mai 29 kl 10.33 Revisiting and replicating authorship attribution methods

May 30, 2023

1 LING4181 Supervised Reading

2 Spring 2023

3 Jihyeong Lee

4 1 June 2023

4.1 Introduction

Authorship attribution is an age old problem that relates to the study of (writing) style where style is a unique variety of language use in an individual. (Crystal, 2008) The previous sentence implies that there is a unique set of stylistic markers that an author habitually or unconsciously employs in their writing, and that style is something that can be attributed back to an individual. In real-life applications, however, it is hard to simply say that, since multiple questions are usually intertwined to form one problem: for example, how many candidates are there, how long are the documents in question, and how old are they? Two famous authorship attribution cases show the variability of questions in application: Shakespeare's authorship problem and the Unabomber case. The two cases show a similarity in that there are multiple lengthy documents to assign, but they are different in that the primary objective of the former is to verify whether the documents share the same author, or whether the proposed candidate (William Shakespeare, or someone else) is the correct author, while that for the latter is finding a certain individual(s) that is likely to have written the document, among an open pool of candidates. What features we consider more important than others can drastically change the approach to the solution, too. Are some features more reflective of the author than others? Is it realistically possible to assign a correct author to an unattributed text with them? With this essay, I aim to attempt to answer such questions, tying together the readings to get an overview of the matter at hand and reporting the procedures and results from my own experiments based on the methods mentioned in the readings. In the grand scheme of things, the semester was divided into two parts that virtually progressed at the same time: first, finding and reading previous studies, and second, replicating methods from the readings, including learning to code in Python and collecting data to test them on. There were clear limitations to this endeavor: to name a few, I could not verify the reliability of each method using valid statistical methods due to lack of time, and I did not replicate methods employing machine learning techniques. Discussions and reflections on the procedures will follow in more detail in the final section. Still, the semester as well as the replication attempts were a good opportunity to investigate a topic, which could potentially lead to more in-depth studies in the future: I plan to continue the research, as this small project left more questions than there were before. The report is structured as the following: I first piece together several previous studies related to the topic and explain central issues of authorship. Then I explain the methods I replicated in more detail with actual codes I used. Finally, I present the results and evaluate the successfulness and limitations of the project.

4.2 Background

4.2.1 Problems and paradigms in stylometry

Authorship attribution, in the sense it has in the present essay, is a quantitative approach to the study of stylistics, and the goal of authorship attribution is to assign correct author(s) to the unattributed text. In automated authorship attribution tasks, "text" refers to written documents and this does not include texts produced and conveyed using other media such as pictures, audio, or video. Stylistics could be understood as an intersection of literary analysis and linguistics, but it is widely applicable in a variety of other contexts as well, such as historical/religious studies, authorship credits/plagiarism-related controversies, political discourse analysis, and criminal investigations (including online attacks and feuds). Different situations call for different approaches, as Koppel et al. (2013) classify typical attribution problems like the following (the names in parentheses are taken from Koppel et al. (2013, p. 318)): 1. Problems where the goal is to find a most likely author among a closed set of candidate authors and an abundance of sample text is available ("simple authorship attribution"), 2. problems where the goal is to identify if two long texts are written by the same author ("long-text verification"): in other words, it is a binary question, and 3. problems where the goal is to attribute a text to one author among a large set of candidates ("many-candidates problem"). In addition to this, there are problems where there are very many candidates among which we do not even know if the true author is included, or the candidates' samples or the unattributed texts are too short for comparison. Koppel et al. poses "the fundamental problem" which is a problem of determining whether two (shorter) texts were written by the same person or two different authors. There are two main paradigms in authorship attribution that is considered, which can (and should) be modified depending on the type of task at hand: similarity-based approach on one hand and machine-learning methods on the other. A similarity-based method relies on the similarity between the query text (the anonymous text to be attributed) and a known author's style profile – all available writing by that author is considered like a single document. "Similarity" can be defined in a variety of ways. A machine-learning method use writings of a candidate author as training data to construct a classifier that categorizes anonymous documents. (Koppel et al., 2013; Koppel et al., 2012; Koppel & Winter, 2014)

