Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

On Authorship Attribution

Relatore:
Chiar.mo Prof.
DANILO MONTESI

Correlatore:
Dott.
FLAVIO BERTINI

Presentata da:
Gabriele Calarota

Sessione III
Anno Accademico 2019-2020

“When I was in college,
I wanted to be involved in things that would change the world”
Elon Musk

Sommario

Il compito di determinare o verificare la paternità di un testo anonimo basandosi solo su analisi del documento è molto antico, e risale almeno all'era medievale, per la quale l'attribuzione affidabile di un testo ad un'antica autorità conosciuta era essenziale per determinarne la veridicità. Più recentemente, questo problema dell'attribuzione della "paternità" di un documento ha guadagnato maggiore importanza a causa delle nuove applicazioni nell'analisi forense, nelle scienze umanistiche e nel commercio elettronico, e lo sviluppo di metodi computazionali per affrontarlo.
Nella forma più semplice, ci vengono dati esempi di scrittura di un certo numero di candidati autori e ci viene chiesto di determinare chi di loro è l'autore di un documento di autore ignoto. In questo caso, il problema dell'attribuzione dell'autore si adatta al paradigma moderno di un problema di categorizzazione del testo. I componenti dei sistemi di categorizzazione del testo sono ormai abbastanza ben compresi: I documenti sono rappresentati come vettori numerici che catturano le statistiche delle caratteristiche potenzialmente rilevanti del testo, e i metodi di apprendimento automatico sono utilizzati per trovare classificatori che separino i documenti che appartengono a classi diverse.
La maggior parte dei lavori pubblicati si concentra sull'attribuzione a serie chiusa dove si presume che l'autore del testo preso in esame sia necessariamente un membro di un insieme ben definito di autori candidati. Questa impostazione si adatta a molte applicazioni forensi in cui di solito individui specifici hanno accesso a certe risorse, hanno conoscenza di certe questioni, ecc. In questo lavoro viene mostrata una panoramica generale di ciò che è l'attribuzione d'autore al giorno d'oggi, con uno sguardo sia dal punto di vista dell'information retrieval che delle metodologie più utilizzate, prestando molta attenzione nella selezione dei dataset per non creare bias di apprendimento. Vengono confrontati dataset molto diversi tra loro sia per contenuto testuale, livello di formalità del linguaggio e lunghezza media dei documenti scritti dagli autori. Confrontando il lavoro con altri correlati, verrà mostrato come alcuni risultati ottenuti sono potenzialmente interessanti e portino a migliorare ulteriormente la confidenza del modello per questo particolare tipo di task.

Abstract

Authorship attribution is the process of identifying the author of a given text and from the machine learning perspective, it can be seen as a classification problem. In the literature, there are a lot of classification methods for which feature extraction techniques are conducted. In this thesis, we explore information retrieval techniques such as Doc2Vec and other useful feature selection and extraction techniques for a given text with different classifiers. The main purpose of this work is to lay the foundations of feature extraction techniques in authorship attribution. At the end of this work, we show how we compared our results with related works and how we managed to improve, to the best of our knowledge, the results on a particular dataset, very known in this field.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Problem Statement

The task of determining or verifying the authorship of an anonymous text based solely on internal evidence is a very old one, dating back at least to the medieval scholastics, for whom the reliable attribution of a given text to a known ancient authority was essential to determining the text's veracity. More recently, this problem of authorship attribution has gained greater prominence due to new applications in forensic analysis, humanities scholarship, and electronic commerce, and the development of computational methods for addressing the problem. In the simplest form of the problem, we are given examples of the writing of a number of candidate authors and are asked to determine which of them authored a given anonymous text. In this straightforward form, the authorship attribution problem fits the standard modern paradigm of a text categorization problem. The components of text categorization systems are by now fairly well understood: Documents are represented as numerical vectors that capture statistics of potentially relevant features of the text, and machine learning methods are used to find classifiers that separate documents that belong to different classes. A scientific approach to the authorship attribution problem was first proposed in the late 19th century in the work of Mendenhall (1887), who studied the authorship of texts attributed to Bacon, Marlowe, and Shakespeare. The key idea was that the writing of each author could be characterized by a unique curve expressing the relationship between word length and relative frequency of occurrence; these characteristic curves would thus provide a basis for author attribution of anonymous texts. This early work was put on a firmer statistical basis in the early 20th century with the search for invariant properties of textual statistics Zipf (1932). The existence of such invariants suggested the possibility that some related feature might be found that was at least invariant for any given author, though possibly varying among different authors. The majority of published works in authorship attribution focus on closed-set attribution where it is assumed that the author of the text under investigation is specifically a member of a given well-defined set of candidate authors. This setting fits many forensic applications where usually specific individuals have access to certain resources, have knowledge of certain issues, etc. A more general framework is open-set attribution [33]. A special case of the latter is authorship verification where the set of candidate authors is singleton. This is essentially a one-class classification problem since the negative class (i.e., all texts by all other authors) is huge and extremely heterogeneous. In authorship attribution it is not always realistic to assume that the texts of known authorship and the texts under investigation belong in the same genre and are in the same thematic area. In most applications, there are certain restrictions that do not allow the construction of a representative training corpus. Unlike other text categorization tasks, a recent trend in authorship attribution research is to build cross-genre and cross-topic models, meaning that the training and test corpora do not share the same properties. One crucial issue in any authorship attribution approach is to quantify the personal style of authors, a line of research also called stylometry [58]. Ideally, stylometric features should not be affected by shifts in topic or genre variations and they should only depend on personal style of the authors. However, it is not yet clear how the topic/genre factor can be separated from the personal writing style.

In this work In this work we will give a general snapshot of what authorship attribution is nowadays. We will give a key of reading from the point of view of both information retrieval and the most used methodologies, paying much attention to the selection of datasets in order not to create learning bias. In our experiment we will compare datasets that are very different from each other both in text content, formality and average length of documents written by authors. We will analyze the methods to extract the best information from the documents and how to select the classifier that best suits this type of task. We will compare the results obtained with those found in related work, taking care to recreate the same experiment performed. Finally, we will give hints for future work and a long-term vision for this branch of text categorization.

1.2 Thesis Structure

The rest of this thesis is organized into the following chapters:

- **Chapter 2**. Chapter 2 provides a more detailed introduction to this task and what are the different scenarios we may encounter when tackling the authorship attribution.
- **Chapter 3**. Chapter 3 presents a background in information retrieval, listing some

of the most common techniques used in previous authorship attribution works for extracting useful data representing the texts.

- **Chapter 4**. Chapter 4 lists a variety of related works in authorship attribution, focusing on works that used SVM as a classifier and that used either one of the Reuters Corpus dataset or The Guardian dataset to test their model.
- **Chapter 5**. Chapter 5 describes our experiment in terms of the setup, the dataset preparation, the process of features extraction and the classifier selection.
- **Chapter 6**. Chapter 6 shows the best results we obtained from the experiments we conducted on the selected datasets.

CHAPTER 2

Authorship attribution

"Science is the systematic classification of experience."
George Henry Lewes

Authorship Attribution tackles the problem of determining who, among a set of authors, wrote the document at hand. This aforementioned task has relevant applications ranging from plagiarism detection to Forensic Linguistics, such as identifying authorship of threatening emails or malicious code. Applied areas such as law and journalism can also benefit from authorship attribution, where identifying the true author of a piece of text (such as a ransom note) may help save lives or catch the offenders. In the next section we will briefly present the approach to this task in the era before machine learning. Later, we will discuss the different approaches of this particular task, whether to approach it as a profile-based or an instance-based problem, depending on which domain are we tackling: single-domain or cross-domain. In the last section, we will introduce the latest state of the art approaches for this particular classification task.

2.1 State of the Art

The first attempts to quantify the writing style go back to the 19th century, with the pioneering study of Mendenhall (1887) on the plays of Shakespeare, followed by statistical studies in the first half of the 20th century by Yule (1938, 1944) and Zipf (1932) [66] [68] [71]. Later, the detailed study by Mosteller and Wallace (1964) on the authorship of "The Federalist Papers" (a series of 146 political essays written by John Jay, Alexander Hamilton, and James Madison, 12 of which claimed by both Hamilton and Madison) was undoubtedly the most influential work in authorship attribution [58]. Their method was based on Bayesian statistical analysis of the frequencies of a small set of common words (e.g., “and,” “to,” etc.) and produced significant discrimination results between the candidate authors. Essentially, the work of Mosteller and Wallace (1964) initiated nontraditional authorship attribution studies, as opposed to traditional human-expert-based methods. Since then and until the late 1990s, research in authorship attribution was dominated by attempts to define features for quantifying writing style, a line of research known as *“stylometry”* [20]. Hence, a great variety of measures, including sentence length, word length, word frequencies, character frequencies, and vocabulary richness functions, had been proposed. Rudman (1998) estimated that nearly 1,000 different measures had been proposed by the late 1990s [51]. The authorship attribution methodologies proposed during that period were computer-assisted rather than computer-based, meaning that the aim was rarely at developing a fully automated system. In certain cases, there were methods that achieved impressive preliminary results and made many people think that the solution of this problem was too close. The most characteristic example is the CUSUM (or QSUM) technique [40] that gained publicity and was accepted in courts as expert evidence; however, the research community heavily criticized it and considered it generally unreliable [21]. Actually, the main problem of that early period was the lack of objective evaluation of the proposed methods. In most of the cases, the testing ground was literary works of unknown or disputed authorship (e.g., the Federalist Papers case), so the estimation of attribution accuracy was not even possible. The main methodological limitations of that period concerning the evaluation procedure were the following:

- The textual data were too long (usually including entire books) and probably not stylistically homogeneous.
- The number of candidate authors was too small (usually two or three).
- The evaluation corpora were not controlled for topic.
- The evaluation of the proposed methods was mainly intuitive (usually based on subjective visual inspection of scatterplots).
- The comparison of different methods was difficult due to lack of suitable benchmark data.

Since the late 1990s, things have changed in authorship attribution studies. The vast amount of electronic texts available through Internet media (e-mail messages, blogs, online forums, etc.) have increased the need for efficiently handling this information. This fact had a significant impact in scientific areas such as information retrieval, machine learning, and natural language processing (NLP). The development of these areas influenced authorship attribution technology as described:

- Information retrieval research developed efficient techniques for representing and classifying large volumes of text.
- Powerful machine learning algorithms became available to handle multidimensional and sparse data, allowing more expressive representations. Moreover, standard evaluation methodologies have been established to compare different approaches on the same benchmark data.
- NLP research developed tools able to analyze text efficiently and provided new forms of measures for representing the style (e.g., syntax-based features).

More importantly, the plethora of available electronic texts revealed the potential of authorship analysis in various applications [37] in diverse areas including intelligence (e.g., attribution of messages or proclamations to known terrorists, linking different messages by authorship) [1], criminal law (e.g., identifying writers of harassing messages, verifying the authenticity of suicide notes) and civil law (e.g., copyright disputes) [7], and computer forensics (e.g., identifying the authors of source code of malicious software) [14] in addition to the traditional application to literary research (e.g., attributing anonymous or disputed literary works to known authors) [5]. Hence, (roughly) the last decade can be viewed as a new era of authorship analysis technology, this time dominated by efforts to develop practical applications dealing with real-world texts (e.g., e-mail messages, blogs, online forum messages, source code, etc.) rather than solving disputed literary questions. Emphasis has now been given to the objective evaluation of the proposed methods as well as the comparison of different methods based on common benchmark corpora. In addition, factors playing a crucial role in the accuracy of the produced models are examined, such as the training text size [39], the number of candidate authors [31], and the distribution of training texts over the candidate authors [57].

2.2 Training's methods

In every authorship-identification problem, there is a set of candidate authors, a set of text samples of known authorship covering all the candidate authors (*training corpus*), and a set of text samples of unknown authorship (*test corpus*); each one of them should be attributed to a candidate author. In this survey, we distinguish the authorship attribution approaches according to whether they treat each training text individually or cumulatively (per author). In more detail, some approaches concatenate all the available training texts per author in one big file and extract a cumulative representation of that author's style (usually called the author's profile) from this concatenated text. That is, the differences between texts written by the same author are disregarded. Such an approach just described is called *"profile-based approach"* and early work in authorship attribution has followed this practice [46]. On the other hand, another family of approaches requires multiple training text samples per author to develop an accurate attribution model. That is, each training text is individually represented as a separate instance of authorial style. Such *"instance-based approaches*[1]*"* are described in the next section, followed by hybrid approaches attempting to combine characteristics of profile-based and instance-based methods. We then compare these two basic approaches and discuss their strengths and weaknesses across several factors.

2.2.1 Profile-based approach

One way to handle the available training texts per author is to concatenate them in one single text file. This large file is used to extract the properties of the author's style. An unseen text is, then, compared with each author file, and the most likely author is estimated based on a distance measure. It should be stressed that there is no separate representation of each text sample but only one representation of a large file per author. As a result, the differences between the training texts by the same author are disregarded. Moreover, the stylometric measures extracted from the concatenated file may be quite different in comparison to each of the original training texts.

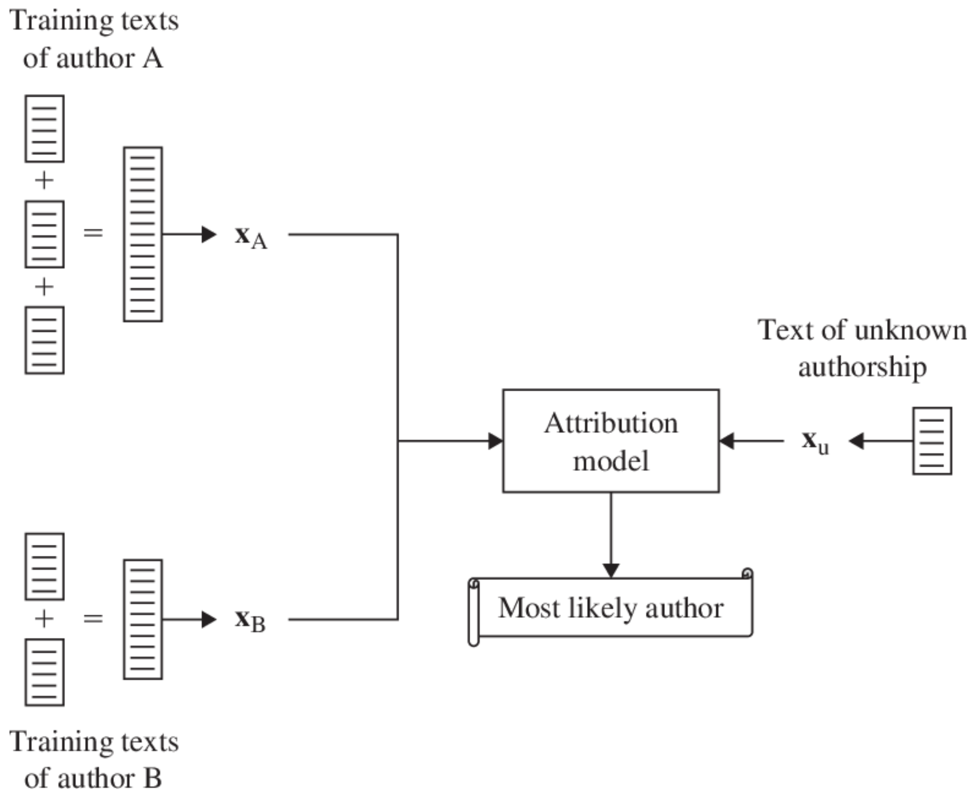
Training texts of author A
$\mathbf{x}_A$
Text of unknown authorship
Attribution model
$\mathbf{x}_u$
Most likely author
$\mathbf{x}_B$
Training texts of author B

Figure 2.1: Typical architecture of profile-based in authorship attribution approaches. [58]

A typical architecture of a profile-based approach is depicted in figure 2.1. Note that

[1]Note that this term should not be confused with instance-based learning methods[45]

x denotes a vector of text representation features. Hence, x_A is the profile of Author A, and x_u is the profile of the unseen text. The profile-based approaches have a very simple training process. Actually, the training phase just comprises the extraction of profiles for the candidate authors. Then, the attribution model is usually based on a distance function that computes the differences of the profile of an unseen text and the profile of each author. Let $PR(x)$ be the profile of text x and $d[PR(x), PR(y)]$ the distance between the profile of text x and the profile of text y. Then, the most likely author of an unseen text x is given in 2.1, where A is the set of candidate authors and x_a is the concatenation of all training texts for author a.

$$author(x) = \arg_{a \in A} min\ d(PR(x), PR(x_a)) \tag{2.1}$$

2.2.2 Instance-based approach

The majority of the modern authorship-identification approaches considers each training text sample as a unit that contributes separately to the attribution model. In other words, each text sample of known authorship is an instance of the problem in question. A typical architecture of such an instance-based approach is shown in figure 2.2. In detail, each text sample of the training corpus is represented by a vector of attributes (x) following methods described earlier, and a classification algorithm is trained using the set of instances of known authorship (*training set*) to develop an attribution model. Then, this model will be able to estimate the true author of an unseen text.

