i in lines:
        string += i
    current_doc = tokens(string)
    other_corpus = [dictionary.doc2bow(current_doc)]
    unseen_doc = other_corpus[0]
    vector = list(lda_model[unseen_doc])[0]
    best_topic = max(vector, key=lambda item: item[1])[0]
    print(best_topic)
```

```
# path to the texts
path_text = '/home/macaire/Bureau/nlpproject/processed_with_script/'
# path to the dictionary saved
path_model = '/home/macaire/Bureau/nlpproject/topic_modeling/'

if __name__ == "__main__":
    text_data = make_data_model(path_text)
    lda, corpus, dictionary = load_lda_model(path_model+'dictionary.gensim', path_model+'mo
del17.gensim', text_data)
    print('Topics : ', lda_model_topics(lda))
    for filename in os.listdir(path_text):
        topic_doc(path_text+filename, lda, dictionary)
```

A.1.3 Produce Author File

```
Produce_Author_File.py       Fri Dec 20 14:18:30 2019       1
import os            #This program generates a file which alphabetically sorts the authors
and all their books, run this
import csv           #before testing the corpora. WARNING, this will generate the file in t
he same folder as program


path = '/home/macaire/Bureau/test/' #ONLY thing for you to do is specify the path where the
 books you want to use are

def  loadbooks(path,completlist,lister):       #Loads all the books from a specified path a
nd extracts name and author
    keeper=''
    templist=[]

    for f in os.walk(path):
          if __name__ == "__main__":
                for name in os.listdir(path):
                    templist.append(name)
    templist.sort()

    for name in templist:
        sameauth=False
        x = name.rfind("_") #Find the _that splits the author and title
        y = name.find("_")
        author=(name[y+1:x])
        if author==keeper:
            sameauth=True

        keeper=author
        z=(len(name))
        booktitle=(name[x+1:z])
        if sameauth==False:
            lister.append(author)
            completlist.append(author)

        lister.append(booktitle)

def assignwrite(completlist,lister):          #Finds each author and marks them, then saves
 them to a file
    for x in range(0,len(completlist)):        #author list
        for val1 in range(0,len(lister)):       #complete list
            if completlist[x]==lister[val1]:
                lister[val1]='$A' + lister[val1]

    with open('Profile list', 'w') as f:
        for x in range(0,len(lister)):
            f.write(lister[x]+'\n')



def Makefile(path):              #Extract the names and based on them make a file
    completlist=[]
    lister=[]
    loadbooks(path,completlist,lister)
    completlist = list(dict.fromkeys(completlist))     #clear duplicates
    assignwrite(completlist,lister)

Makefile(path)
```

A.1.4 Mega script

```
Test_Books.py       Mon Dec 23 20:40:49 2019       1
import nltk
from nltk.tokenize import word_tokenize
import csv
import string
from nltk.tokenize import RegexpTokenizer


def search(searcharray, inputstring):  # Finds the author and all their books
    flag = False
    filname = 'Profile list'
    with open(filname) as file:
        fl = file.readlines()  # read all file contents into fl
        file.close

    for x in fl:  # for each entry in the list
        temp = x
        if temp[0] == '$' and temp[1] == "A":
            flag = False
            temp = temp[2:-1]  # Cut the sign
            for y in inputstring:  # looking for all entries with the same author
                if temp == y:  # if the selected author is the one sought after
                    flag = True
                    name = temp
        if flag == True and temp != y:
            booknamearr.append(temp)  # Assign the book name
            temp = name + '_' + temp
            autharray.append(name)  # assign the authorname
            searcharray.append(temp)  # Assigns the full path
    return searcharray