4.2.2 Stylistic features

Which linguistic features represent the individual writing better than others is the central question in authorship attribution, especially in similarity-based methods. Here, I introduce two concepts proposed in previous studies with varying reliability: the methods I replicate are presented in more detail in the next section. CUSUM method (Morton, 1991; Morton and Michaelson, 1990), which is short for cumulative sum analysis, considers the occurrence of several variables within a sentence, such as the words starting with a vowel or words that are two or three letters long. The cumulative sum of to what degree these measurements as well as the sentence length deviate from the average of the whole text is calculated and compared to known author profiles that are calculated the same way. The assumption that forms the basis of this method is that language habits are constant, and the two profiles would match each other if the same author wrote both. (Coulthard et al., 2017) This method was accepted in court multiple times from 1991, before its reliability and the validity of its assumptions were refuted (Hardcastle, 1993, 1997; Totty et al., 1987). Zipf's law (1932) provide us with useful insights about the pattern of how word-types in a (natural language) text are distributed. Zipf found that when we rank all word-types in order of frequency, the frequency of the type at ith rank is inversely proportional of the rank. Put differently, the frequency of the most common word is about twice as much as the second most common word. From this, we can predict that a few word-types cover a majority of tokens in a corpus: just 150 most frequent word-types account for 50-65% of a text. (Savoy, 2020) This inspires using word frequency as an important variable in characterizing a certain individual's language habit. Various measurement methods were since proposed with relation to word frequency, each with a different view on what words are more meaningful to measure frequency of: type-token ratio is a classic index; Burrows (2002) takes the most frequent word-types, Labbé (2007) considers all word-types, and so on. Several non-linguistic aspects influence one's writing style, including gender, period in which the text is written, age, and genre. In what form these aspects are reflected in statistically measurable linguistic features, and to what extent those linguistic features are chosen consciously, and therefore to what extent these stylistic preferences reflect cognitive processes or social backgrounds, are all unclear. The endeavor to find answers to these questions might cross the boundaries of stylistics.

4.3 Methods

In this section, I explain in detail the data collection procedure and methods of analysis. As was mentioned in the previous section, there are various types of tasks in authorship attribution: simple closed-set problem, verification problems with a binary question, many-candidates problems are the common types of stylometry tasks. In real-world authorship attribution problems, however, there are also instances of co-authorship, which requires more complex methods of analysis which will not dealt with in the present paper. The methods employed in the replications illustrated in this section are applicable for closed-set, single-author problems, and the data were collected accordingly. The methods can be divided to two large sections: lexical analysis and distance-based analysis. I collected data and wrote the codes myself, but the formula for each quantitative analysis largely followed chapters 2 and 3 from Savoy(2020).

4.3.1 Data collection

Two sets of data consisted with texts from three authors each were collected. I call them `data1` and `data2` respectively. As the writing style of an individual varies across themes (what the text is about) and channels (what platform the text was intended for), I decided one channel and two themes: about gardening and true crime, on blogs run by private individuals. Sample blogs are found by several simple google search sessions using keywords "gardening" and "true crime". Three random blogs each with enough amount of text were selected. In total, texts from 3 gardening-related blogs were included in the first dataset (`data1`), and three true-crime blogs were included in the other (`data2`). Texts were manually collected going from reverse chronological order. A short excerpt from each author were stored separately as test(query) text. The query text is not included in the sample text. (In this essay, “text” refer to each sample text whose author is the same, unless otherwise noted.)

Things worth noting about texts and collecting them: 1. I do not personally know any of the blog owners that were included in the sample. (Links to the source blogs are included in the bibliography.) 2. Author C in data1 was one of three co-writers in the blog, so I filtered out the two other authors and only included one. 3. Some manual processing was involved to delete links, advertisement, mentions of the names of the blog owners.