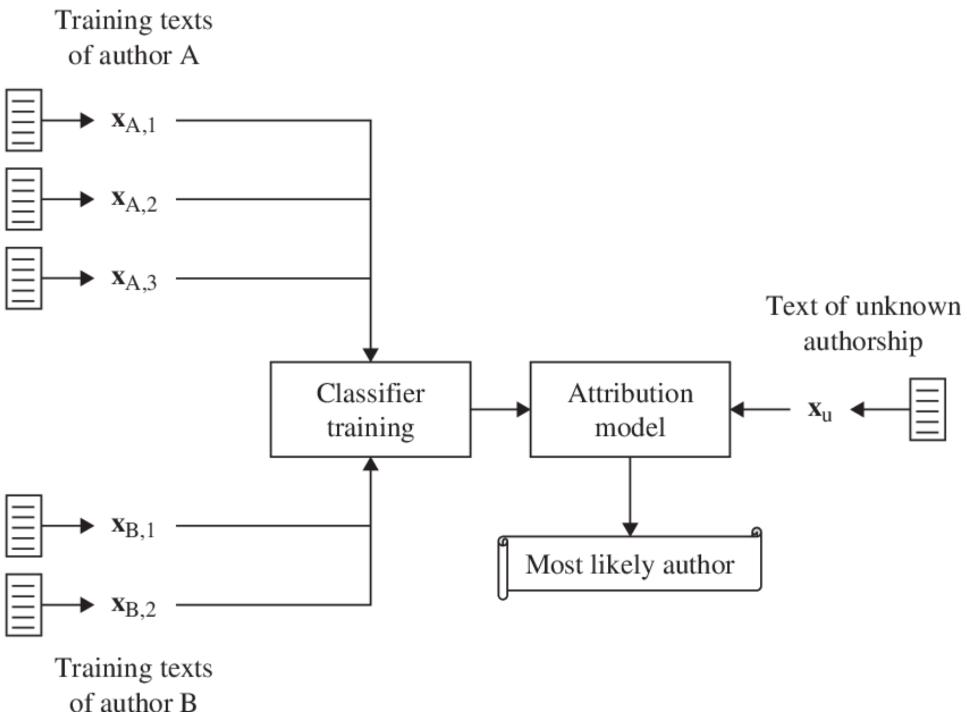

Figure 2.2: Typical architecture of instance-based in authorship attribution approaches. [58]

Note that such classification algorithms require multiple training instances per class for extracting a reliable model. Therefore, according to instance-based approaches, in case we have only one, but a quite long, training text for a particular candidate author (e.g., an entire book), this should be segmented into multiple parts, probably of equal length. From another point of view, when there are multiple training text samples of variable length per author, the training text instance length should be normalized. To that end, the training texts per author are segmented to equal-sized samples [52]. In all these cases, the text samples should be long enough so that the text representation features can adequately represent their style. Various lengths of text samples have been reported in the literature. Sanderson and Guenter (2006) produced chunks of 500 characters [52]. Koppel et al. (2006) segmented the training texts into chunks of about 500 words [31]. Hirst and Feiguina (2007) conducted experiments with text blocks of varying length (i.e.: 200, 500, and 1000 words) and reported significantly reduced accuracy as the text-block length decreases [19]. Therefore, the choice of the training instance text sample is not a trivial process and directly affects the performance of the attribution model.

2.3 Identify the task

Since authorship attribution has been studied for many decades now, we have witnessed the rise of many different subtasks. In the next part of this section we will illustrate one of the main difference between approaching a single domain or a cross domain dataset in order to get the stylometric markers of each authors. We will also discuss the difference between closed set and open set authorship attribution problem.

2.3.1 Single domain vs Cross domain

When dealing with a dataset with documents written by different authors, the first analysis to carry is to identify the different topics of each document written by the same author. This analysis is very important for the rest of the analysis, because previous works have shown a degradation in terms of performance when approaching cross domain dataset, being suited for the single domain approach. The reason behind this is trivial: content words used by the same author will vary a lot when writing about different topics (such as: motors, science, literature,or politics). What should remain steady in the style of writing of an author across different topics should be function words, the use of punctuation ..etc Although a lot of work showed good results in both context ([54], [27], [47]) for the rest of this work, we will focus on single domain authorship attribution, but we will also show results of our model trained on a cross domain dataset[2].

[2] *The Guardian*, described in Chapter 4.

2.3.2 Closed set vs Open set

The simplest kind of authorship attribution problem—and the one that has received the most attention—is the one in which we are given a small closed set of candidate authors and are asked to attribute an anonymous text to one of them. Usually, it is assumed that we have copious quantities of text by each candidate author and that the anonymous text is reasonably long. A number of recent survey papers amply cover the variety of methods used for solving this problem, known as *closed set authorship attribution.*
A significant fact that examination of the literature reveals is that nearly all research in the field only considers the simplest version of the problem.

Unfortunately, this "vanilla" version of the authorship attribution problem does not often arise in the real world. We often encounter situations in which our list of candidates might be very large and in which there is no guarantee that the true author of an anonymous text is even among the candidates. [34] Similarly, almost all work in authorship attribution has focused on the case in which the candidate set is a closed set—the anonymous text is assumed to have been written by one of the known candidates. The more general case, in which the true author of an anonymous text might not be one of the known candidates, reduces to the binary authorship verification problem: determine if the given document was written by a specific author or not. Some references of works tackling open set authorship attribution problem are shown in Koppel et al. (2011) and in Badirli et al. (2019). However, we decided to focus our work on tackling the closed set authorship attribution problem, but we will leave some thoughts for future work in the final chapter.

2.4 Research question

Statistical authorship attribution has a long history, culminating in the use of modern machine learning classification methods. Nevertheless, most of this work suffers from the limitation of assuming a small closed set of candidate authors and essentially unlimited training text for each. Real-life authorship attribution problems, however, typically fall short of this ideal. As in [32], three scenarios are presented for which solutions to the basic attribution problem are inadequate. For example, we may encounter scenarios such as:

1. There is no candidate set at all. In this case, the challenge is to provide as much demographic or psychological information as possible about the author. This is the *profiling problem.*

2. There are many thousands of candidates for each of whom we might have a very limited writing sample. This is the *needle-in-a-haystack* problem.

3. There is no closed candidate set but there is one suspect. In this case, the challenge is to determine if the suspect is or is not the author. This is the *verification problem*.

In the following section we will show how machine learning methods can be adapted to handle the special challenges of each variant.

2.4.1 Profiling the author

Even in cases where we have an anonymous text and no candidate authors, we would like to say something about the anonymous author. That is, we wish to exploit the sociolinguistic observation that different groups of people speaking or writing in a particular genre and in a particular language use that language differently [6]. This authorship profiling problem is of growing importance in the current global information environment applications, existing in forensics, security, and commercial settings. For example, authorship profiling can help police identify characteristics of the perpetrator of a crime when there are too few (or too many) specific suspects to consider. Similarly, large corporations may be interested in knowing what types of people like or dislike their products, based on analysis of blogs and online product reviews. The question we therefore ask is: How much can we discern about the author of a text simply by analyzing the text itself? It turns out that, with varying degrees of accuracy, we can say a great deal indeed. One of the approaches to authorship profiling is to apply machine learning to text categorization. The process is as follows: First, we take a given corpus of training documents, each labeled according to its category for a particular profiling dimension. For example, when addressing classification by author gender, training documents are labeled as either *male* or *female*. Each document is then processed to produce a numerical vector, each of whose elements represents some feature of the text that might help discriminate the relevant categories. A machine learning method then computes a classifier that, to the extent possible, classifies the training examples correctly. Finally, the predictive power of the classifier is tested on out-of-training data. Essentially the same paradigm can be used for authorship attribution, where the training texts are known writings of given candidate authors.[25]

2.4.2 Finding a needle in a haystack

Consider now the scenario where we seek to determine the specific identity of a document's author, but there are many thousands of potential candidates. We call this the *needle-in-a-haystack* attribution problem. In this case, standard text-classification techniques are unlikely to give reasonable accuracy and may require excessive computation time to learn classification models; however, in [32] and [33] it states that if we are willing to tolerate our system telling us it does not know the answer, we can achieve high accuracy for the cases where the system does give us an attribution it considers reliable. Recent works, [56] and [63] have addressed the problem with tackling a large set of candidate authors while keeping the documents very short, also known as "micro-messages" or "tweets"[3].

2.4.3 Verification of an author

Considering the case in which we are given examples of the writing of a single author and are asked to verify that a given target text was or was not written by this author. As a categorization problem, verification is significantly more difficult than basic attribution, and virtually no work has been done on it (but see Halteren 2004), outside the framework of plagiarism detection Zu Eissen et al. 2007. If, for example, all we wished to do is to determine if a text had been written by Shakespeare or by Marlowe, it would be sufficient to use their respective known writings, to construct a model distinguishing them, and to test the unknown text against the model. If, on the other hand, we need to determine if a text was written by Shakespeare, it is difficult to assemble a representative sample of non-Shakespeare texts. The situation in which we suspect that a given author may have written some text, but do not have an exhaustive list of alternative candidates, is a common one [32]. The problem is complicated by the fact that a single author may vary his or her style from text to text or may unconsciously drift stylistically over time, not to mention the possibility of conscious deception. Thus, we must learn to somehow distinguish between relatively shallow differences that reflect conscious or unconscious changes in an author's style and deeper differences that reflect styles of different authors.

[3]As the name recall, the name of the posts on the popular social media platform Twitter. Its main characteristics is that originally only 140 characters per post were allowed (at the moment of writing, it has been upgraded up to 280 characters per tweet).

CHAPTER 3

Text characteristics analysis

The main problem in working with language processing is that machine learning algorithms cannot work on the raw text directly. So, we need some feature extraction techniques to convert text into a matrix(or vector) of features.

In order to identify the authorship of an unknown text document using machine learning the document needs to be quantified first. The simple and natural way to characterize a document is to consider it as a sequence of tokens grouped into sentences where each token can be one of the three: word, number, punctuation mark. To quantify the overall writing style of an author, stylometric features are defined and studied in different domains. Mainly, computations of stylometric features can be categorized into five groups as lexical, character, semantic, syntactic, and application specific features. Lexical and character features mainly considers a text document as a sequence of word tokens or characters, respectively. This makes it easier to do computations compared to other features. On the other hand, syntactic and semantic features require deeper linguistic analysis and more computation time. Application specific features are defined based on the text domains or languages. These five features are studied and the methods to extract them are also provided for interested readers. Moreover there's a sixth characteristic regarding only hand-written text, which was used for years in the past and it's still studied nowadays [44], which is the *graphological analysis.* Although the problem of recognition of handwriting text is still far from its final solution, in this work, we will focus only on the first 5 characteristics because the main focus since the digitalization era has been on studies of digital text characteristics analysis.

3.1 Character Features

Based on these features a sentence consists of a characters sequence. Some of the character-level features are alphabetic characters count, digit characters count, uppercase and lowercase character counts, letter and character n-gram frequencies. This type of feature extraction techniques has been found quite useful to quantify the writing style.[16] A more practical approach in character-level features are the extraction of n-gram characters. This procedure of extracting such features are language independent and require less complex toolboxes. On the other hand, comparing to word n-grams approach the dimensional of these approaches are vastly increased and it has a curse of dimensional problem. A simple way of explaining what a character n-grams could be with the following example: assume that a word “thesis” is going to be represented by 2-gram characters. So, the resulting sets of points will be “th”, “he”, “es”, “si”, “is”.
A simple python algorithm is shown in 3.1:

Code Listing 3.1: Split word into character n-grams, parametric on n.

```
def get_char_n_gram(text, n=2):
    """Convert text into character n-grams list"""
    return [text[i:i+n] for i in range(len(text)-n+1)]

# Examples character 2-grams

print(get_char_n_gram("thesis"))
>>Out: ['th', 'he', 'es', 'si', 'is']

print(get_char_n_gram("student", n=3))
>>Out: ['stu', 'tud', 'ude', 'den', 'ent']
```

In [54] 10 character n-grams categories have been identified, being proven as the most successful feature in both single-domain and cross-domain Authorship Attribution.
These 10 categories are grouped into 3 groups: Affix n-grams, Word n-grams, Punctuation n-grams.

3.1.1 Affix n-grams

Character n-grams are generally too short to represent any deep syntax, but some of them can reflect morphology to some degree. In particular, the following affix-like features are extracted by looking at n-grams that begin or end a word:

- **prefix**: A character n-gram that covers the first n characters of a word that is at least n+1 characters long.

- **suffix**: A character n-gram that covers the last n characters of a word that is at least n + 1 characters long.
- **space-prefix**: A character n-gram that begins with a space.
- **space-suffix**: A character n-gram that ends with a space.

3.1.2 Word n-grams

While character n-grams are often too short to capture entire words, some types can capture partial words and other word-relevant tokens. There's a distinction among:

- **whole-word**: A character n-gram that covers all characters of a word that is exactly n characters long.
- **mid-word**: A character n-gram that covers n characters of a word that is at least n + 2 characters long, and that covers neither the first nor the last character of the word.
- **multi-word**: N -grams that span multiple words, identified by the presence of a space in the middle of the n-gram.

3.1.3 Punctuation n-grams

The main stylistic choices that character n-grams can capture are the author's preferences for particular patterns of punctuation. The following features characterize punctuation by its location in the n-gram.

- **beg-punct**: A character n-gram whose first character is punctuation, but middle characters are not.
- **mid-punct**: A character n-gram with at least one punctuation character that is neither the first nor the last character.
- **end-punct**: A character n-gram whose last character is punctuation, but middle characters are not.

In Table 3.1 we can see an example of the n-gram categories ($n = 3$) for the sentence *"The actors wanted to see if the pact seemed like an old-fashioned one."*. In [54] they've observed that in their data almost 80% of the n-grams in the punct-beg and punct-mid categories contain a space. They stated that "this tight coupling of punctuation and spaces is due to the rules of English orthography: most punctuation marks require a space

Table 3.1: Example of the *n*-gram categories (n=3) for the sentence: *The actors wanted to see if the pact seemed like an old-fashioned one.*
The *n*-grams that appear in more than one category are in bold.

Category	**Category n-grams**
prefix	**act** wan pac **see** lik fas
suffix	ors ted **act** med ike ned
space-prefix	_ac _wa _to _se _if _th _pa _li _an _ol _on
space suffix	he_ rs_ ed_ to_ ee_ if_ ct_ ke_ an_
whole-word	The **see** the old **one**
mid-word	cto tor ant nte eem eme ash shi hio ion **one**
multi-word	e_a s_w d_t o_s e_i f_t e_p t_s d_l n_o d_o
beg-punct	-fa
mid-punct	d-f
end-punct	ld- ne.

following them". The 20% of n-grams that have punctuation but no spaces correspond mostly to the exceptions to this rule: quotation marks, mid-word hyphens, etc. The authors of [54] conducted an experiment on Reuters Corpus Volume 1 (RCV1) dataset both the *CCAT_10 split* and the *CCAT_50 split*[1]. They've also used the Guardian dataset as a cross-domain authorship attribution experiment. In their work they stated that for the single-domain the top four categories for authorship attribution are: *prefix*, *suffix*, *space-prefix* and *mid-word*. On the other hand, for cross-domain authorship attribution the top four categories have been proven to be: *prefix*, *space-prefix*, *beg-punct* and *mid-punct*. For both single-domain and cross-domain authorship attribution, *prefix* and *space-prefix* are strong features, and are generally better than the *suffix* features, perhaps because authors have more control over prefixes in English, while suffixes are often obligatory for grammatical reasons. For cross-domain authorship attribution, *beg-punct* and *mid-punct* they found to be the top features, likely because an author's use of punctuation is consistent even when the topic changes. For single-domain authorship attribution, *mid-word* was also a good feature, probably because it captured lexical information that correlates with authors' preferences towards writing about specific topic.