def loadina(path, firstpart, find):  # This one loads in the selected books by author
    filist = []
    internalpath = path + '/'
    count = 0
    for y in find:
        new = firstpart + y
        new = new.replace('\n', '')
        new = internalpath + new
        print(new)
        try:
            with open(new) as file:  # Opens the filename
                fl = file.readlines()
                filist.append(fl)
                fl = ''
                file.close
            pass
        except:  # If it doesn't exist because e.g: $ is used then it skips that
            print(new, 'Does not exist in the filepath!')
            autharray.pop(count)  # likewise removes the same entry from the author and boo
k title arrays
            booknamearr.pop(count)
            continue  # carry on to the next entry
        count += 1

    return filist


def number_of_sentences(doc):  # includes punctuation
    resultlist = []
    for x in doc:
        resultlist.append(len(x))
    return resultlist


def calculatemean(autharray, resultlist):  # My test program calculates the mean, you can p
aste your progs here
    count = 1
```

```
    mean = 0
    interarray = []
    takearray = autharray
    takearray.append('stopper')  # For mine to work and process the last value needed to be
 appended
    for x in range(0, len(takearray) - 1):
        if takearray[x] == takearray[x + 1]:
            mean = mean + resultlist[x]
            count += 1
        else:
            mean = mean + resultlist[x]
            mean = mean / count
            interarray.append(autharray[x])
            interarray.append(mean)
            mean = 0
            count = 1
    autharray.remove(autharray[-1])
    return interarray


def concat_results(autharray, resultlist):
    finalarray = []
    for i in range(len(resultlist)):
        finalarray.append(autharray[i])
        finalarray.append(resultlist[i])
    return finalarray


autharray = []  # Takes the name of each author
booknamearr = []  # Takes the name of the book


def counting_vocabulary(doc):
    """Counting vocabulary of a text"""
    resultlist = []
    for x in doc:
        string = ''
        for i in x:
            string += i
        tokens_text = word_tokenize(string)
        resultlist.append(len(tokens_text))
    return resultlist


def lexical_richness(doc):
    """Counting lexical richness"""
    resultlist = []
    for x in doc:
        string = ''
        for i in x:
            string += i
        tokens_text = word_tokenize(string)
        vocabulary = len(set(tokens_text))
        resultlist.append((vocabulary / len(tokens_text)) * 100)
    return resultlist


def number_of_punctuation(doc):  # number of punctuation signs in a list of books
    resultlist = []
    for x in doc:
        text = ''
        for i in x:
            text += i
        punctuation = string.punctuation
        n = 0
        for l in text:
            if l in punctuation:
                n += 1
```

```
        resultlist.append(n)
    return resultlist


def number_of_words_np(doc):  # no punctuation
    resultlist = []
    for x in doc:
        string = ''
        for i in x:
            string += i
        tokenizer = RegexpTokenizer(r'\w+')
        num = len(tokenizer.tokenize(string))
        resultlist.append(num)
    return resultlist


def words_per_sentence(doc):  # average number of words per sentence (includes puncutation)
    l1 = counting_vocabulary(doc)
    l2 = number_of_sentences(doc)
    num = [x / y for x, y in zip(l1, l2)]
    return num


def words_per_sentence_np(doc):  # average number of words per sentence (no punctuation)
    l1 = number_of_words_np(doc)
    l2 = number_of_sentences(doc)
    num = [x / y for x, y in zip(l1, l2)]
    return num


def punctuation_per_sentence(doc):  # number of punct per sentences
    l1 = number_of_punctuation(doc)
    l2 = number_of_sentences(doc)
    num = [x / y for x, y in zip(l1, l2)]
    return num


def number_of_letters(doc):  # includes punctuation
    resultlist = []
    for x in doc:
        string = ''
        for i in x:
            string += i
        num_l = len(string)
        resultlist.append(num_l)
    return resultlist


def number_of_letters_np(doc):  # no punctuation
    l1 = number_of_letters(doc)
    l2 = number_of_punctuation(doc)
    num = [x - y for x, y in zip(l1, l2)]
    return num


def average_word_length_np(doc):  # average length words per doc
    l1 = number_of_letters_np(doc)
    l2 = number_of_words_np(doc)
    num = [x / y for x, y in zip(l1, l2)]
    return num


def egocentricity_level(doc):
    """Return frequency value of egocentric words for a list of books"""
    ego_list = ["me", "Me", "myself", "Myself", "I", "mine", "Mine", "my", "My"]
    result_l = []
    l2 = number_of_words_np(doc)
    for x in doc:
```