4.3.2 Blog posts as a text

I use blog posts as sample texts for the experiment. Blogs, like other channels, show several characteristic features which should be considered when analyzed: 1. Blogs are online platforms for individuals or groups of people to share thoughts, experiences, or knowledge on various topics. Blog posts are typically reverse-chronologically organized, though there is no set rule for organizing posts. 2. There is no set rules for the type of content, formality or layout for blog posts. Therefore, blog writers have a flexibility of writing styles. Some blogs deal with a single topic seriously, while others have different purposes. 3. Many different types of media including images, audio files, and videos can be incorporated into the posts. These can be embedded inside the posts so that the readers can directly check them out without leaving the page, or provided as hyperlinks to external websites. This implies that a lot can be explained without using written words, and there can be mixed-medium contexts where the context cannot be understood entirely if only the text is considered. 4. It is a platform for relatively casual writings, and thus blog posts include words and phrases characteristic of casual online texts. At the same time, blogs are usually for long-form contents: language use characteristic of short-form text platforms such as YouTube comment sections and Twitter is rarely present in blog posts. It is difficult to generalize linguistic features specific for blog posts since there is a great variety of forms and themes across blogs, but the blogs included in the sample tended to be casual and long-form.

I presumed that inter-genre variability within a single individual would be bigger than interpersonal variability within a single genre, but blogs might be a different story. Blogs operate under self-determined rules, which might lead us to predict that there can be more room for variability between blogs than, say, tweets.

Sample texts as well as full codes can be seen on ((LINK)). Codes are in addition attached as appendix.

Table 1 Summary of sample data (texts a, b, c belong to `data1`, while d, e, f belong to `data2`.

	token	type
a	21654	3560
b	22897	3640
c	15234	2287
d	51333	4848
e	51063	6679
f	50910	7382

4.3.3 Preparation

First, necessary packages are installed:

[18]:
```
import nltk
import os
import random
import pandas as pd
import collections
import string
import numpy as np
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.probability import FreqDist

nltk.download('punkt')
print("Done!")
```

```
Done!

[nltk_data] Downloading package punkt to /home/jupyter/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

4.3.4 Preprocessing

Each text went through preprocessing as follows:

[19]:
```
def preprocess(filename):
    text = open(filename, 'r').read().replace("\n"," ").lower()
    return text.translate(str.maketrans("","", string.punctuation)).split()
```

As such, all letters were converted to lowercase letters, and was removed punctuation marks (more about this later), then split to word units instead of letter unit strings. In other words, all texts after this are a list of all the words it contains, not a string of letters, as seen below:

[21]:
```
text_a = preprocess('author_a.txt')
text_b = preprocess('author_b.txt')
```

```
text_c = preprocess('author_c.txt')
text_d = preprocess('author_d.txt')
text_e = preprocess('author_e.txt')
text_f = preprocess('author_f.txt')
print(text_a[0:50]) # processed text looks like this
```

```
['these', 'weeks', 'just', 'after', 'the', 'calendar', 'turns', 'from', 'one',
'year', 'to', 'the', 'next', 'are', 'the', 'perfect', 'time', 'to', 'think',
'about', 'your', 'goals', 'for', 'the', 'coming', 'gardening', 'season', 'on',
'this', 'week’s', 'podcast', 'i', 'discuss', 'plotting', 'out', 'plans', 'for',
'doubling', 'down', 'on', 'what', 'worked', 'well', 'in', 'the', 'garden',
'while', 'also', 'deciding', 'on']
```

4.3.5 Lexical Analysis

Lexical analysis methods refer to methods that make use of surface lexical information. They all have in common the fact that they utilize some aspect of word frequency in a given corpus(text). Several such methods have been proposed that consider different aspects of the word frequency distribution.

Lexical Diversity One way of quantitatively evaluating word choice is to measure the diversity of word use - in other words, how many types of words were used in relation to the total length (number of word-tokens) in a text.

Type-token ratio Type-token ratio is a simple index for measuring lexical diversity in a given text. A low type-token ratio indicates low degree of diversity in the choice of words the author made. It can be simply calculated as the following:

[25]:
```
def typetoken_ratio(text):
    return round(len(set(text))/len(text),4)
```

And thus Table 1 is completed as the following: **Table 2**

	token	type	ratio
a	21654	3560	0.1644
b	22897	3640	0.1590
c	15234	2287	0.1501
d	51333	4848	0.0944
e	51063	6679	0.1308
f	50910	7382	0.1450

The number of all word-types is divided by the number of all tokens to calculate type-token ratio, and more nuanced information such as the frequency of each word-type is not reflected.

Simpson's D Simpson's D (1949) is another indicator of lexical diversity. It is a useful index that is less influenced by the text length. Simpson's D measures vocabulary richness by calculating the sum of probabilities of selecting the same word twice in two separate trials. The formula is as follows (taken from Savoy(2020):

(1)

$$Simpson'sD(T) = \sum_{r=1} \frac{r}{n} \cdot \frac{r-1}{n-1} \cdot \mid Voc_r(T) \mid$$

- T refers to the text.
- n refers to the corpus size, i.e. the number of tokens in T.
- r refers to the number of times a given word type appears in T.
- VOCr(T) refers to the number of word types that appear in T exactly r times.

When there is no diversity in the usage of words, in other words there are only one word that is used throughout the entire text ($ r=n $), the formula returns 1, which is the maximum value, Hence, the closer to 0 Simpson's D value is, the richer the vocabulary use is for the given text.

The formula in **(1)** is rewritten as codes as follows:

```
[ ]: def simpson_D(text):
         count = collections.Counter(text)
         types = set(text)
         n = len(text)
         def VOC(r):
             VOC = 0
             for i in types: # i is a word(type)
                 if count.get(i) == r:
                     VOC += 1
             return VOC
         if sum(VOC(r) for r in range(1, n)) == 0:
             return 1
         else:
             return round(sum(VOC(r) * (r**2 - r) / (n**2 - n) for r in␣
       ↪range(1,n+1)),4)
```

The function `simpson_D(text)` takes a preprocessed text as its argument. `collections.Counter` function creates a dictionary type data where the key is the word type and the value is its occurence in the text. `VOC(r)` is an inner function that is defined as the number of word types(`i`) in `text` that appears r times. Since it presupposes `i` appears at least once in `text`, `VOC(r)` always appears as a positive number. Hence the absolute value sign in **(1)** is unnecessary. The function, then, calculates the sum of $\frac{r}{n} \cdot \frac{r-1}{n-1} \cdot Voc_r$ for all r.

By definition, the function should return 1 when $r = n$. However, the code above somehow always returns `0.0`. From a practical point of view, $ r=n $ is unlikely to happen, since we are dealing with a real-life language use where there are more than 1 word type in a text. But for the sake of completing the equation, the following lines were added,

```
if sum(VOC(r) for r in range(1, n)) == 0:
        return 1
```

which returns 1 if there is all `r` is zero until $r = n - 1$. Simpson's D can be used to compare the query text to candidate texts to find out which candidate text is the closest in terms of lexical diversity. Results will be discussed in section 4.

Other lexical stylometric measures

Big Words Index Big Words Index is defined as the percentage of words consisted of 6 letters or more in a given text. A text with high percentage of big words require more cognitive resources to process, effectively meaning more difficult to read. (Savoy, 2020) I used 7 letters as the threshold instead, because I was sceptical of the significance of 6 letter words, since 4-5 character long nouns can take plural form and become 6 character long. The same applies for verbs and a few other parts of speech. The function `BWI` illustrated below takes a text as its argument. The text will have been preprocessed and made into a list type data before being put. If a word in that text is 7 letters or longer, `big_word` is increased by 1, and after going through all items in the list, the function returns the percentage of 'big words'.

```
[ ]: def BWI(text):
         big_word = 0
         for word in text:
             if len(word) >= 7:
                 big_word += 1
         return round(big_word / len(text),4)
```

Mean sentence length In a similar way to big words, mean sentence length can help us understand the complexity of the text, as longer sentences are more difficult to understand than short ones. Sentence length can also reflect syntactical choices made by the writer; phrase-structuring habits like *'Tom's'* as opposed to *'of Tom'* can influence the length of the sentence. The codes below show how it is calculated:

```
[ ]: def mean_sent(filename): # put raw text, not split by space or deleted␣
      ↪punctuation marks!
         text = open(filename, 'r').read().replace("\n"," ").lower()
         t = sent_tokenize(text)
         split = []
         for sent in t:
             a = sent.translate(str.maketrans("","", string.punctuation)).split()
             split.append(a)
         return sum(len(sent) for sent in split) / len(split)
```

The function `mean_sent` takes a raw, unprocessed text as its argument and preprocesses the text inside itself. I used sentence tokenizer provided in the Natural Language ToolKit(NLTK), which turns the original text into a list of sentences, split by full stops, question marks and other common sentence-ending punctuation marks. Then, each item in the list(each sentence) is split by spaces.

At this point, the text is a multi-dimensional list where each item(sentence) is again a list of words. It is then possible to calculate how many items(words) there are inside each item(sentence), and calculating mean sentence length is as simple as dividing the sum of sentence lengths by the length of the text(number of sentences).

Mean word length To make a fairer comparison for mean word length between sample texts, I took the first 15,000 words from each text of `data1`, since the shortest text of data1 (`text_c`) has about 15,000 tokens. The sample texts in `data2` do not have to go through this process, since they all contain more than 50,000 tokens each and there are small differences between the lengths.

```
[ ]: text_a_5k = text_a[:15000]
     text_b_5k = text_b[:15000]
     text_c_5k = text_c[:15000]
```

Calculating mean word length is simpler and more precise than calculating mean sentence length:

```
[ ]: def mean_word(text):
         return sum(len(word) for word in text) / len(text)
```

Word length distribution Another way of taking word length into consideration is checking word length distribution. The function `wordlength` presented below shows how many words that are exactly `r` letters long are in a given text. It creates a list `X` of integers from 1 to the maximum word length in the text (`n`). The internal function `length` counts the number of words that consist of exactly `r` letters. Then `X`,which contains all possible word length, is paired with the the values from the internal function `length` and turned into a dictionary-type data.

```
[27]: def wordlength(text):
          dist = {}
          n = len(max(text, key=len))
          X = list(i for i in range(1,n+1))
          def length(r):
              length = 0
              for word in text:
                  if len(word) == r:
                      length += 1
              return length
          for x in X:
              dist[x] = length(x)
          return dist
```

The dictionary returned at the end, `dist`, will be turned into a dataframe later. We can then compare it to another text's word length distribution, or visualize it.

Lexical Density Lexical density (LD) is the ratio between the number of lexical items (1-functional words) and the text length. Functional words (stop words) include frequently used words that carry little meaning but grammatical information. Here, I used a predefined list of stop words provided in NLTK and selected English.

[ ]:
```
from nltk.corpus import stopwords
stopwords = set(stopwords.words('english'))

def l_density(text):
    filtered = []
    for w in text:
        if w not in stopwords:
            filtered.append(w)
    return round(len(filtered) / len(text),4)
```

The function `l_density` filters out function words from a text and computes the percentage for the remainder against the number of all tokens in the text. Therefore, the closer to 1 the value is, the 'denser' the text is. Some comparative values are necessary: I follow Savoy(p.30) where it said (and I paraphrase) that an LD value of around 0.3 for an oral production and around 0.4 and higher for writings are the norm.

4.3.6 Distance-based Analysis

Distance-based methods establish a profile for each candidate author to which we can compare the query text's profile.

Burrow's Delta (Savoy 2020: 34-36) is one of such methods: it considers 40-150 most frequent word types, and the style is reflected through the word choice. According to Savoy(34), 150 most frequent word types cover 50-65% of all tokens in a certain text, with the percentage varying depending on the theme, genre, etc. of the text.

The following is the formula for Delta (taken from Savoy(p.37)):

(2)

$$Burrow'sDelta(A_j, Q) = \frac{1}{m} \cdot \sum_{i=1}^{m} \mid Zscore(t_{i,Aj}) - Zscore(t_{i,Q}) \mid$$

- Aj is a candidate author A's profile.
- Q is the query text.
- t is a set of word-types in the MFW list.

Each t in the MFW list has the same importance, but the impact depends on their Z score values. To get the Delta value between the query text and a sample text, a list of most frequent word-types is necessary. A relative frequency value for each term can be calculated for each text: the number of occurrences for a certain word-type in a certain text is divided by the length of the text. Then the relative frequency values are compared against each other to get mean and standard deviation values. This is to get Z score for each term in each text: Z score is the relative frequency minus mean divided by standard deviation. Z score helps us understand where a certain value lies in relation to the entire sample. By comparing a Z score for a certain term in both texts, we know how much difference in using that word there is, and the bigger the sum is, the bigger the difference in word choices between the texts will be.

The function MFW below returns a list of 300 most frequent words and their frequency. The number 300 can be changed if necessary. Frequency here is absolute frequency, i.e. how many times it appears in the text. Function MFW_100 returns the percentage of MFW tokens in relation to the entire text.