3.2 Lexical Features

Lexical features relate to the words or vocabulary of a language. It is the very plain way of representing a sentence structure that consists of words, numbers, punctuation marks. These features are very first attempts to attribute authorship in earlier studies

[1]CCAT_10/CCAT_50 split on RCV1 dataset is the selection of the 10/50 most prolific authors belonging to the Corporate/Industrial Category (CCAT)

[13], [2], [61]. The main advantage of Lexical features is that it is universal and can be applied to any language easily. These features consist of bag of words representation, word N-grams, vocabulary richness, number of punctuation marks, average number of words in a sentence, and many more. Even though the number of lexical features can vary a lot, not all of them are good for every authorship attribution problem. That is why, it is important to know how to extract these features and try out different combinations on different classifiers.

3.2.1 Bag of Words

It is the representation of a sentence with frequency of words. It is a simple and efficient solution but it disregards word-order information. At the very beginning, we applied this representation to our datasets, using the already implemented *CountVectorizer* from *sklearn.feature_extraction.text.* We gave this hyper-parameter to the function:

- **max_df**=0.8
- **max_features**=10000
- **min_df**=0.02
- **ngram_range**=(1, 2)

In the approach, for each text fragment the number of instances of each unique word is found to create a vector representation of word counts. We capped the max number of features to 10'000 words and discarding the words with a higher frequency of 0.8 across the document and a lower frequency of 0.02. We've also taken into account both single word and word bi-grams. In order to give the reader a better perspective of the impact of this process for the task of authorship attribution, we choose two among the top ten most prolific authors in the RCV1 dataset: *David Lawder* and *Alexander Smith.* In Figure 3.1 we can see the number of documents written by the two selected authors in the RCV1 corpus for the *CCAT* topic.

With a simple snippet of python code shown in 3.2, we can get the most common words for an author across all the documents we gathered in the dataset.

Code Listing 3.2: Top 20 most common words in a document or a collection of document by the same author.

```
from collections import Counter

def get_most_common_words_in_df(df, n=20):
  """Split a collection of document in single word and order by
                                      frequencies across all documents"""
```

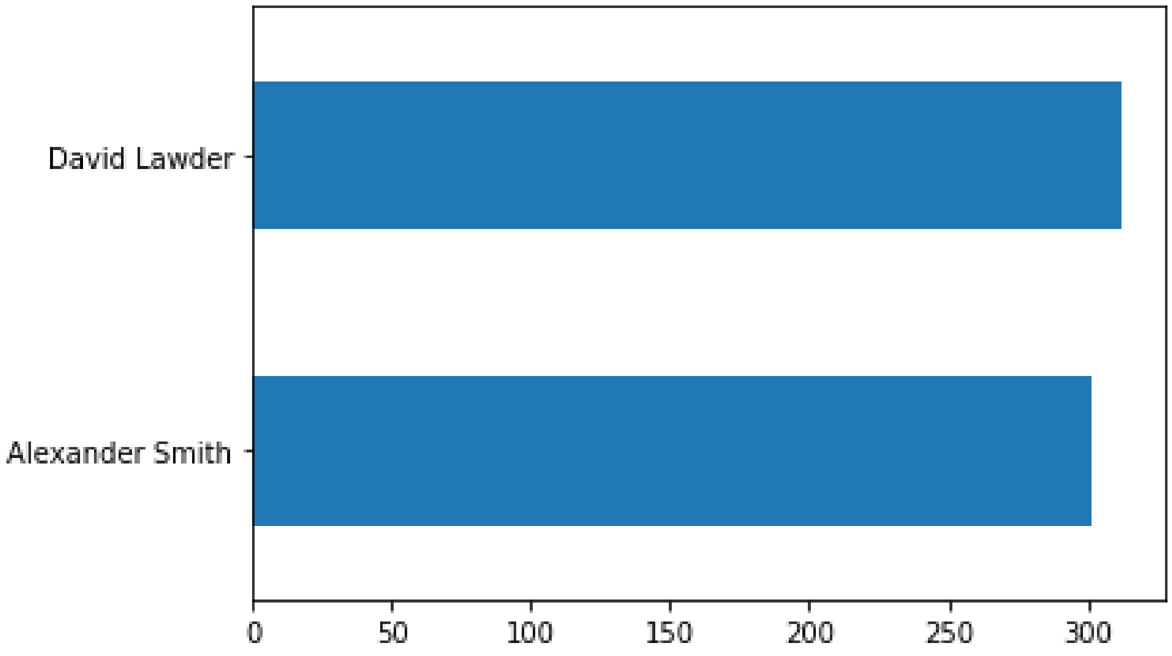

Figure 3.1: The number of documents written by David Lawder and Alexander Smith in the Reuters Corpus for the *CCAT* topic.

```
    most_common_words = Counter(" ".join(df["articles"]).split()).
                                    most_common(n)
    return most_common_words
```

```
# Examples

# 1. Top 20 words with their frequency for every document written by
                                    David Lawder

print(get_most_common_words_in_df(dataset[dataset['author']=='David
                                    Lawder']))
>>Out: [('the', 7844), ('to', 5133), ('of', 3875), ('and', 3746), ('a
                                    ', 3719), ('in', 3552), ('said',
                                    1749), ('that', 1720), ('for', 1574
                                    ), ('GM', 1453), ('on', 1286), ('at
                                    ', 1267), ('its', 1153), ('The',
                                    1098), ('with', 990), ('is', 957),
                                    ('it', 897), ('will', 875), ('by',
                                    868), ('from', 824)]

# 2. Top 20 words without their frequency for every document written
                                    by David Lawder

print([m[0] for m in get_most_common_words_in_df(dataset[dataset['
                                    author']=='David Lawder'])])
```

```
>>Out: ['the',
       'to',
       'of',
       'and',
       'a',
       'in',
       'said',
       'that',
       'for',
       'GM',
       'on',
       'at',
       'its',
       'The',
       'with',
       'is',
       'it',
       'will',
       'by',
       'from']
```

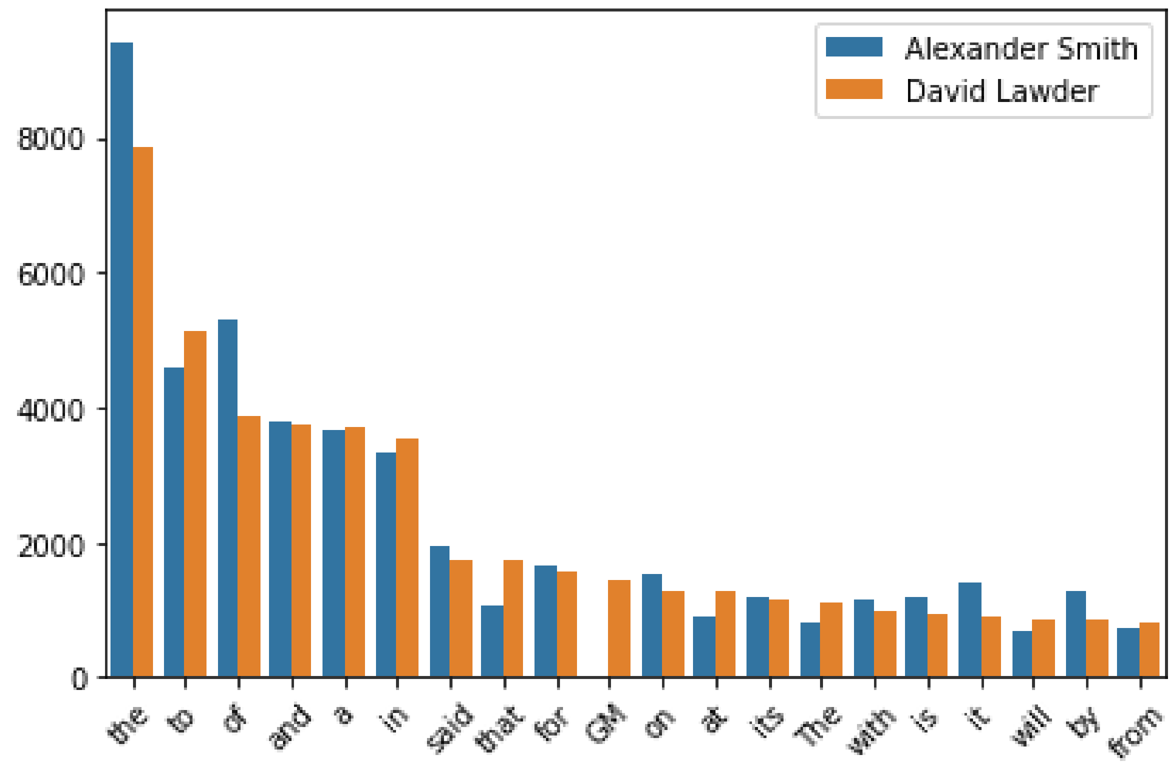

Figure 3.2: Frequency usage of the top 20 words used in the RCV1 corpus by David Lawder, compared to the frequencies of the same words in the corpus by Alexander Smith.

As expected top words are determiners that every writer uses while constructing an

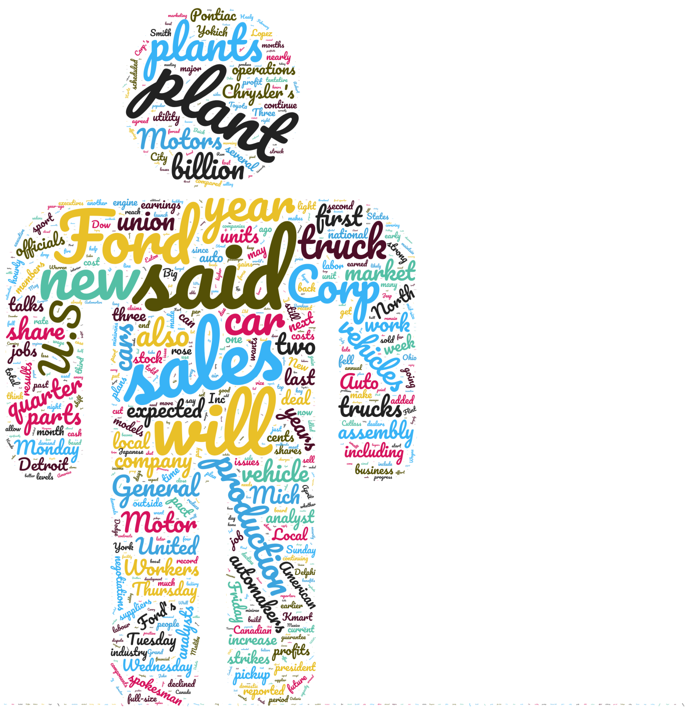

Figure 3.3: Image Generated on for every word in Reuters Corpus data for the documents written by David Lawder.

English sentence. For example, for *David Lawder* top 20 words are *"the, to, of, and, a, in, said, that, for, GM, on, at, its, The, with, is, it, will, by, from"* but for *Alexander Smith* they are *"the, of, to, and, a, in, said, was, for, on, it, had, by, be, its, is, with, would, that, as"* in decreasing order. Even though the two sets are mostly the same, the orders and the frequency are different for most authors.

The main assumption with authorship attribution problems is that every authors word usage and content differs and based on these differences the work of one author can be differentiated from the other. In order to illustrate this assumption, in Figure 3.2 we can see the top 20 words in the RCV1 corpus in the document written by David Lawder, compared to the same words in the documents of Alexander Smith.

In Figure 3.3 & Figure 3.4 we plotted using the Word Cloud identikit of David Lawder and Alexander Smith, generated by every document written by them in the RCV1 corpus.

3.2.2 Word N-grams

It is a type of probabilistic language model for predicting the next item in the form of a (n-1) order. Considering n-grams are useful since Bag of words miss out the word order when considering a text. For example, a phrasal verb "take on" can be missed out by Bag of words representation which considers "take" and "on" as two separate words. N-gram also establishes the approach of "Skip-gram" language model. An N-gram is a consecutive sub-sequence of length N of some sequence of sentence while a Skip-gram is

Figure 3.4: Image Generated on for every word in Reuters Corpus Volume 1 corpus data for the documents written by Alexander Smith.

a N-length sub-sequence where the components occur at a distance of at most k from each other [42]. In order to extract N-grams from a given text data a simple snippet of code is shown in Code Listing 3.3, tested for the word grams on the documents written by David Lawder and Alexander Smith of the RCV1 corpus data. No pre-processing on the dataset, such as discarding stopwords, has been done while constructing the N-grams. For David Lawder "the, of the, General Motors Corp." are the most occurrent uni-gram, bi-gram and three-gram respectively whilst in Alexander Smith documents they are "the, of the, said it would".

Code Listing 3.3: Get the top 3 most common grams in the corpus, for uni-grams, bi-grams and three-grams.

```
from collections import Counter
from nltk import ngrams

def get_Xigram(text, n):
  """Get n-grams for a given text. The number of grams are controlled
                                   by parameter n"""
  return list(ngrams(text.split(), n))

def get_top_3_gram(df):
  """Return a list of three elements, each one containing the top 3
                                  most common grams in the corpus
```

```
                                     given as a Dataframe parameter. The
                                      three elements corresponds to the
                                     uni-gram, bi-grams and three-grams.
                                     """
    result = []
    for i in range(1,4):
      result.append(Counter(get_Xigram(" ".join(df["articles"]), i)).
                                     most_common(3))
    return result

  print(get_top_3_gram(df_david_lawder))
  print(get_top_3_gram(df_alexander_smith))
```

Table 3.2: Top 3 uni-grams, bi-grams and three-grams by David Lawder and Alexander Smith in the Reuters corpus data.

Author	Uni-gram	Bi-gram	Three-gram
David Lawder	the	of the	General Motors Corp.
David Lawder	to	in the	United Auto Workers
David Lawder	of	for the	Ford Motor Co.
Alexander Smith	the	of the	said it would
Alexander Smith	of	in the	the end of
Alexander Smith	to	said the	a result of

Table 3.2 contains top 3 uni-grams, bi-grams, three-grams from the authors David Lawder and Alexander Smith.

3.2.3 Vocabulary Richness

It is also referred as vocabulary diversity. It attempts to quantify the diversity of the vocabulary text. It is simply the ratio of V/N where V refers to the total number of unique tokens and N refers to the total number of tokens in the considered texts [58]. As an example, we applied this feature to the RCV1 CCAT_10 dataset[2]. A snippet of the code to extract such feature, is shown in 3.4.

Code Listing 3.4: Calculate vocabulary richness in Reuters Corpus Volume 1 CCAT_10 split dataset.

```
    tmp_df = dataset
    tmp_df["num_unique_words"] = tmp_df["articles"].apply(lambda x: len
                                     (set(str(x).split()))/len(str(x).
                                     split()))
```

[2]The top ten most prolific authors in the RCV1 corpus, selecting the documents belonging to the *CCAT* topic

```
objects = {}
for author_i in tmp_df.author.unique():
  objects[author_i] = sum(tmp_df[tmp_df.author==author_i]['
                                     num_unique_words'])/len(tmp_df[
                                     tmp_df.author==author_i])

plt.bar(range(len(objects)), list(objects.values()), align='center'
                                     )
plt.xticks(range(len(objects)), list(objects.keys()))
ax = plt.gca()
plt.setp(ax.get_xticklabels(), rotation=30, horizontalalignment='
                                     right')
plt.show()
```

For this portion of the dataset, has been found the lowest vocabulary richness in *Marcel Michelson* and the highest in *Jim Gilchrist*. Overall distribution of vocabulary richness is plotted in Figure 3.5.