```
        n = 0
        string = ''
        for i in x:
            string += i
        for w in word_tokenize(string):
            for item in ego_list:
                if w == item:
                    n += 1
        result_l.append(n)
    num = [x / y for x, y in zip(result_l, l2)]
    return num


def modal_verbs_frequency(doc):
    """Return frequency of modal verbs for a list of books"""
    modal_list = ["can", "could", "could have", "must", "need", "must have", "may", "might"
, "would", "would have",
                  "shall", "need", "have to", "ought to", "dare", "should", "should have",
"will be able to", "forced",
                  "will have to", "allowed", "Can", "Could", "Could have", "Must", "Need",
"Must have", "May", "Might",
                  "Would", "Would have", "Shall", "Need", "Have to", "Ought to", "Dare", "S
hould", "Should have",
                  "Will be able to", "Forced", "Will have to", "Allowed"]
    result_l = []
    l2 = number_of_words_np(doc)
    for x in doc:
        string = ''
        n = 0
        for i in x:
            string += i
        for w in word_tokenize(string):
            for item in modal_list:
                if w == item:
                    n += 1
        result_l.append(n)
    num = [x / y for x, y in zip(result_l, l2)]
    return num


def connector_frequency(doc):
    """ Return frequency of different type of connectors for a list of books"""
    emphasis_list = ["Especially", "Also", "In particular", "Furthermore", "In addition", "
Indeed", "Of course",
                     "Certainly", "Above all", "Specifically", "Significantly", "Notably"]
    emphasis_list2 = []
    comparison_list = ["As if", "As", "Equally", "Similarly", "In the same way", "Comparabl
e", "In like manner",
                       "Alternatively", "Unless", "Despite this", "By the way"]
    comparison_list2 = []
    contrast_list = ["But", "However", "On the other hand", "Otherwise", "Unlike", "Convers
ely", "At the same time",
                     "In spite of", "Whereas", "While", "Yet", "Apart from"]
    contrast_list2 = []
    addition_list = ["As well as", "Further", "Furthermore", "And then", "And", "Too", "Als
o", "In addition to",
                     "Not only", "Or"]
    addition_list2 = []
    illustration_list = ["Such as", "In this case", "For one thing", "For instance", "For e
xample", "In the case of",
                         "Illustrated by", "As an example", "As instance", "In other words"
]
    illustration_list2 = []
    for item in emphasis_list:
        emphasis_list2.append(item.lower())
    emphasis_list.extend(comparison_list2)
    for item in comparison_list:
        comparison_list2.append(item.lower())
```

```
    comparison_list.extend(comparison_list2)
    for item in contrast_list:
        contrast_list2.append(item.lower())
    contrast_list.extend(contrast_list2)
    for item in addition_list:
        addition_list2.append(item.lower())
    addition_list.extend(addition_list2)
    for item in illustration_list:
        illustration_list2.append(item.lower())
    illustration_list.extend(illustration_list2)
    final1 = []
    final2 = []
    final3 = []
    final4 = []
    final5 = []
    l2 = number_of_words_np(doc)
    for x in doc:
        l = []
        for i in range(5):
            l.append(l2[0])
        result_l = []
        string = ''
        emphasis = comparison = contrast = addition = illustration = 0
        for i in x:
            string += i
        for w in word_tokenize(string):
            if w in emphasis_list:
                emphasis += 1
            if w in comparison_list:
                comparison += 1
            if w in contrast_list:
                contrast += 1
            if w in addition_list:
                addition += 1
            if w in illustration_list:
                illustration += 1
        result_l.append(emphasis)
        result_l.append(comparison)
        result_l.append(contrast)
        result_l.append(addition)
        result_l.append(illustration)
        num = [x / y for x, y in zip(result_l, l)]
        final1.append(num[0])
        final2.append(num[1])
        final3.append(num[2])
        final4.append(num[3])
        final5.append(num[4])
        l2.pop(0)
    return final1, final2, final3, final4, final5