```
[ ]: def MFW(text):
         freq = FreqDist(text)
         MFWlist = freq.most_common(300)
         return MFWlist

     def MFW_100(text):
         return 100 * sum(i[1] for i in MFW(text)) / len(text)
```

The codes below create a table of most frequent words (MFW) with their absolute frequency in the three respective texts. Obviously, the MFW list is different for each of the text with some overlap, and to be able to compare to each other, I took only MFWs that are present in all three lists, which makes the list shorter than the original.

```
[ ]: def abs_table(xa, xb, xc):
         dict_b = (dict(MFW(xb)))
         dict_c = (dict(MFW(xc)))
         table = pd.DataFrame(MFW(xa)).rename(columns={0: 'word', 1:'a'})
         table.set_index('word',inplace=True)
         table["b"] = ""
         table["c"] = ""
         for n in MFW(xa):
             word = n[0]
             if dict_b.get(word) != None and dict_c.get(word) != None:
                 table.loc[word,"b"] = dict_b.get(word)
                 table.loc[word,"c"] = dict_c.get(word)
             else:
                 table.loc[word,"b"] = np.nan
                 table.loc[word,"c"] = np.nan
             table.dropna(inplace= True)
         return table

     abs_table(text_a,text_b,text_c)
```

The table we get from abs_table is then turned into a relative frequency table. Relative frequency table takes each text's length (number of tokens) into consideration. Since the absolute frequency of MFW will be heavily influenced by the size of the corpus, a relative term frequency is more useful.

```
[ ]: def rel_table(xa, xb, xc):
         table = abs_table(xa, xb, xc)
         table = table.astype(float)
         table["words"] = table.index
         table.loc[:,"a"] = round(table["a"] / len(xa),5)
```

```
    table.loc[:,"b"] = round(table["b"] / len(xb),5)
    table.loc[:,"c"] = round(table["c"] / len(xc),5)
    table.loc[:,"mean"] = table.mean(axis='columns')
    table.loc[:,"sd"] = table.std(axis='columns')
    return table

table = rel_table(text_a,text_b,text_c)
```

Lastly, the codes below calculate z-score and eventually Delta score. The query text has to be preprocessed before running these lines.

```
[ ]: def zscore_table(a,b,c):
         dict_q = dict(collections.Counter(q))
         table = abs_table(a, b, c)
         table = table.astype(float)
         table["words"] = table.index
         table["q"] = ""
         table.loc[:,"a"] = round(table["a"] / len(a),5)
         table.loc[:,"b"] = round(table["b"] / len(b),5)
         table.loc[:,"c"] = round(table["c"] / len(c),5)
         table.loc[:,"mean"] = table.mean(axis='columns')
         table.loc[:,"sd"] = table.std(axis='columns')
         for word in table["words"]:
             if dict_q.get(word) != None:
                 table.loc[word,"q"] = round((dict_q.get(word) / len(q)),5)
             else:
                 table.loc[word,"q"] = np.nan
         table.loc[:,"z_a"] = (table["a"] - table["mean"]) / table["sd"] #␣
      ↪calculates z-scores for columns a,b,c,q
         table.loc[:,"z_b"] = (table["b"] - table["mean"]) / table["sd"]
         table.loc[:,"z_c"] = (table["c"] - table["mean"]) / table["sd"]
         table.loc[:,"z_q"] = (table["q"] - table["mean"]) / table["sd"]
         table.dropna(inplace= True) # deletes rows that contain NaN
         table.drop('words', axis = 'columns',inplace= True) # deletes the redundant␣
      ↪column
         return table
```