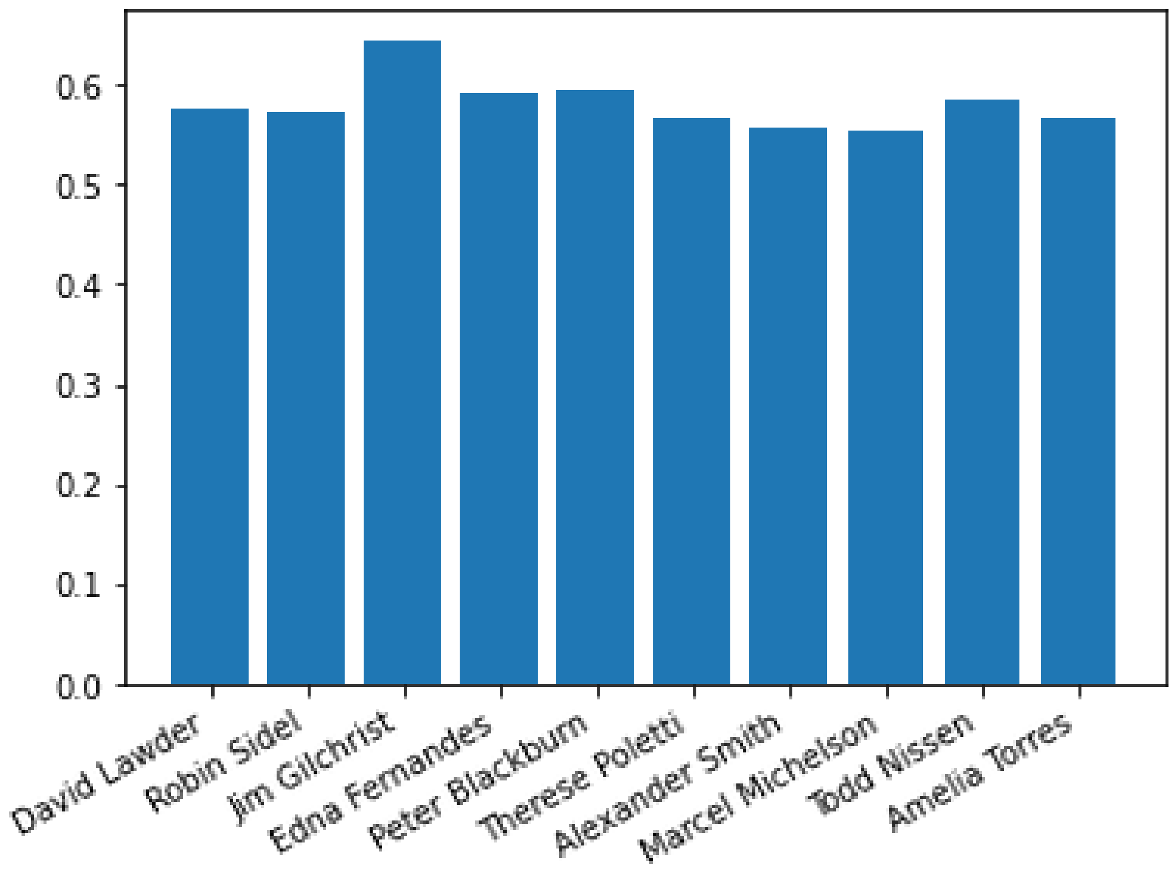

Figure 3.5: Bar Plot of vocabulary richness of Reuters Corpus CCAT_10 authors across all their documents.

3.2.4 Stylometric features

These are features such as number of sentences in a text piece, number of words in a text piece, average number of words in a sentence, average word length in a text piece, number of periods, number of exclamation marks, number of commas, number of colons, number of semicolons, number of incomplete sentences, number of uppercase, title case, camel case, lower case letters. We computed these features comparing the stylometric differences for the documents belonging to David Lawder and the ones of Alexander Smith in the RCV1 corpus. Overall distribution of some of the features introduced here (such as: *number of punctuation, number of words upper case, number of words title, average length of the word, number of stopwords*) are applied and the resulting density measures are calculated for each of the two authors and shown in Table 3.3. Among these five features introduced, number of punctuations and number of stop words usage varies the most among the authors and hence they can be better distinguisher comparing to other feature sets.

Table 3.3: Comparing some stylometric features between David Lawder and Alexander Smith calculated on the documents in the Reuters corpus data and normalized by the number of documents.

Stylometric Feature	David Lawder	Alexander Smith
Number of punctuation symbols	101.7800	88.7300
Number of uppercase words	12.5900	9.8900
Number of title words	76.5900	68.3000
Word length mean	5.0900	5.1100
Number of stopwords	191.3800	234.9700

3.2.5 Function Words

Function words are the words that have little meaning on their own but they're necessary to construct a sentence in English language. They express grammatical relationships among other words within a sentence, or specify the attitude or mood of the speaker. Some of the examples of function words might be prepositions, pronouns, auxiliary verbs, conjunctions, grammatical articles. Words that are not functions words are called as content words and they can also be studied to further analysis the use case in the authorship attribution problems. The search for a single invariant measure of textual style was natural in the early stages of stylometric analysis, but with the development of more sophisticated multivariate analysis techniques, larger sets of features could be considered. Among the earliest studies to use multivariate approaches was that of Mosteller and Wallace (1964), who considered distributions of FWs. The reason for using FWs in preference to others is that we do not expect their frequencies to vary greatly with the topic of the text, and hence, we may hope to recognize texts by the same author on different topics. It also is unlikely that the frequency of FW use can be consciously controlled, so one may hope that use of FWs for attribution will minimize the risk of being deceived [9]. Many studies since that of Mosteller and Wallace (1964) have shown the efficacy of FWs for authorship attribution in different scenarios [2], [3], [30], [69], confirming the hypothesis that different authors tend to have different characteristic patterns of FW use. Typical modern studies using FWs in English use lists of a few hundred words, including pronouns, prepositions, auxiliary and modal verbs, conjunctions, and determiners. Numbers and interjections are usually included as well since they are essentially independent of topic, although they are not, strictly speaking, FWs. Results of different studies using somewhat different lists of FW have been similar, indicating that the precise choice of FW is not crucial. Discriminators built from FW frequencies often perform at levels competitive with those constructed from more complex features.

3.2.6 Term Frequency - Inverse Document Frequency

It stands for term frequency-inverse document frequency. It is often used as a weight in feature extraction techniques. The reason why Tf-Idf is a good feature can be explained in an example. Let's assume that a text summarization needs to be done using few keywords. One strategy is to pick the most frequently occurring terms meaning words that have high term frequency (*tf*). The problem here is that, the most frequent word is a less useful metric since some words like 'a', 'the' occur very frequently across all documents. Hence, a measure of how unique a word across all text documents needs to be measured as well (*idf*). Hence, the product of *tf* x *idf* (3.3) of a word gives a measure of how frequent this word is in the document multiplied by how unique the word is with respect to the entire corpus of documents. Words with a high tf-idf score are assumed to provide the most information about that specific text [58].

$$TF(t) = \frac{\text{Number of times term t appears in a document}}{\text{Total numbers of terms in the document}} \tag{3.1}$$

$$IDF(t) = log_e(\frac{\text{Total number of documents}}{\text{Number of documents with term t in it}}) \tag{3.2}$$

$$Tf - Idf = TF(t) * IDF(t) \tag{3.3}$$

As an example, we built a Tf-Idf model by considering documents alone within the text corpus for the authors David Lawder and Alexander Smith. In the model, not only the single forms of word tokens but their n-grams are considered as well. Table 3.4 provides the top 5 words with highest Tf-Idf scores for the two authors. Comparing between Table 3.2 and Table 3.4 new meaningful words have appeared that could serve as a new feature for each author such as "dow, kmart, coupe" for David Lawder or "hsbc, pensions, panel" for Alexander Smith.

Table 3.4: Top 5 Term Frequency - Inversed Document Frequency words n-grams by David Lawder and Alexander Smith in the Reuters Corpus data.

Author	Token	Value	Author	Token	Value
David Lawder	**dow**	**0.7020**	Alexander Smith	**hsbc**	**0.6970**
David Lawder	kmart	0.6580	Alexander Smith	pensions	0.6010
David Lawder	south	0.5590	Alexander Smith	bzw	0.5920
David Lawder	coupe	0.5390	Alexander Smith	panel	0.5790
David Lawder	bags	0.5170	Alexander Smith	read	0.5700

3.3 Syntactic Features

For certain text grammatical and syntactic features could be more useful compared to lexical or character level features. However, this kind of feature extraction techniques requires specific usage of Part of Speech taggers. Some of these features consider the frequency of nouns, adjectives, verbs, adverbs, prepositions, and tense information (past tense,etc). The motivation for extracting these features is that authors tend to use similar syntactic patterns unconsciously [58]. Some researchers are also interested in exploring different dialects of the same language and building classifiers based on features derived from syntactic characteristic of the text. One great example is the work that aims to discriminate between texts written in either the Netherlandic or the Flemish variant of the Dutch language [64]. The feature set in this case consists of lexical, syntactic and word-n grams build on different classifiers and F1 has been recorded for each cases.
Employed syntactic features are function words ratio, descriptive words to nominal words ratio personal pronouns ratio, question words ratio, question mark ratio, exclamation mark ratio [64].

3.4 Semantic Features

Features that we discussed so far aim at analyzing the structural concept of a text. Semantic feature extraction from text data is a bit challenging. That might explain why there is limited work in this area. One example is the work of Yang who has proposed combination of lexical and semantic features for short text classification [67]. Their approach consists of choosing a broader domain related to target categories and then applying topic models such as *Latent Dirichlet Allocation* to learn a certain number of topics from longer documents. The most discriminative feature words of short text are then mapped to corresponding topics in longer documents [67]. Their experimental results show significant improvements compared to other related techniques studying short text classification. Positivity, neutrality, and negativity index, and synonym usage preference are good examples of semantic features. Distributed representation of words, Word2Vec, is also an attempt to extract and represent the semantic features of a word, sentence, and paragraph [41]. The usage of Word2Vec in authorship attribution tasks has not yet been studied explicitly. Due to the application domain dependency of Word2Vec features their usage will be introduced when discussing application specific feature sets.

3.4.1 Positivity and Negativity index

In order to understand the general mood and the preference of positive and negative sentence structure in each author's work, a positivity and negativity score has been calculated for the authors *David Lawder* and *Alexander Smith* for the documents in the RCV1 corpora. The code is shown in Code Listing 3.5, whereas in Figure 3.6 we can see the results that points out Alexander Smith writing more negative articles than David Lawder. Both of the authors wrote the majority of the articles classified as positive than negative.

Code Listing 3.5: Compute sentence Positivity and Negativity scores.

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
# Pre-Processing
SIA = SentimentIntensityAnalyzer()
# Applying Model, Variable Creation
sentiment = pd.concat([df_david_lawder, df_alexander_smith])
sentiment['polarity_score']=sentiment.articles.apply(lambda x:SIA.
                                  polarity_scores(x)['compound'])
sentiment['neutral_score']=sentiment.articles.apply(lambda x:SIA.
                                  polarity_scores(x)['neu'])
sentiment['negative_score']=sentiment.articles.apply(lambda x:SIA.
                                  polarity_scores(x)['neg'])
sentiment['positive_score']=sentiment.articles.apply(lambda x:SIA.
                                  polarity_scores(x)['pos'])
sentiment['sentiment']=''
sentiment.loc[sentiment.polarity_score>0,'sentiment']='POSITIVE'
sentiment.loc[sentiment.polarity_score==0,'sentiment']='NEUTRAL'
sentiment.loc[sentiment.polarity_score<0,'sentiment']='NEGATIVE'
# Normalize for Size
```