def type_of_punctuation(
        doc):
    """Return a list of frequencies of different punctuations of a list of books: comma, do
tcomma, exclamation, question mark, ..."""
    final1 = []
    final2 = []
    final3 = []
    final4 = []
    final5 = []
    final6 = []
    final7 = []
    l2 = number_of_punctuation(doc)
    for x in doc:
        comma = dotcomma = excl = qmark = dash = slash = twodot = 0
        n = []
        for i in range(7):
            n.append(l2[0])
```

```
        result_l = []
        string = ''
        for i in x:
            string += i
        for l in string:
            if l == ',':
                comma += 1
            elif l == ';':
                dotcomma += 1
            elif l == '!':
                excl += 1
            elif l == '?':
                qmark += 1
            elif l == '-':
                dash += 1
            elif l == '/':
                slash += 1
            elif l == ':':
                twodot += 1
        result_l.append(comma)
        result_l.append(dotcomma)
        result_l.append(excl)
        result_l.append(qmark)
        result_l.append(dash)
        result_l.append(slash)
        result_l.append(twodot)
        num = [x / y for x, y in zip(result_l, n)]
        final1.append(num[0])
        final2.append(num[1])
        final3.append(num[2])
        final4.append(num[3])
        final5.append(num[4])
        final6.append(num[5])
        final7.append(num[6])
        l2.pop(0)
    return final1, final2, final3, final4, final5, final6, final7


def pos_tag_frequency(doc):
    """Return frequency of each POS tag of a list of books"""
    list_pos = ["CC", "CD", "DT", "EX", "FW", "IN", "JJ", "JJR", "JJS", "LS", "MD", "NN", "
NNS", "NNP", "NNPS",
                "PDT",
                "POS", "PRP", "PRP$", "RB", "RBR", "RBS", "RP", "TO", "UH", "VB", "VBD", "V
BG", "VBN", "VBP",
                "VBZ",
                "WDT", "WP", "WP$", "WRB"]
    obj = {}
    for i in range(35):
        obj[str(i)] = []
    l2 = number_of_words_np(doc)
    for x in doc:
        l = []
        for i in range(35):
            l.append(l2[0])
        string = ''
        for i in x:
            string += i
        list_pos_numb = []
        for i in range(0, len(list_pos)):
            list_pos_numb.append(0)
        text = nltk.pos_tag(word_tokenize(string))
        for elem in text:
            for pos in list_pos:
                if elem[1] == pos:
                    list_pos_numb[list_pos.index(elem[1])] += 1
        num = [x / y for x, y in zip(list_pos_numb, l)]
        for m in range(len(num)):
```

```
            obj[str(m)].append(num[m])
    return obj


def loadinbooks():  # loads in the selected books/authors
    path = '/home/macaire/Bureau/test'  # specify the path
    inputlist = ['Addams', 'Bacon', 'Bentham', 'Berkeley', 'Burke', 'Cavendish', 'Clifford'
, 'Dewey',
                 'Emerson', 'Fellerton', 'Godwin', 'Goldman', 'Hide', 'Hobbes',
                 'Hume', 'James', 'Jordan', 'Locke', 'Martineau', 'Mill',
                 'Russell', 'Santayana', 'Sidgwick', 'Smith', 'Spencer', 'Thoreau',
                 'Wollstonecraft', 'unknow']  # You select which authors to test on here!
    find = []
    search(find, inputlist)  # Assign the filename as well as initialise author and booknam
e
    firstpart = 'processedbysentences_'  # specify what type of file do you want to test on
    bookid = loadina(path, firstpart, find)  # load the contents of the specified filename
list
    return bookid, find