```
[ ]: def delta(df): # calculates delta score between the column a in the given␣
      ↪dataframe and the query text
         delta_a = round(sum(list(abs(df["z_a"]-df["z_q"]))) / len(df),5)
         delta_b = round(sum(list(abs(df["z_b"]-df["z_q"]))) / len(df),5)
         delta_c = round(sum(list(abs(df["z_c"]-df["z_q"]))) / len(df),5)
         return delta_a, delta_b, delta_c, 'are Delta distance values between the␣
      ↪query text and text a, b, c, respectively.'
```

4.4 Results

In the previous section, several methods of characterizing writing styles for each author were introduced. In this section, I present results and examine the possibility of attributing correct authors to the query texts based on each index.

Type-token ratio Simpson's D Big words index Mean sentence length Mean word length Word length distribution Lexical Density Burrow's Delta

```
[ ]:
```

```
[ ]:
```

4.5 5. Discussion

Interpret the results Ways forward (What more could be done, what more am I interested in, what I will do next)

Limitations

4.5.1 Bibliography

Bird, S., Klein, E., & Loper, E. (2009). Natural language processing with python. O'Reilly Media. Burrows, J. (2002). 'Delta': a Measure of Stylistic Difference and a Guide to Likely Authorship. Lit Linguist Computing, 17(3), 267-287. https://doi.org/10.1093/llc/17.3.267 Coulthard, M., Johnson, A., & Wright, D. (2017). An introduction to forensic linguistics : language in evidence (Second edition. ed.). Routledge. Crystal, D. (2008). 'Think on my words' : exploring Shakespeare's language. Cambridge University Press. Hardcastle, R. A. (1993). Forensic linguistics: an assessment of the CUSUM method for the determination of authorship. Journal - Forensic Science Society, 33(2), 95-106. https://doi.org/10.1016/S0015-7368(93)72987-4 Hardcastle, R. A. (1997). CUSUM: a credible method for the determination of authorship? Sci Justice, 37(2), 129-138. https://doi.org/10.1016/S1355-0306(97)72158-0 Koppel, M., Schler, J., & Argamon, S. (2013). Authorship attribution: what's easy and what's hard? Journal of law and policy, 21(2), 317. Koppel, M., Schler, J., Argamon, S., & Winter, Y. (2012). The "Fundamental Problem" of Authorship Attribution. English studies, 93(3), 284-291. https://doi.org/10.1080/0013838X.2012.668794 Koppel, M., & Winter, Y. (2014). Determining if two documents are written by the same author. J Assn Inf Sci Tec, 65(1), 178-187. https://doi.org/10.1002/asi.22954 Labbé, D. (2007). Experiments on authorship attribution by intertextual distance in english. Journal of quantitative linguistics, 14(1), 33-80. https://doi.org/10.1080/09296170600850601 Morton, A. Q. (1991). Proper words on proper places. Department of Computing Science Research Report, R18, University of Glasgow. Morton, A. Q., Michaelson, S. . (1990). The Q-Sum Plot. internal report CSR-3-90, Department of Computer Science, University of Edinburgh. Savoy, J. (2020). Machine learning methods for stylometry : authorship attribution and author profiling (1st 2020. ed.). Springer. Simpson, E. H. (1949). Measurement of Diversity. Nature (London), 163(4148), 688-688. https://doi.org/10.1038/163688a0 Totty, R. N., Hardcastle, R. A., & Pearson, J. (1987). Forensic linguistics: the determination of authorship from habits of style. Journal - Forensic Science Society, 27(1), 13-28. https://doi.org/10.1016/S0015-7368(87)72702-9 Zipf, G. K. (1932). Selected studies of the principle of relative frequency in language [doi:10.4159/harvard.9780674434929]. Harvard Univ. Press. https://doi.org/10.4159/harvard.9780674434929

4.5.2 Appendix

codes

```
[ ]:
```