```
auth_sent= sentiment.groupby(['author','sentiment'])[['articles']].
                                  count().reset_index()
for x in ['David Lawder', 'Alexander Smith']:
auth_sent.articles[auth_sent.author == x] = (auth_sent.articles[
                                  auth_sent.author == x]/\
auth_sent[auth_sent.author ==x].articles.sum())*100
ax= sns.barplot(x='sentiment', y='articles',hue='author',data=
                                  auth_sent)
ax.set(xlabel='Author', ylabel='Sentiment Percentage')
ax.figure.suptitle("Author by Sentiment", fontsize = 24)
plt.show()
```

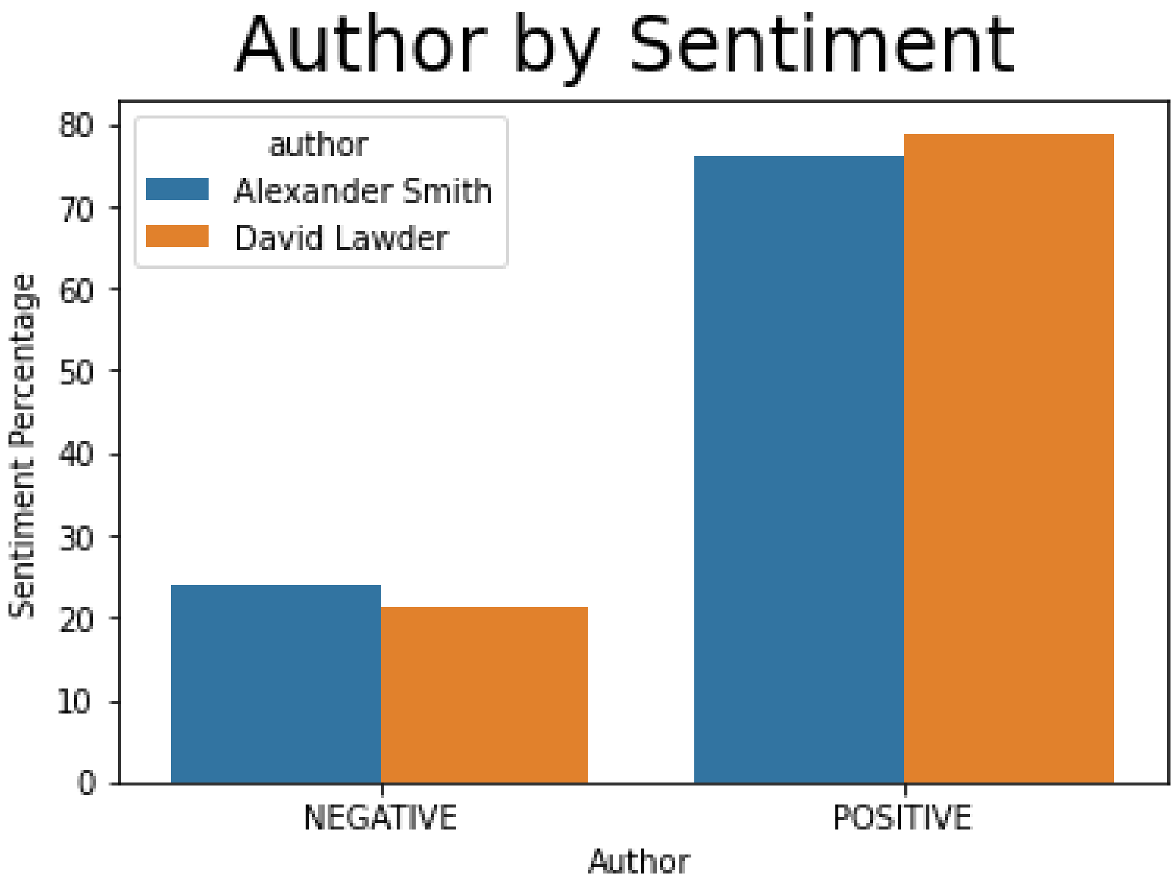

Figure 3.6: Bar Plot of sentiment analysis for 2 of the authors in the Reuters Corpus CCAT_10 corpora across all their documents.

3.5 Application Specific Features

When the application domain of the authorship attribution problems are different such as email messages or online forum messages, author style can be better characterized using structural, content specific, and language specific features. In such domains, the use of greetings and farewells, types of signatures, use of indentation, paragraph lengths, font color, font size could be good features [58].

3.5.1 Vector embeddings of words (Word2Vec)

Word2Vec[3] leverages the context of the target words. Essentially, we want to use the surrounding words to represent the target words with a Neural Network whose hidden layer encodes the word representation. Similar words are close to each other in the vector space. For example, it was shown in [43] that $vector[King] - vector[Man] + vector[Woman]$ results in the vector that is closest to the representation of the $vector[Queen]$. Figure 3.7 shows this representation in a simple way. The ways to make use of Word2Vec in the dataset is various. For example, a Word2Vec model can either be built by considering every authors text data separately, or can be imported using previously trained word vectors on other large text corpus. It can, then, be plotted into two dimensional vector space by using dimensionality reduction techniques (TSNE, for example[4]). We can also make use of pre-trained word vectors of Glove to see the difference of usages in such words between an author and a pretrained word vector. Moving with the idea of training Word2Vec per author, one can also do a cosine distance measure for the same word or same sentence. There are two types of Word2Vec, Skip-gram and Continuous Bag of Words (CBOW). I will briefly describe how these two methods work in the following paragraphs.

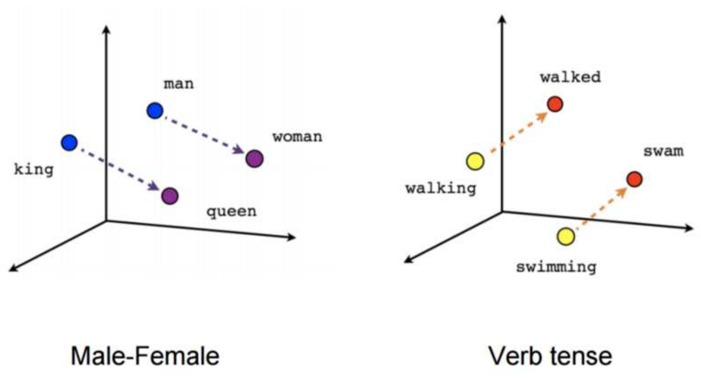

Figure 3.7: Examples of Word2Vec representation with vector distance.

3.5.1.1 Skip-gram

For skip-gram, the input is the target word, while the outputs are the words surrounding the target words. For instance, in the sentence "I have a cute dog", the input would be

[3]`https://code.google.com/archive/p/word2vec/`

[4]t-Distributed Stochastic Neighbor Embedding (t-SNE) is a technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets

"a", whereas the output is "I", "have", "cute", and "dog", assuming the window size is 5. All the input and output data are of the same dimension and one-hot encoded. The network contains 1 hidden layer whose dimension is equal to the embedding size, which is smaller than the input/output vector size. At the end of the output layer, a softmax activation function is applied so that each element of the output vector describes how likely a specific word will appear in the context. In mathematics, the softmax function, or normalized exponential function is a generalization of the logistic function that *squashes* a K-dimensional vector z of arbitrary real values to a K-dimensional vector δ(z) of real values, where each entry is in the range (0,1) and all the entries add up to 1. The target is a (K-1)-dimensional space, so one dimension has been lost.

With skip-gram, the representation dimension decreases from the vocabulary size (V) to the length of the hidden layer (N). Furthermore, the vectors are more "meaningful" in terms of describing the relationship between words. The vectors obtained by subtracting two related words sometimes express a meaningful concept such as gender or verb tense, as shown in the following figure (dimensionality reduced).

3.5.1.2 Continuous Bag Of Words (CBOW)

Continuous Bag of Words (CBOW)[5] is very similar to skip-gram, except that it swaps the input and output. The idea is that given a context, we want to know which word is most likely to appear in it.
The biggest difference between Skip-gram and CBOW is that the way the word vectors are generated. For CBOW, all the examples with the target word as target are fed into the networks, and taking the average of the extracted hidden layer. For example, assume we only have two sentences, "He is a nice guy" and "She is a wise queen". To compute the word representation for the word "a", we need to feed in these two examples, "He is nice guy", and "She is wise queen" into the Neural Network and take the average of the value in the hidden layer. Skip-gram only feed in the one and only one target word one-hot vector as input.

It is claimed that Skip-gram tends to do better in rare words. Nevertheless, the performance of Skip-gram and CBOW are generally similar.

3.5.2 Vector embeddings of documents (Doc2Vec)

Distributed word representation in a vector space (word embeddings) is a novel technique that allows to represent words in terms of the elements in the neighborhood. Distributed representations can be extended to larger language structures like phrases, sentences,

[5]`https://iksinc.online/tag/continuous-bag-of-words-cbow/`

paragraphs and documents. The capability to encode semantic information of texts and the ability to handle high-dimensional datasets are the reasons why this representation is widely used in various natural language processing tasks such as text summarization, sentiment analysis and syntactic parsing [36].

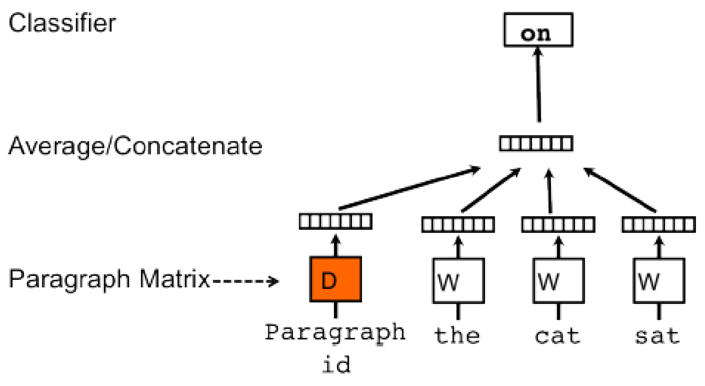

Figure 3.8: Paragraph Vector-Distributed Memory model.

The goal of doc2vec is to create a numeric representation of a document, regardless of its length. Unlike words, documents do not come in logical structures such as words, so the another method has to be found. The concept that Mikilov and Le [36] had used was simple, yet clever: they used the word2vec model, but instead of using just words to predict the next word, we also added another feature vector, which is document-unique. When training the word vectors W, the document vector D is trained as well, and in the end of training, it holds a numeric representation of the document.

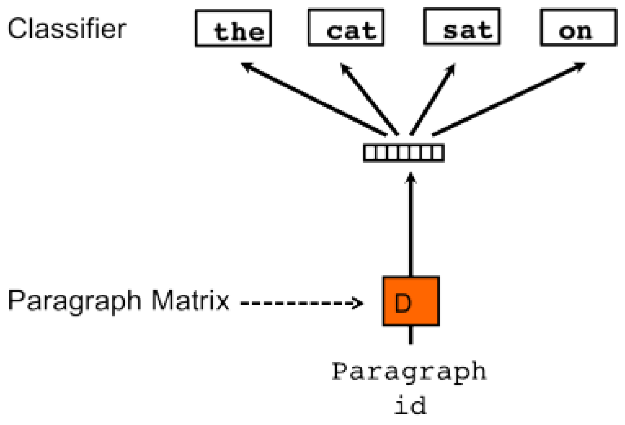

Figure 3.9: Paragraph Vector-Distributed Bag of Words model.

The model above is called *Distributed Memory version of Paragraph Vector* (PV-DM) Figure 3.8. It acts as a memory that remembers what is missing from the current context — or as the topic of the paragraph. While the word vectors represent the concept of a word, the document vector intends to represent the concept of a document. As in word2vec, another algorithm, which is similar to skip-gram may be used *Distributed Bag of Words version of Paragraph Vector* (PV-DBOW). As we can see in Figure 3.9, this algorithm is actually faster (as opposed to word2vec) and consumes less memory, since there is no need to save the word vectors.

CHAPTER 4

Techniques and data collection for Authorship Attribution

'*State of the Art is the frenetic and relentless pursuit of doing what its best at that time!*'
Da Anunciação Marco

There are different types of authorship attribution studies in the literature such as predicting the date of authorship of historical texts or text genre detection [62], [26]. Vast majority of previous works focuses on authorship identification by taking into consideration the stylistic features of authors such as use of grammar, function words, frequent word allocations [2], [13], [19]. Some of the well-known problems in authorship attribution are disputed Federalist Papers classification and Shakespearean Authorship Dispute. The Federalist Papers are a collection of 85 articles and essays written by Alexander Hamilton, James Madison, and John Jay to persuade the citizens of New York to ratify the U.S. Constitution. Authorship of twelve of these papers has been in dispute. To address this problem, using linear support vector machines as classifier and relative frequencies of words as features a study identified these papers to be written by James Madison [46].
Another dispute in authorship attribution among scholars across the world is whether William Shakespeare wrote the works attributed to him or not. It was argued that Shakespeare was not even educated and more than 80 authors were suggested to be the author of the writings that were under the name of Shakespeare. Christopher Marlowe is considered the most likely candidate to write these works under the name of Shakespeare when he was in jail. In order to analyze the stylistic fingerprint of Shakespeare and

Marlowe and non-Shakespearean authors, namely Chapman, Jonson, Middleton, a corpus has been put together [13].
The classification results for non-Shakespearean author candidates turned out to be highly accurate (Johnson %100, Chapman %92.9 and Middleton %88.9). The results supported the hypothesis that writing styles of Marlowe and Shakespeare were as distinguishable as other authors unless Marlowe did not show a linear change in style over time. Meaning, Marlowe has found not to be the authors of Shakespearean writings. Another interesting study on the unknown texts is also done based on word-level features, vocabulary richness and syntactic features by using Liblinear SVM for classification purposes [61]. Even though the classification accuracy results are not as high as other related works features like 'number of unique words' should be noted for use in any attribution problem.
Usefulness of function words in authorship attribution is introduced by Mosteller and Wallace in their work on Federalist papers [46]. Argamon and Levitan has compared the characteristic features of frequent words, pairs and collocations using the SMO algorithm, and implemented it for two class (American or British) author nationality classification problem. Their results conclude that function words are useful as stylistic text attribution and frequent words are the best features among others. The reason behind it is that a given same size frequent collocations has less different words comparing to frequent words so it carries less discriminatory features [2]. In summary, there has been substantial work done in authorship attribution and mainly people in forensic linguistic or computer scientists aim to build 'stylistic fingerprint of author' by using several features of a given text such as function words, stylometry. It is a classification problem and several classifiers are used such as Naïve Bayes, SVM. Among them, SVM is observed to fit best for these kinds of problems.

4.1 Support Vector Machine

Support Vector Machines (SVMs) recently gained popularity in the learning community [65]. In its simplest linear form, an SVM is a hyperplane that separates a set of positive examples from a set of negative examples with maximum interclass distance, the margin. Figure 4.1 shows such a hyperplane with the associated margin. The formula for the output of a linear SVM is show in Equation 4.1, where w is the normal vector to the hyperplane, and x is the input vector. The margin is defined by the distance of the hyperplane to the nearest of the positive and negative examples.

$$u = w * x + b \tag{4.1}$$

Maximizing the margin can be expressed as an optimization problem, as shown in Equation 4.2:

$$minimize\frac{1}{2}||w||^2 \ subject \ to \ y_i(w * x_i + b) \geq 1, \forall_i \tag{4.2}$$

where x_i is the $i - th$ training example and $y_i \in -1, 1$ is the correct output of the SVM for the $i - th$ training example. Note that the hyperplane is only determined by the training instances x_i on the margin, *the support vectors*. Support vector machines are based on the structural risk minimization principle from computational learning theory [65]. The idea is to find a model for which we can guarantee the lowest true error. This limits the probability that the model will make an error on an unseen and randomly selected test example. An SVM finds a model which minimizes (approximately) a bound on the true error by controlling the model complexity (VC-Dimension). This avoids over-fitting, which is the main problem for other semi-parametric models.

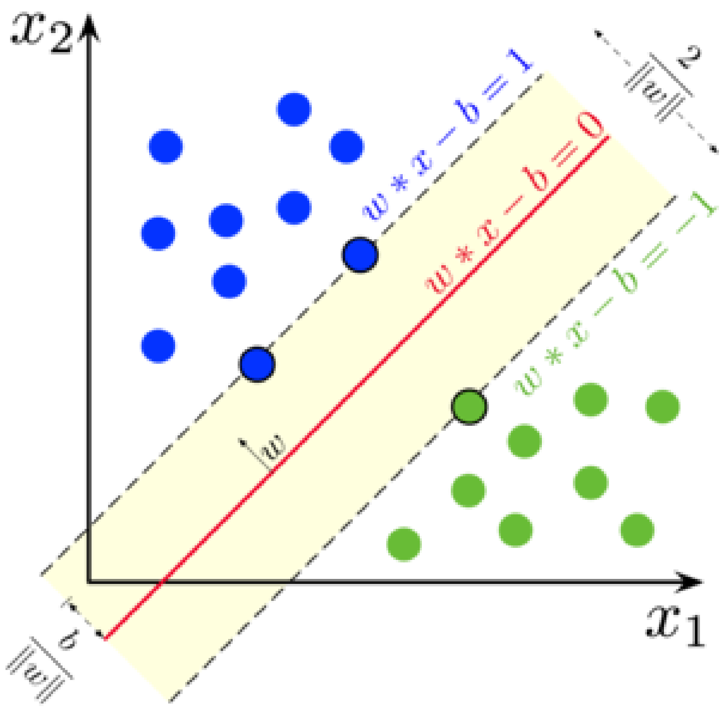

Figure 4.1: Support Vector Machine Hyperplane with the associated margin formula.

4.1.1 Support Vector Machine for authorship attribution

Unlike currently used classification approaches, like neural networks or decision trees, SVM allows for the processing of hundreds of thousands of features. This offers the opportunity to use all words of a text as inputs instead of a few hundred carefully selected characteristic words only. In similar text classification problems aiming at thematic categorization, the SVM has been shown to be quite effective [23], [12]. A SVM is able to classify a text with respect to content. In the framework of author attribution, it is not clear whether a specific topic addressed by the author or the structural or stylistic features of the authors language lead to a successful classification. Among the earliest methods to be applied were various types of neural networks, typically using small sets of FWs as features [21]. More recently, Hirst and Feiguina used neural networks on a wide variety of features. Other studies have used k-nearest neighbor [69], support vector machines [10], [30], [70], and Bayesian regression [37]. Comparative studies on machine learning methods for topic-based text categorization problems [12], [23] have shown that in general, support vector machine (SVM) learning is at least as good for text categorization as any other learning method, and the same has been found for authorship attribution [70]. The distinctive advantage of the SVM for text categorization is its ability to process many thousand different inputs. This opens the opportunity to use all words in a text directly as features. For each word w_i the number of times of occurrence is recorded. Typically a corpus contains more than 100,000 different words, with each text covering only a small fraction. Joachims [23] used the SVM for the classification of text into different topic categories. As features he uses word stems. To establish statistically significant features he requires that each feature occurs at least three times in a text. The empirical evaluation was done on two test collections: the Reuter-21578 news agency data set covering different topics and the Ohsumed corpus of William Hersh describing diseases. Using about 10000 features in every case, the two SVM versions (polynomial and rbf) performed substantially better than the currently best performing conventional methods (naive Bayes, Rocchio, decision trees, k-nearest neighbor). Joachims et al. [24] used a transductive SVM for text categorization which is able to exploit the information in unlabeled training data. Dumais et al. [12] use linear SVMs for text categorization because they are both accurate and fast. They are 35 times faster to train than the next most accurate (a decision tree) of the tested classifiers. They applied SVMs to the Reuter-21578 collection, emails and web pages. Drucker et al. [11] classify emails as spam and non spam. They find that boosting trees and SVMs have similar performance in terms of accuracy and speed. SVMs train significantly faster.

4.2 Studies on Reuters Corpus

Reuters is the world's largest international multimedia news agency, providing myriad news and mutual fund information available on `Reuters.com`, video, mobile, and interactive television platforms. Reuters Corpus Volume 1 (RCV1) is drawn from one of those online databases[1]. This dataset consists of all English language stories produced by Reuters journalists between August 20, 1996 and August 19, 1997. The dataset is made available on two CD-ROMs and has been formatted in XML by Reuters, Ltd. in 2000, for research

[1]Reuters corpora [Online]. Available, `http://trec.nist.gov/data/reuters/reuters.html`; 2000

purposes. Both the archiving process and later preparation of the XML dataset involved substantial verification and validation of the content, attempts to remove spurious or duplicated documents, normalization of dateline and byline formats, addition of copyright statements, and so on [8]. The stories cover a range of content typical of a large English language international newswire. They vary from a few hundred to several thousand words in length.

It consists of a collection of newswire stories written in English that cover four main topics: corporate/industrial (CCAT), economics (ECAT), government/social (GCAT) and markets (MCAT). Although it was not compiled for authorship attribution task, it has been adapted to this task in previous works. For example, in [57]; [48] the 10 most prolific authors were chosen from the CCAT category, and then, 50 examples per author for training and 50 examples for testing were selected randomly with no overlapping between training and testing sets. In further sections, we will reference to this corpus as RCV1-10.

In Houvardas and Stamatatos [22], the authors proposed another adaptation of the RCV1 corpus for the authorship attribution task. They choose the 50 most prolific authors from the Reuters Corpus, keeping 50 examples per author for training and 50 examples per author for testing with no overlapping between them. We will refer to this corpus as RCV1-50.

The RCV1-10 and RCV1-50 datasets are both balanced over different authors and have their genre fixed to news. The main category of the news in both cases is fixed to corporate/industrial, but there are many subtopics covered in the news and the length of the texts is short (from 2 to 8 KBytes). These corpora resemble a more realistic scenario, when the amount of texts is limited and the number of candidate authors is large.

4.2.1 Studies on Reuters Corpus on authorship attribution

Although, not particularly designed for evaluating author identification approaches, the RCV1 corpus contains 'by-lines' in many documents indicating authorship. In particular, there are 109,433 texts with indicated authorship and 2,361 different authors in total. RCV1 texts are short (approximately 2KBytes – 8KBytes), so they resemble a realworld author identification task where only short text samples per author may be available. Moreover, all the texts belong to the same text genre (newswire stories), so the genre factor is reduced in distinguishing among the texts. On the other hand, there are many duplicates (exactly the same or plagiarized texts). The RCV1 corpus has already been used in author identification experiments. In Khmelev and Teahan [28] the top 50 authors (with respect to total size of articles) were selected. Moreover, in the framework of the AuthorID project, the top 114 authors of RCV1 with at least 200 available text samples were selected [37]. In contrast to these approaches, in this study, the criterion for selecting the authors was the topic of the available text samples. Hence, the top 50 authors of texts labeled with at least one subtopic of the class CCAT (corporate/industrial) were selected. That way, it is attempted to minimize the topic factor in distinguishing among the texts. Therefore, since steps to reduce the impact of genre have been taken, it is to be hoped that authorship differences will be a more significant factor in differentiating the texts. Consequently, it is more difficult to distinguish among authors when all the text samples deal with similar topics rather than when some authors deal mainly with economics, others with foreign affairs etc. The training corpus consists of 2,500 texts (50 per author) and the test corpus includes other 2,500 texts (50 per author) non-overlapping with the training texts [22].

4.3 The guardian corpus: a case of cross-topic authorship attribution

First introduced by Stamatatos [59], *The Guardian corpus* is composed of texts published in The Guardian daily newspaper. The texts were downloaded using the publicly available API[2] and preprocessed to keep the unformatted main text. The majority of the corpus comprises opinion articles (comments). The newspaper describes the opinion articles using a set of tags indicating its subject. There are eight top-level tags (World, U.S., U.K., Belief, Culture, Life&Style, Politics, Society), each one of them having multiple subtags. It is possible (and very common) for an article to be described by multiple tags belonging to different main categories (e.g., a specific article may simultaneously belong to U.K., Politics, and Society). In order to have a clearer picture of the thematic area of the collected texts, they only used articles that belong to a single main category. Therefore, each article can be described by multiple tags, all of them belonging to a single main category. Moreover, articles coauthored by multiple authors were discarded. In addition to opinion articles on several thematic areas, the presented corpus comprises a second text genre-book reviews. The book reviews are also described by a set of tags similar to the opinion articles. However, no thematic tag restriction was taken into account when collecting book reviews. Note that since all texts come from the same newspaper, they are expected to have been edited according to the same rules, so any significant difference among the texts is not likely to be attributed to the editing process.

Table 6.6 shows details about The Guardian Corpus ("TGC"). It comprises texts from thirteen authors selected on the basis of having published texts in multiple thematic areas (Politics, Society, World, U.K.) and different genres (opinion articles and book reviews). At most 100 texts per author and category have been collected—all of them published

[2] *Open Platform*, GUARDIAN, `http://explorer.content.guardianapis.com/`

Table 4.1: Details about *The Guardian Corpus* ("TGC") distribution in every topic among the authors.

Author	Politics	Society	World	UK	Books
CB	12	4	11	14	16
GM	6	3	41	3	0
HY	8	6	35	5	3
JF	9	1	100	16	2
MK	7	0	36	3	2
MR	8	12	23	24	4
NC	30	2	9	7	5
PP	14	1	66	10	72
PT	17	36	12	5	4
RH	22	4	3	15	39
SH	100	5	5	6	2
WH	17	6	22	5	7
ZW	4	14	14	6	4
Total:	254	94	377	119	160

within a decade (from 1999 to 2009). Note that the opinion article thematic areas can be divided into two pairs of low similarity, namely Politics-Society and World-U.K. In other words, the Politics texts are more likely to have some thematic similarities with World or U.K. texts than with the Society texts. TGC provides texts on two different genres from the same set of authors. Moreover, one genre is divided into four thematic areas. Therefore, it can be used to examine authorship attribution models under cross-genre and cross-topic conditions [15]. Stamatatos (2013) demonstrated that high frequency character n-grams allow to discriminate effectively between authors not only for single-topic authorship attribution, but also for cross-topic authorship attribution. Sapkota et al. [53] improved the prediction results in cross-topic authorship attribution using an enriched training corpus in order to predict authors on a corpus with different topics. The role of preprocessing steps was evaluated in [38]. The approach proposed in that paper is considered to be more topic-neutral by their authors, because they replace the named entities and some topic-related words while preprocessing the corpus. Their approach showed the importance of preprocessing, because it gave the improvement of 4%. In [59], it is mentioned that the use of semantic features for the authorship attribution task usually improves the obtained results, however, very few attempts have been done to exploit high-level features for stylometric purposes. More recently, the usage of the distributed document representation for the cross-topic authorship attribution task has shown great results in terms of accuracy [49] and [15], because of its capability to encode the semantic information of texts in a low dimension vector,. Recently, the Paragraph Vector (Doc2vec) model was proposed by Le & Mikolov [36] for learning distributed representation for both sentences and documents. The Doc2vec model basically treats each document as a special word and learns both document vectors and word vectors simultaneously by predicting the target word. Vectors obtained by the Doc2vec model outperforms both bag-of-words and word ngrams models producing the new state-of-the-art results for several text classification and sentiment analysis tasks [15]. *The Guardian corpus* offers the opportunity to explore a scenario with different topics under the same genre with the exception of the category "Books reviews," which is considered as another genre. It is assumed that each category represents a topic, which is different enough from the other categories. In contrast to the previously described benchmarks, The Guardian is a cross-topic, cross-genre and unbalanced benchmark, representing in this way a very challenging scenario.

At the beginning of this work, we wondered which dataset would best validate our work. In fact, we quickly realized the importance of using a dataset that did not present bias and invalidate our results. Contrary to some previous work ([10], [29]), we thought it was not appropriate to use datasets that could not be reproduced by other authors and on which results could not be compared. In the work of Potthast et al. [50], the authors performed the reimplementation of 15 authorship attribution methodologies and concluded that very few of them achieve consistent results across different corpora. In this work, we show the consistency of our approach on 4 different datasets, evaluating a wide range of testing scenarios [49]. Despite what we had initially included, datasets used in the literature such as Enron's email collection and documents collected experimentally, we decided not to use them because they were not recognized as valid by other studies on authorship attribution. We would have liked to obtain the twitter dataset used by Layton et al. in [35], to have a comparison also with very short text; we wrote him and his collaborators an email but we never got a reply :(.
The datasets we selected at the end of the selection process are the following:

- **RCV1**: both in the form RCV1_10 and RCV1_50, i.e. 10 and 50 authors always belonging to the CCAT category, in order to reproduce the same conditions of previous related works.
- **GDELT**: A selection of 45 authors taken from a work of [17], selected by BigQuery, available on the GDELT project[3].
- **AFR**[4]: This dataset consists of reviews of fine foods from amazon. The data span

[3]The GDELT Project is one of the largest publicly available digitized book database which has more than 3.5 million books published from 1800-2015. The GDELT Project is an open platform for research and analysis of global society and thus all datasets released by the GDELT Project are available for unlimited and unrestricted use for any academic, commercial, or governmental use of any kind without any fee. `https://www.gdeltproject.org/about.html`

[4]Amazon Food Reviews available at `https://snap.stanford.edu/data/web-FineFoods.html`

a period of more than 10 years, including all 500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.

All previously selected datasets were used as authorship attribution closed set analysis and in single topic context. We then also selected the TGC dataset (The Guardian Corpus) as a comparison and benchmark of our study for a cross-topic and cross-genre dataset to show the performance of the model in both single topic and cross topic contexts and understand critical points and differences between them.

CHAPTER 5

Our approach

In authorship attribution problems, there is a set of candidate authors and a set of text samples in the training set covering some of the works of the authors. In the test dataset, there are samples of texts and each of them needs to be attributed to a candidate author. In the next sections, we are going to describe the experiment we carried out taking care of the chronological path of the events. Our main focus has always been on closed set authorship attribution, training with instance-based approach (i.e. extracting features by not considering the other available text samples in the training). The three milestones can be summarized as follows:

1. Dataset selection and preparation
2. Method selection
3. Features extraction

5.1 Dataset preparation

In section 4.6 we have already shown the datasets we selected. In particular, in this section we are going to show the procedures done to prepare the datasets for the next steps. For the single topic authorship attribution task we decided to select the RCV1 dataset, the dataset of 45 Victorian era book authors from the GDELT project and the dataset of amazon food reviews collected in the first decade of the 2000s. Regarding the cross domain authorship attribution task, we selected the dataset extracted from The Guardian newspaper.

5.1.1 Reuters Corpus

It consists of a collection of newswire stories written in English that cover four main topics: corporate/industrial (CCAT), economics (ECAT), government/social (GCAT) and markets (MCAT). We sent a request to obtain the dataset on this webpage `https://trec.nist.gov/data/reuters/reuters.html`. After few days, we gathered the RCV1 Corpus as it contains 810,000 Reuters, English Language News stories (about 2.5 GB). First of all we had to convert the dataset, that contained folders of xml files, into a big csv with author's labels and document text. Code Listing 5.1 shows the process of documents and authors extraction, using 'xml' python library. We decided to take into account this properties of the document: *text, title, headline, byline, dateline, lang, corpus_path, corpus_subdirectory, corpus_filename.*

Code Listing 5.1: Extract and Parse RCV1 XML document into csv.