bookid, find = loadinbooks()


def displayscore(resultlist, authorsummary):
    for x in range(0, len(resultlist)):
        print(autharray[x], booknamearr[x], resultlist[x])
    print(authorsummary)


def create_csv(filename, *args):
    with open(filename, 'w') as csvfile:
        filewriter = csv.writer(csvfile, delimiter=',')
        filewriter.writerow(['Authors', 'av_sentences', 'av_voc', 'lex_richness', 'av_punct
', 'av_num_word_ns',
                             'av_num_words_sentences', 'av_num_words_sentences_ns', 'av_num
_ns_sentences',
                             'av_words_length', 'av_ego', 'av_modal', 'emphasis', 'comparis
on', 'contrast', 'addition',
                             'illustration', 'comma', 'dotcomma', 'excl', 'qmark', 'dash',
'slash', 'twodot', "CC",
                             "CD", "DT", "EX", "FW", "IN", "JJ", "JJR", "JJS", "LS", "MD",
"NN", "NNS", "NNP", "NNPS",
                             "PDT",
                             "POS", "PRP", "PRP$", "RB", "RBR", "RBS", "RP", "TO", "UH", "V
B", "VBD", "VBG", "VBN",
                             "VBP", "VBZ",
                             "WDT", "WP", "WP$", "WRB"])
        i = 0
        while i < len(args[0]):
            res = []
            j = 0
            for l in args:
                if j == 0:
                    res.append(l[i])
                    res.append(l[i + 1])
                else:
                    res.append(l[i + 1])
                j += 1
            filewriter.writerow(res)
            i += 2


resultlist = number_of_sentences(bookid)  # find the total len of sentences of the books
# For your programs, here you need to populate the result list
# with your results, then print out the score by passing it further
resultlist2 = counting_vocabulary(bookid)  # find the vocabulary size per books
resultlist3 = lexical_richness(bookid)  # find the percentage of lexical richness per books
```

```
resultlist4 = number_of_punctuation(bookid)  # find the number of punctuations signs per bo
oks
resultlist5 = number_of_words_np(bookid)  # find the number of words without punctuation si
gns per books
resultlist6 = words_per_sentence(bookid)  # find the number of words per sentences per book
s
resultlist7 = words_per_sentence_np(bookid)  # find the number of words per sentences with
no punctuation per books
resultlist8 = punctuation_per_sentence(bookid)  # find the number of punctuation signs per
sentence per books
resultlist9 = average_word_length_np(bookid)  # find the words length per books
resultlist10 = egocentricity_level(bookid)  # find the frequence of egocentric words per bo
oks
resultlist11 = modal_verbs_frequency(bookid)  # find frequence of modal words per books
r1, r2, r3, r4, r5 = connector_frequency(bookid)  # find percentage of connector words per
books
p1, p2, p3, p4, p5, p6, p7 = type_of_punctuation(bookid)  # find percentage of type of punc
tuation per books
pos = pos_tag_frequency(bookid)  # find percentage of POS tag per books