```
import os
import xml.etree.ElementTree as ET

for f in files:
  try:
    data_path = os.sep.join([dir_path, f])
    raw_data = open(data_path).read()
    try:
      xml_parse = ET.fromstring(raw_data)
    except:
      print(D,"/",f,"failed to parse XML.")
      continue

    def get_text(tag):
      stuff = xml_parse.find(tag)
      if stuff:
        return stuff.text
      else:
        return None

  text = "\n\n".join([str(p.text) for p in xml_parse.findall(".//p")]
                                     )
  title = get_text("title")
  headline = get_text("headline")
  byline = get_text("byline")
  dateline = get_text("dateline")

  #this bit got funky in the XML parse
  lang_key = [k for k in xml_parse.attrib if "lang" in k][0]
  lang = xml_parse.attrib[lang_key]
```

```
    code_classes = [c.attrib["class"]
    for c in xml_parse.findall(".//codes")]
    codes = {cc: [c.attrib["code"] for c in
      xml_parse.findall(".//codes[@class='%s']/code"%cc)]
      for cc in code_classes}
    dcs = {d.attrib["element"]: d.attrib["value"]
      for d in xml_parse.findall(".//dc")}

    #assemble output
    output = {"text": text,
      "title": title,
      "headline": headline,
      "byline": byline,
      "dateline": dateline,
      "lang": lang,
      "corpus_path": corpus_path,
      "corpus_subdirectory": D,
      "corpus_filename": f,
    }

    # merge and flatten the other big hashmaps
    output.update(codes.items())
    output.update(dcs.items())

    result.append(output)
  except Exception as e:
    print(e)
```

The dataset was then filtered only with the documents with a *"byline"* property defined. We end up with 109'433 documents written by 2400 distinct authors. At this point, we labeled this portion of the RCV1 original dataset as the *"Full RCV1 dataset"*. In order to test and compare our approach, reproducing the testing scenario described in previous research [58], the 10 most prolific authors were chosen from the CCAT category, and then, 50 examples per author for training and 50 examples for testing were selected randomly with no overlapping between training and testing sets. We will reference to this portion of the RCV1 dataset as the *"RCV1_10"*. In previous works [22], the authors proposed another adaptation of the RCV1 corpus for the authorship attribution task. They choose the 50 most prolific authors from the Reuters Corpus, keeping 50 examples per author for training and 50 examples per author for testing with no overlapping between them. We will refer to this corpus as the *RCV1_50*.
The RCV1_10 and RCV1_50 datasets are both balanced over different authors and have their genre fixed to news. The majority of our work has been conducted on the RCV1_50,

although to compare results with previous works we will show also the same techniques applied to the RCV1_10 corpus. Table 5.1 shows the main metrics to describe these different portions of the original dataset.

Table 5.1: Main metrics to describe different portion of the Reuters Corpus dataset.

Name	N# docs	N# authors	Avg docs length	Avg n# docs/author
Full RCV1	109433	2400	3061.95	45.60
RCV1_10	1000	10	3093.82	100
RCV1_50	5000	50	3251.16	100

5.1.2 GDELT

The GDELT Project is one of the largest publicly available digitized book database which has more than 3.5 million books published from 1800-2015. The GDELT Project is an open platform for research and analysis of global society and thus all datasets released by the GDELT Project are available for unlimited and unrestricted use for any academic, commercial, or governmental use of any kind without any fee[1]. The whole digitized dataset is publicly available and interested researchers can freely perform SQL queries using the Google big query platform. For example; the book names, publication year, quotations, themes, the original text of the book of "Mark Twain" which were written between 1890 to 1900 can be found as follows using the Big query platform of Google in Code Listing 5.2.

Code Listing 5.2: Google Big Query on GDELT.

```
SELECT Themes, V2Themes, Quotations, AllNames,
TranslationInfo, BookMeta_Identifier, BookMeta_Title,
BookMeta_Creator, BookMeta_Subjects, BookMeta_Year,
FROM (TABLE_QUERY([gdelt-bq:internetarchivebooks],
'REGEXP_EXTRACT(table_id, r"(\d{4})") BETWEEN "1890"
AND "1900"'))
WHERE BookMeta_Creator CONTAINS "Mark Twain"
LIMIT 50
```

To decrease the bias and create a reliable dataset the following criteria have been chosen to filter out authors: English language writing authors, authors that have enough books available (at least 5), 19th century authors. With these criteria 50 authors have been selected and their books were queried through Big Query Gdelt database. The next

[1]https://www.gdeltproject.org/about.html

task has been cleaning the dataset due to OCR reading problems in the original raw form. To achieve that, firstly all books have been scanned through to get the overall number of unique words and each words frequencies. While scanning the texts, the first 500 words and the last 500 words have been removed to take out specific features such as the name of the author, the name of the book and other word specific features that could make the classification task easier. After this step, we have chosen top 10, 000 words that occurred in the whole 50 authors text data corpus. The words that are not in top 10, 000 words were removed while keeping the rest of the sentence structure intact. Afterwards, the words are represented with numbers from 1 to 10, 000 reverse ordered according to their frequencies. The entire book is split into text fragments with 1000 words each. We separately maintained author and book identification number for each one of them in different arrays. Text segments with less than 1000 words were filled with zeros to keep them in the dataset as well. 1000 words make approximately 2 pages of writing, which is long enough to extract a variety of features from the document. The reason why we have represented top 10, 000 words with numbers is to keep the anonymity of texts and allow researchers to run feature extraction techniques faster. Dealing with large amounts of text data can be more challenging than numerical data for some feature extraction techniques. When gathering the dataset, we decided to discard 5 authors for which their writings were not consistent enough for the authorship attribution task. We ended up with a full dataset with 53'678 documents instances, each one containing 1000 words. In order to make training methods reliable across dataset, we decided to select 100 documents of each authors, with a 50/50 split (i.e. 50 documents in the training set, 50 documents in the testing set, no overlapping among them). In the following sections, we will refer to this as the *"GDELT_45"*. Table 5.2 shows the metrics that describe best this dataset.

Table 5.2: Main metrics to describe different portion of the GDELT Corpus dataset.

Name	N# docs	N# authors	Avg docs length	Avg n# docs/author
Full GDELT	53678	45	4950.61	1192.84
GDELT_45	4500	45	4911.91	100

5.1.3 Amazon Food Reviews

This dataset consists of reviews of fine foods from amazon. The data span of over a period of more than 10 years, including all 500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories. We decided to consider this dataset for our experiment, because we were missing a more "everyday" example of dataset to work with. As Table 5.3 shows, the average documents length is dramatically lower than the other two datasets presented previously, providing us with a good challenge to show consistency of our method across all these different scenarios. Moreover, in order to make training methods reliable across dataset, we decided to select 100 reviews of each customers, with a 50/50 split (i.e. 50 reviews in the training set, 50 reviews in the testing set, no overlapping among them). In the following sections, we will refer to this as the *"AFR_50"*.

Table 5.3: Main metrics to describe different portion of the Amazon Food Reviews dataset.

Name	N# docs	N# authors	Avg docs length	Avg n# docs/author
Full AFR	568454	256059	380.70	2.2
AFR_50	5000	50	990.45	100

5.1.4 The Guardian newspaper

Although the majority of our time and effort was focused on the first 3 single domain datasets for closed set authorship attribution task, we wanted to test our approach with a cross domain dataset. *The Guardian corpus* is composed of texts published in The Guardian daily newspaper. The majority of the corpus comprises opinion articles (comments). The newspaper describes the opinion articles using a set of tags indicating its subject. There are eight top-level tags (World, U.S., U.K., Belief, Culture, Life&Style, Politics, Society), each one of them having multiple subtags. In order to test and compare our approach, we reproduce the testing scenario described in the previous research [59] using the Guardian corpus. The experimental scenario is as follows:

1. Select at most ten samples per author in each topic category (in Figure 5.1 we can see the distribution of the samples per author for the Politics category after considering the restriction of ten samples per author)
2. Use the samples in the Politics category as training set and train the classifier
3. Finally, test the classifier using another topic category different from Politics (four possible pairings)

5.2 Features extraction

After the choice of dataset and classification method, all our energies were spent on the choice of feature extraction. In order to identify the authorship of an unknown text document using machine learning the document needs to be quantified first. The

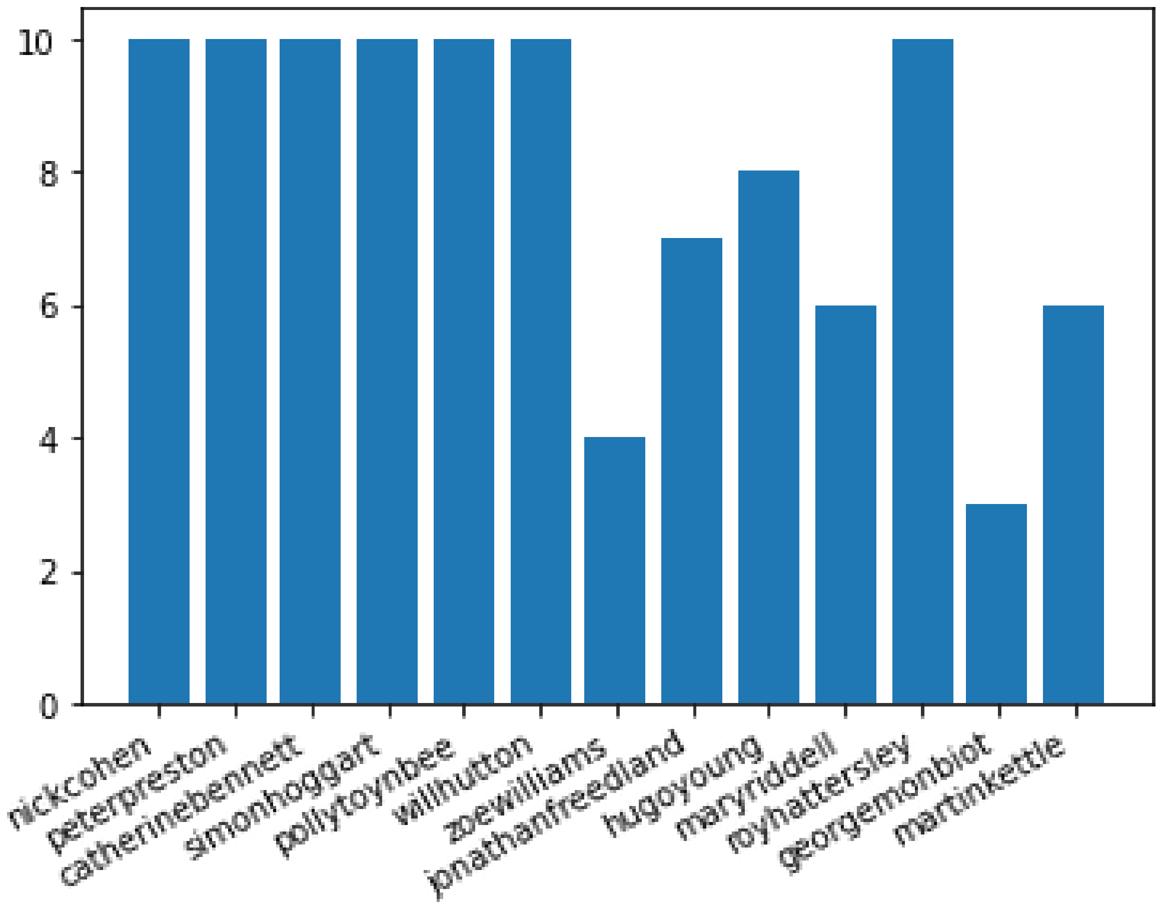

Figure 5.1: The Guardian samples distribution for the Politics topic.

simple and natural way to characterize a document is to consider it as a sequence of tokens grouped into sentences where each token can be one of the three: word, number, punctuation mark. As with the choice of classification method, we initially attempted a naive approach. In fact, many studies have focused on using methods such as TFIDF and Bag Of Words. In fact, our experiment is mainly aimed at showing that simple numerical text representation methods such as TFIDF or BOW, applied with the right hyperparameters, can somehow yield good performance in the same way as other more complex methods such as Doc2Vec or n-grams character selection.

5.2.1 Term Frequency - Inverse Document Frequency & Bag Of Words

Once taken the route of the simple feature extraction approach via TFIDF and BOW, we had to choose the hyperparameters that would play a main role in the model performances for this task. We initially chose a classical approach to the problem, extracting features with both TFIDFVectorizer and CountVectorizer, with standard hyperparameters.

In Code Listing 5.3 we can see the initialization of the vectorizers with the chosen hyperparameters after a long process of tuning and validation.

Code Listing 5.3: TFIDF & BOW Vectorizer.

```
tfidf_vec = TfidfVectorizer(max_df=0.75, max_features=None,
min_df=0.02, use_idf=False, tokenizer=custom_tokenizer,
ngram_range=(1, 4))
counter_vect = CountVectorizer(max_df=0.8, max_features=10000,
min_df=0.02, tokenizer=custom_tokenizer, ngram_range=(1, 2))
```

After that, we shifted our focus to the tokenizer that instead of choosing to use the standard one, we preferred to use a "custom" one. We initially used a robust tokenizer for text categorization tasks, but we realized that for this kind of authorship attribution problem, some approaches valid for many text categorization problems would not work. In fact, in Code Listing 5.4 shows the choice of the three tokenizers we tried experimentally and which sequentially showed better and better results. The first one we tried was a custom tokenizer with classical approaches to text categorization. In fact, we used a "snowball" type stemmer for the English language and applied it to all the filtered words. We also converted all words to lowercase and removed the words from the English stopwords group. This type of tokenizer proved to be the weakest of the three because it removes too many features that best distinguish and characterize a text with respect to the author of the document itself.

Code Listing 5.4: Custom tokenizer for TFIDF and BOW.

```
def tokenize_and_stem(text):
  """
  Below function tokenizes and lemmatizes the texts. It also does
                                  some cleaning by removing non
                                  dictionary words
  This can be used to replace default tokenizer provided by feature
                                  extraction api of sklearn.
  :param text: str
  :return: list
  """
  stemmer = SnowballStemmer("english")
  stop_words = stopwords.words("english")
  tokens = [word.lower() for sent in nltk.sent_tokenize(text) for
                                  word in nltk.word_tokenize(sent)]
  filtered_tokens = []
  for token in tokens:
    if re.search(r'[a-zA-Z-]{4,}', token) and token not in stop_words
                                   and len(wn.synsets(token)) > 0:
      token.strip()
```

```
        filtered_tokens.append(token)
    filtered_tokens = [stemmer.stem(token) for token in filtered_tokens
                                    ]
  return filtered_tokens

  def simple_tokenizer(text):
    text = re.sub('"([^"]*)"', '', text)
    tokens = [word.lower() for sent in nltk.sent_tokenize(text) for
                                    word in nltk.word_tokenize(sent)]
    filtered_tokens = []
    for token in tokens:
      if len(wn.synsets(token)) > 0:
        token.strip()
        filtered_tokens.append(token)
    return filtered_tokens

  def only_remove_quoting_tokenizer(text):
    text = re.sub('"([^"]*)"', '', text)
    tokens = [word.lower() for sent in nltk.sent_tokenize(text) for
                                    word in nltk.word_tokenize(sent)]
    return tokens
```

In fact, the approach of purposely modifying words and removing stopwords, in the literature on authorship attribution has proven to be a wrong one. The most frequent words defined as "non-content" categorize worse than a text in the sense of content and introduce noise, but better classify a text in respect of the author who wrote it, especially in cross domain contexts in which are precisely the content words that go to introduce noise. After evaluating our results with the available datasets, we agreed to change our approach regarding the tokenizer. We tried to build a simpler tokenizer called *"simple_tokenizer"* that would remove only the words between double-quotes because they were considered as phrases or quotation words and therefore would not classify well the text in respect of the author who reported them, after which we only removed those words that, after transforming them into lowercase, were not found as synonyms of an English dictionary (and therefore words that do not conform to the dictionary). This second approach showed better results than the first, but once again we wondered if the approach of removing words that did not conform to the dictionary was a correct approach for author attribution analysis. With these premises in fact, in the third and last approach we thought about removing only the words or phrases contained in quotation marks, therefore without removing the wrong or not present words in the official dictionary of the English language. This last approach, called *"only_remove_quoting_tokenizer"* proved to be the best of the three, thus underlining the importance of stopwords and common mistakes or words commonly used by the author and not present in the official English dictionary, regarding this specific task of authorship attribution. Just for the purpose of making the reader aware, as a tokenizer we tried two additional approaches but they did not show the desired results. Along the lines of thinking that the content words of a text are the ones that litter the numerical representation of a text the most in an authorship attribution context, one of the approaches attempted was text distortion. In fact as also shown in some previous articles [60], using text distortion for autorship attribution tasks, especially in cross domain contexts, could be very effective. The concept behind text distortion is to obfuscate and hide words in a document based on their frequency, so as not to create noise during feature extraction and focus only on the most relevant words. In the case of authorship attribution, the opposite is applied, i.e. less frequent words in a text are obfuscated (i.e. replaced by symbols like $*$ and #) and the same applies to numbers. This approach can be divided into two: an approach that is length-preserving and an approach that particularly shortens the length of the text in order to make feature extraction even easier. In the first case we replace all letters of the selected target word with $*$ or # symbols for numbers, in the second case we replace the selected word with only one $*$ or # symbol for numbers, shortening the resulting text. We applied both of these two approaches as tokenizers of TFIDF and BOW, but the results obtained did not even pass the threshold of mention as they were considered completely unsuccessful.

5.2.2 Grid Search Cross Validation

In almost any Machine Learning project, we train different models on the dataset and selecting the one with the best performance. However, there is almost a room for improvement as we cannot say for sure that this particular model is best for the problem at hand, hence our aim is to improve the model in any way possible. One important factor in the performances of these models are their hyperparameters, once we set appropriate values for these hyperparameters, the performance of a model can improve significantly. At the state of the art, we can say that one of the well-established approaches is to optimize the values of the hyperparameters of a model using GridSearchCV. Note that there is no way to know in advance the best values for hyperparameters so ideally, we need to try all possible values to know the optimal values. Doing this manually could take a considerable amount of time and resources and thus we use GridSearchCV to automate the tuning of hyperparameters. GridSearchCV is a function that comes in Scikit-learn's model_selection package. This function helps to loop through predefined hyperparameters and fit the model on the training set. So, in the end, we can select the best parameters from the listed hyperparameters. As mentioned above, we pass predefined values for hyperparameters to the GridSearchCV function. We do this by defining a dictionary in which we mention a particular hyperparameter along with the values it can take.

Code Listing 5.5: GridSearchCV with BOW, TFIDF and SGD.

```
pipeline = Pipeline([
('vect', CountVectorizer()),
('tfidf', TfidfTransformer()),
('clf', SGDClassifier()),
])

# uncommenting more parameters will give better exploring power but
                                  will
# increase processing time in a combinatorial way
parameters = {
  'vect__max_df': (0.5, 0.75, 1.0),
  'vect__max_features': (None, 5000, 10000, 50000),
  'vect__ngram_range': ((1, 1), (1, 2)),  # unigrams or bigrams
  'tfidf__use_idf': (True, False),
  'tfidf__norm': ('l1', 'l2'),
  'clf__max_iter': (20,),
  'clf__alpha': (0.00001, 0.000001),
  'clf__penalty': ('l2', 'elasticnet'),
  'clf__max_iter': (10, 50, 80,),
}

# find the best parameters for both the feature extraction and the
# classifier
grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1, verbose=1
                                  )

print("Performing grid search...")
print("pipeline:", [name for name, _ in pipeline.steps])
print("parameters:")
print(parameters)
t0 = time.time()
grid_search.fit(dataset['articles'], dataset['author'])
print("done in %0.3fs" % (time.time() - t0))
print()

print("Best score: %0.3f" % grid_search.best_score_)
print("Best parameters set:")
best_parameters = grid_search.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
print("\t%s: %r" % (param_name, best_parameters[param_name]))
```

In Code Listing 5.5 we can see an example of GridSearchCV applied to the datasets with a simple pipeline with: CountVectorizer, TfidfTransformer and SGDClassifier. The pool of parameters we chose were based on previous research on same datasets.

GridSearchCV tries all the combinations of the values passed in the dictionary and evaluates the model for each combination using the Cross-Validation method. Hence after using this function we get accuracy/loss for every combination of hyperparameters and we can choose the one with the best performance. The result of this first attempt at GridSearchCV was shown in Code Listing 5.3, the picture of the final hyperparameter tuning we mentioned earlier in the section.

5.2.3 Vector embeddings of documents

Since we did not want to rely only on some classical feature extraction methods such as TFIDF and BOW mentioned above, recent studies have shown how the use of techniques such as the vector spacing model that transforms document instances into vectors can be successfully applied to tasks such as autorship attribution. To make the reader understand better, the broad idea is to transform each author's documents into vectors of fixed size. These vectors will be "similar" for documents of the same author, so on documents of unknown author we will look for the collection of documents whose representation in vector format is closest to the vector representation of the document of unknown author. We used the Doc2vec [36] method available in the freely downloadable GENSIM module in order to implement our proposal. The implementation of the Doc2vec method requires the following three parameters:

1. the number of features to be returned (length of the vector)
2. the size of the window that captures the neighborhood
3. the minimum frequency of words to be considered into the model

The values of these parameters depend on the used corpus. In a previous work [49] it reported a representation of 300 features, a window size equal to 10 and minimum frequency of 5. In Code Listing 5.6 we can see the implementation of the tagging document algorithm and the computation of the 2 different models for document embedding: Distributed Memory Model and Distributed Bag Of Words Model.

Code Listing 5.6: Doc2Vec for features extraction with gensim python library.