# compute average between books which belongs to the same author
c1 = calculatemean(autharray, resultlist)  # average len of sentences
c2 = calculatemean(autharray, resultlist2)  # average vocabulary size
c3 = calculatemean(autharray, resultlist3)  # average percentage of lexical richness
c4 = calculatemean(autharray, resultlist4)  # average number of punctuations signs
c5 = calculatemean(autharray, resultlist5)  # average number of words without punctuation s
igns
c6 = calculatemean(autharray, resultlist6)  # average number of words per sentences
c7 = calculatemean(autharray, resultlist7)  # average number of words per sentences with no
 punctuation
c8 = calculatemean(autharray, resultlist8)  # average number of punctuation signs per sente
nce
c9 = calculatemean(autharray, resultlist9)  # average words length
c10 = calculatemean(autharray, resultlist10)  # average frequence of egocentric words
c11 = calculatemean(autharray, resultlist11)  # average frequence of modal words
con1 = calculatemean(autharray, r1)  # average frequence of emphasis connector words
con2 = calculatemean(autharray, r2)  # average frequence of comparison connector words
con3 = calculatemean(autharray, r3)  # average frequence of contrast connector words
con4 = calculatemean(autharray, r4)  # average frequence of addition connector words
con5 = calculatemean(autharray, r5)  # average frequence of illustration connector words
punct1 = calculatemean(autharray, p1)  # average frequence of comma punctuations
punct2 = calculatemean(autharray, p2)  # average frequence of dotcomma punctuations
punct3 = calculatemean(autharray, p3)  # average frequence of excl punctuations
punct4 = calculatemean(autharray, p4)  # average frequence of qmark punctuations
punct5 = calculatemean(autharray, p5)  # average frequence of dash punctuations
punct6 = calculatemean(autharray, p6)  # average frequence of slash punctuations
punct7 = calculatemean(autharray, p7)  # average frequence of twodot punctuations
pos_tag = {}
for i in range(35):
    pos_tag[str(i)] = []
for key, val in pos.items():
    pos_tag[key] = calculatemean(autharray, val)  # average frequence of each POS tags

# create csv file with all the stat values previously computed
create_csv('data_authors2.csv', c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, con1, con2, c
on3, con4, con5, punct1,
           punct2, punct3, punct4,
           punct5, punct6, punct7, pos_tag['0'], pos_tag['1'], pos_tag['2'], pos_tag['3'],
pos_tag['4'], pos_tag['5'],
           pos_tag['6'], pos_tag['7'],
           pos_tag['8'], pos_tag['9'], pos_tag['10'], pos_tag['11'], pos_tag['12'], pos_tag
['13'], pos_tag['14'],
           pos_tag['15'], pos_tag['16'], pos_tag['17'],
           pos_tag['18'], pos_tag['19'], pos_tag['20'], pos_tag['21'], pos_tag['22'], pos_t
ag['23'], pos_tag['24'],
           pos_tag['25'], pos_tag['26'], pos_tag['27'],
           pos_tag['28'], pos_tag['29'], pos_tag['30'], pos_tag['31'], pos_tag['32'], pos_t
ag['33'], pos_tag['34'])
```

A.1.5 Principle Component Analysis (PCA) for the train data

```
PCA_authors.py        Tue Dec 24 12:45:19 2019       1
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# code from https://github.com/mGalarnyk/Python_Tutorials/blob/master/Sklearn/PCA/PCA_Data_
Visualization_Iris_Dataset_Blog.ipynb

df = pd.read_csv('data_authors.csv') # read csv file

# list of features which will be used to create the PCA model
features = ['av_sentences', 'av_voc', 'lex_richness', 'av_punct', 'av_num_word_ns',
            'av_num_words_sentences', 'av_num_words_sentences_ns', 'av_num_ns_sentences',
            'av_words_length', 'av_ego', 'av_modal', 'emphasis', 'comparison', 'contrast',
'addition',
            'illustration', 'comma', 'dotcomma', 'excl', 'qmark', 'dash', 'slash', 'twodot'
, "CC", "CD", "DT", "EX",
            "FW", "IN", "JJ", "JJR", "JJS", "LS", "MD", "NN", "NNS", "NNP", "NNPS", "PDT",
            "POS", "PRP", "PRP$", "RB", "RBR", "RBS", "RP", "TO", "UH", "VB", "VBD", "VBG",
 "VBN", "VBP", "VBZ",
            "WDT", "WP", "WP$", "WRB"]

# extract features
x = df.loc[:, features].values
y = df.loc[:, ['Authors']].values

# start creating the PCA model with 2 components
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data=principalComponents
                           , columns=['principal component 1', 'principal component 2'])
finalDf = pd.concat([principalDf, df[['Authors']]], axis=1)

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlabel('Principal Component 1', fontsize=15)
ax.set_ylabel('Principal Component 2', fontsize=15)
ax.set_title('PCA : Authors profiles', fontsize=20)

# authors to take into account
targets = ['Bacon', 'Bentham', 'Berkeley', 'Burke', 'Cavendish', 'Clifford', 'Dewey',
           'Emerson', 'Fellerton', 'Godwin', 'Goldman', 'Hide', 'Hobbes',
           'Hume', 'James', 'Jordan', 'Martineau', 'Mill',
           'Russell', 'Santayana', 'Sidgwick', 'Spencer', 'Thoreau',
           'Wollstonecraft']

colors = ['#C0C0C0', '#808080', '#5D6D7E', '#FF0000', '#F41F53', '#F08080', '#800000', '#F3
9C12',
          '#E67E22', '#D35400', '#FFFF00', '#FFFE91', '#808000', '#00FF00', '#70E770',
          '#008000', '#00B9B9','#00FFFF', '#00D1A0', '#008080', '#0000FF', '#000080', '#8C8
CEC',
          '#FF00FF']

# create the graph
for target, color in zip(targets, colors):
    indicesToKeep = finalDf['Authors'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c=color
               , s=50)
    ax.annotate(target, (finalDf.loc[indicesToKeep, 'principal component 1'], finalDf.loc[i
ndicesToKeep, 'principal component 2'])) # to annotate each point with the name of the auth
or

ax.grid()
plt.show()
```

A.1.6 Principle Component Analysis (PCA) for the tested data

```
PCA_new_text.py       Tue Dec 24 15:06:58 2019       1
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# read csv files
df = pd.read_csv('data_authors.csv')
df2 = pd.read_csv('data_authors2.csv')

# list of features which will be used to create the PCA model
features = ['av_sentences', 'av_voc', 'lex_richness', 'av_punct', 'av_num_word_ns',
            'av_num_words_sentences', 'av_num_words_sentences_ns', 'av_num_ns_sentences',
            'av_words_length', 'av_ego', 'av_modal', 'emphasis', 'comparison', 'contrast',
'addition',
            'illustration', 'comma', 'dotcomma', 'excl', 'qmark', 'dash', 'slash', 'twodot'
, "CC", "CD", "DT", "EX",
            "FW", "IN", "JJ", "JJR", "JJS", "LS", "MD", "NN", "NNS", "NNP", "NNPS", "PDT",
            "POS", "PRP", "PRP$", "RB", "RBR", "RBS", "RP", "TO", "UH", "VB", "VBD", "VBG",
 "VBN", "VBP", "VBZ",
            "WDT", "WP", "WP$", "WRB"]


x = df.loc[:, features].values
x2 = df2.loc[:, features].values
final = np.concatenate((x, x2)) # add the test book values

final_auth = pd.DataFrame(np.concatenate((df[['Authors']], df2[['Authors']]))) # add the un
know author of test book
final_auth.columns = ['Authors']

pca = PCA(n_components=2)
principalComponents = pca.fit_transform(final)
principalDf = pd.DataFrame(data=principalComponents
                           , columns=['principal component 1', 'principal component 2'])
finalDf = pd.concat([principalDf, final_auth], axis=1)

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlabel('Principal Component 1', fontsize=15)
ax.set_ylabel('Principal Component 2', fontsize=15)
ax.set_title('2 Component PCA', fontsize=20)

# authors to take into account
targets = ['Bacon', 'Bentham', 'Berkeley', 'Burke', 'Cavendish', 'Clifford', 'Dewey',
           'Emerson', 'Fellerton', 'Godwin', 'Goldman', 'Hide', 'Hobbes',
           'Hume', 'James', 'Jordan', 'Martineau', 'Mill',
           'Russell', 'Santayana', 'Sidgwick', 'Spencer', 'Thoreau',
           'Wollstonecraft', 'unknow']
colors = ['#C0C0C0', '#808080', '#5D6D7E', '#FF0000', '#F41F53', '#F08080', '#800000', '#F3
9C12',
          '#E67E22', '#D35400', '#FFFF00', '#FFFE91', '#808000', '#00FF00', '#70E770',
          '#008000', '#00B9B9','#00FFFF', '#00D1A0', '#008080', '#0000FF', '#000080', '#8C8
CEC',
          '#FF00FF', '#000000']

# create the graph
for target, color in zip(targets, colors):
    indicesToKeep = finalDf['Authors'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c=color
               , s=50)
    ax.annotate(target, (
    finalDf.loc[indicesToKeep, 'principal component 1'], finalDf.loc[indicesToKeep, 'princi
pal component 2']))

ax.grid()
plt.show()
```

A.2 CSV file

Authors	av_sentences	av_voc	lex_richness	av_punct	av_num_word_ns	av_num_words_sentences
Bacon	2170.0	46782.666666666664	14.994637717319861	6469.166666666667	40593.5	22.308650524753705
Bentham	8168.0	150676.0	5.726193952587008	33340.0	125141.0	18.44711067580803
Berkeley	1748.0	35219.5	9.224330575678117	4019.0	31425.0	20.08432597823915
Burke	8221.142857142857	149857.14285714287	6.842143605418983 5	20526.64285714286	132268.92857142858	18.841148421750685
Cavendish	2138.0	59531.5	7.962114372341912	10830.5	50363.5	28.299934034208107
Clifford	2652.0	44351.0	10.139568442650674	4606.0	40215.0	16.723604826546
Dewey	5376.857142857143	75465.92857142857	11.733365113343458	9728.5	67410.64285714286	14.180055113817383
Emerson	5043.2	63789.4	14.003826232497676	9390.6	55104.0	12.543160874837014
Fellerton	7204.666666666667	94962.66666666667	6.634249916118136 5	12683.333333333334	83894.33333333333	13.586034913609039
Godwin	8082.0	129720.0	8.039623805118717	15593.0	114996.0	16.050482553822331
Goldman	6597.333333333333	67858.0	11.416674207716722	10811.0	59794.33333333333336	10.51117312789619
Hide	4446.0	62763.0	11.039742742790768	8074.0	55787.0	14.038216008049126
Hobbes	12578.0	249627.0	5.042723743825788	40250.0	211292.0	19.846318969962951
Hume	5634.5	102933.625	7.883135532962582	13265.25	90474.125	18.330363544802246
James	5926.285714285715	85546.42857142857	10.827552710378 98	12395.142857142857	75864.28571428571	14.349571050012814
Jordan	544.0	6570.0	28.203957382039572	919.0	5792.0	12.077205882352942
Martineau	9091.230769230 77	111400.076923076 92	7.798292348087295	17183.384615384617	96431.38461538461	13.237736795624311
Mill	4346.363636363636	86287.72727272728	9.98422940771533	10766.181818181818	76682.36363636363	20.38342982337896
Russell	3944.0	59102.90909090909	10.416539280109992	7857.454545454545	52584.09090909091	14.827411072126834
Santayana	5914.444444444444	94366.88888888889	11.456137089770 24	11432.111111111111	84140.77777777778	16.132921972379766
Sidgwick	10718.0	233370.0	4.020225393152504	25885.0	209278.0	21.773651800709086
Spencer	6488.5	115356.0	9.993192958050983	14537.0	103085.5	17.406892456384195
Thoreau	5406.6	76509.1	13.908973572158564	10716.2	67581.6	14.478148657159 92
Wollstonecraft	2870.4	51708.2	12.466428712548915	7559.2	44727.2	17.525588915374 33