```
from gensim.models.doc2vec import TaggedDocument
import gensim
from tqdm import tqdm
from gensim.models import Doc2Vec

def tag_dataset(df):
    return df.apply(lambda r: TaggedDocument(words=
                                  only_remove_quoting_tokenizer(r['
                                  articles']), tags=[r.author]), axis
                                  =1)

  df_train_tagged = tag_dataset(df_train)
  df_test_tagged = tag_dataset(df_test)

  import multiprocessing

  cores = multiprocessing.cpu_count()
  model_dmm = Doc2Vec(dm=1, dm_mean=1, vector_size=300, window=10,
                                  negative=5, min_count=1, workers=
                                  cores, alpha=0.065, min_alpha=0.065
                                  )
  model_dmm.build_vocab([x for x in tqdm(df_train_tagged.values)])

  model_dbow = Doc2Vec(dm=0, vector_size=300, negative=5, hs=0,
                                  min_count=2, sample = 0, workers=
                                  cores)
  model_dbow.build_vocab([x for x in tqdm(df_train_tagged.values)])

  d2v_model = model_dmm

  from sklearn import utils

  # time
  def train_d2v_model(model, df):
    for epoch in range(30):
      model.train(utils.shuffle([x for x in tqdm(df.values)]),
                                  total_examples=len(df.values),
                                  epochs=1)
      model.alpha -= 0.002
      model.min_alpha = model_dmm.alpha
    model.save(os.path.join(base_dir, 'd2v_{}_model.vec'.format(
                                  PROJECT_NAME)))
```

After the evaluation of both models, we decided to keep only the Distributed Memory Model which resulted in better performance in terms of the score values of the testing set.

5.3 Method selection

At this point, we had to face the problem of deciding the classifier method that would best solve our authorship attribution task. Although in previous studies over the past decades on authorship attribution SVM has been shown to be very convincing ([10], [30],

[70]), we initially wanted to construct an experimental approach that would lead us to exclude the other classifiers for our task.

5.3.1 Manual approach

Our very first naive approach was to compare on different portions of the dataset (increasing number of authors) different classification methods to see which one performed best. Initially, we considered the authorship attribution study for groups of authors consisting of 6 or 10 authors. In truth, as many previous studies show, an authorship attribution model must perform well especially in situations where the group of authors is composed of several dozen candidates. The classifiers initially chosen were:

- Naive Bayes
- Multinomial Naive Bayes
- Logistic Regression
- XGBoost
- XGBoost with Neural Networks
- Random Forest

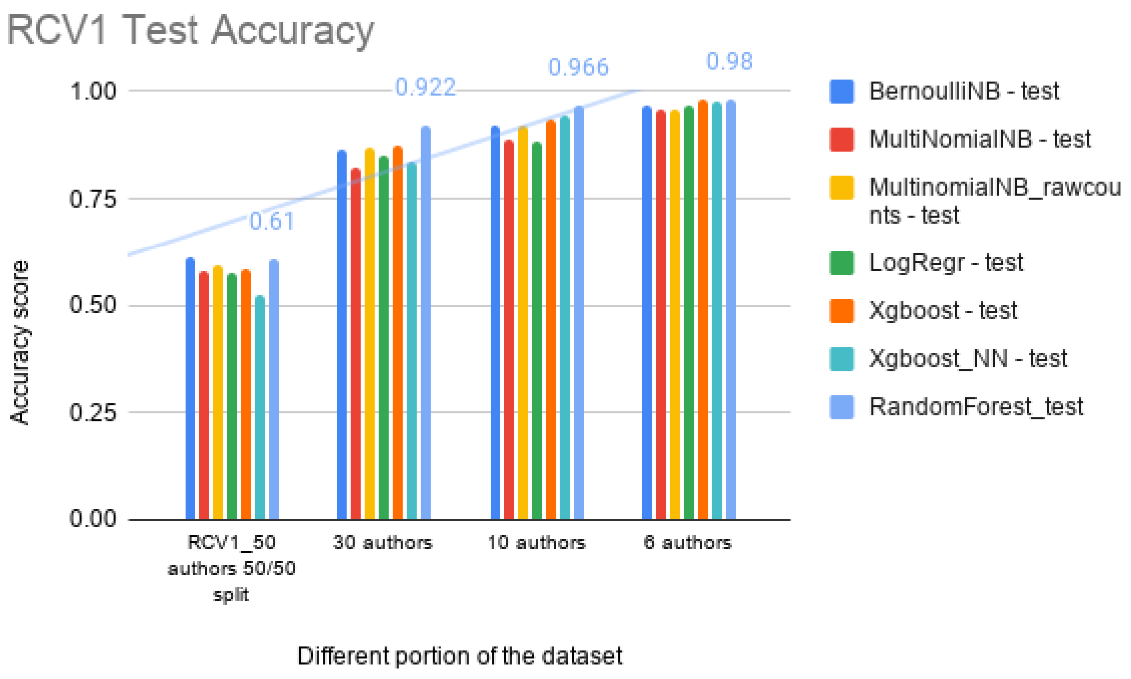

Figure 5.2: Accuracy scores for different groups of authors on Reuters Corpus dataset.

For text representation, we chose to use TFIDF and Bag Of Words, comparing the results depending on the dataset, the number of authors, and the method used. In Figure 5.2 we can see the accuracy score of the testing set of the various classifiers tested on the groups of authors increasing from right to left on the RCV1 dataset.

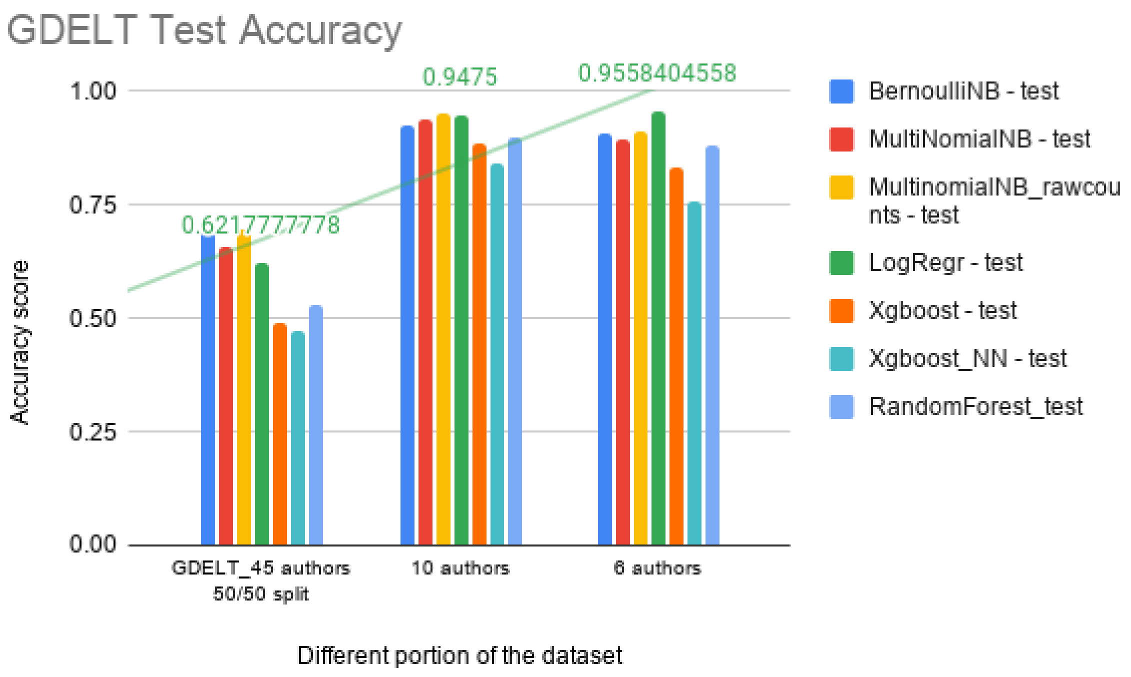

Figure 5.3: Accuracy scores for different groups of authors on GDELT corpus dataset.

As on the groups of “small” authors, i.e. composed of 6 authors and 10 authors, almost all the classifiers exceed the threshold of 95% accuracy that validates the approach even in non-research contexts. The classifier that seems to perform best among the various groups of authors increasing in number is RandomForest. On the other hand, it has been shown that decision tree type classifiers struggle to maintain high performance when the number of features used increases.

In fact in Figure 5.3 and Figure 5.4 we can see that in all 3 single topic datasets the various methods proposed have a decrease in performance when the number of authors increases reaching 50 authors (or 45 in the case of the GDELT dataset). This is probably due to the fact that by keeping the number of documents per author fixed at 50 in the training test (and in the testing set), the number of features to represent grows proportionately as the number of authors increases. Therefore, we need to select a classification method that remains stable as the number of features we want to represent increases, and therefore remains valid for 6, 10, 30, 50 authors (and more).

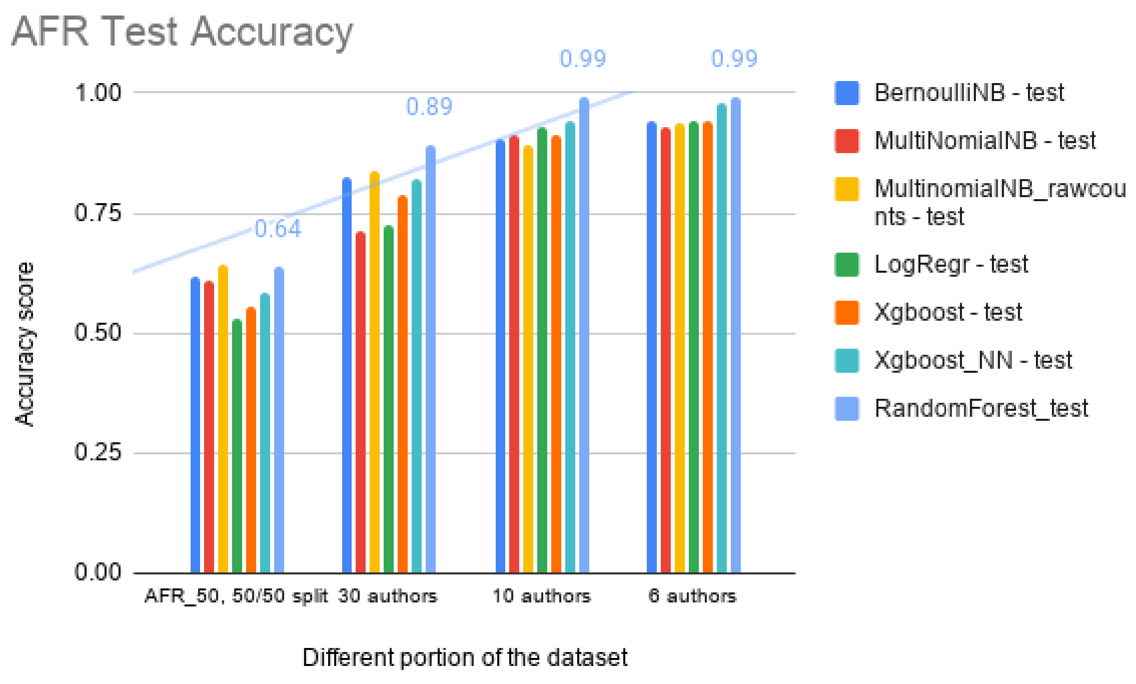

Figure 5.4: Accuracy scores for different groups of authors on Amazon Food Reviews dataset.

5.3.2 Tree-based Pipeline Optimization Tool

Therefore, considering the more or less unsuccessful approaches of the methods presented in the previous section, we tried to find an approach that would validate our choice and those of previous works on the use of the Support Vector Machine as a classification method. Our choice fell on Tree-based Pipeline Optimization Tool (TPOT)[2], an automated machine learning (autoML) tool in Python. In order to give the reader of what TPOT is and how it works, we'll report the first paragraph quoting the TPOT website:

> TPOT is meant to be an assistant that gives you ideas on how to solve a particular machine learning problem by exploring pipeline configurations that you might have never considered, then leaves the fine-tuning to more constrained parameter tuning techniques such as grid search.

So TPOT helps you find good algorithms. TPOT is built on the scikit learn library and follows the scikit learn API closely. It can be used for regression and classification tasks and has special implementations for medical research. TPOT is open source, well documented, and under active development. It's development was spearheaded by researchers at the University of Pennsylvania. TPOT appears to be one of the most popular autoML libraries, with more than 7,800 GitHub stars as of the moment of writing.

[2]`http://epistasislab.github.io/tpot/`

TPOT has what its developers call a genetic search algorithm to find the best parameters and model ensembles. It could also be thought of as a natural selection or evolutionary algorithm. TPOT tries a pipeline, evaluates its performance, and randomly changes parts of the pipeline in search of better performing algorithms (An example is shown in Figure 5.5). This power of TPOT comes from evaluating all kinds of possible pipelines automatically and efficiently. Doing this manually is cumbersome and slower.

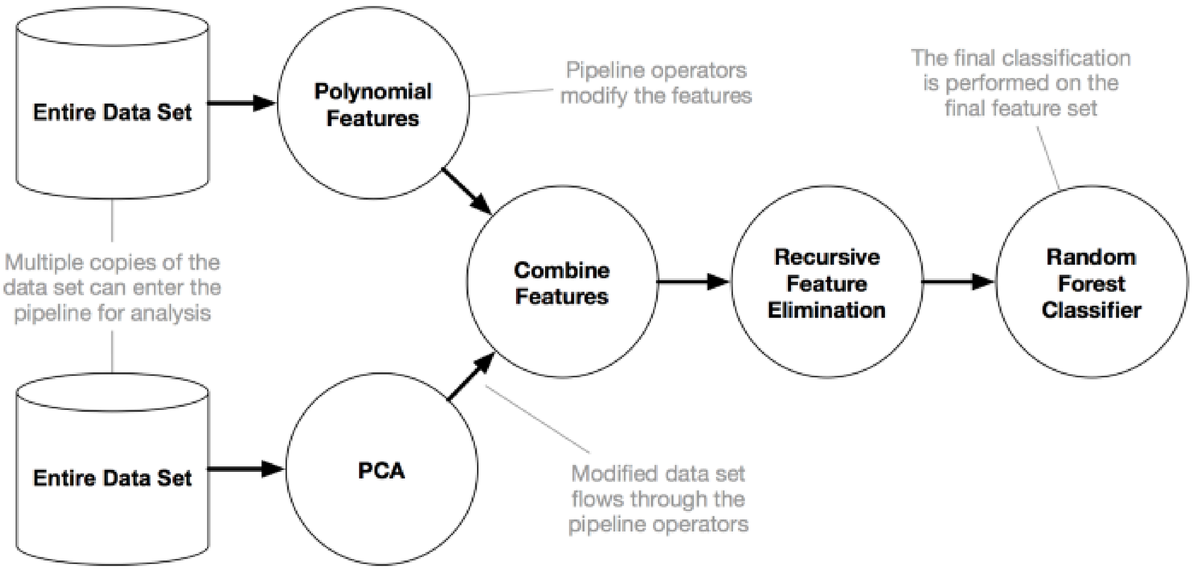
Entire Data Set
Polynomial Features
Pipeline operators modify the features
The final classification is performed on the final feature set
Multiple copies of the data set can enter the pipeline for analysis
Combine Features
Recursive Feature Elimination
Random Forest Classifier
Entire Data Set
PCA
Modified data set flows through the pipeline operators

Figure 5.5: An example TPOT Pipeline from TPOT docs.

In Code Listing A.1 we show a snippet of code we used for extracting TPOT pipeline with hyper parameters and research space.

Code Listing 5.7: TPOT pipeline generation.

```
!pip install -q tpot
from tpot import TPOTClassifier, TPOTRegressor
pipeline_optimizer = TPOTClassifier(generations=5, population_size=20
                                    , cv=5,
random_state=42, verbosity=2, scoring='accuracy', config_dict='TPOT
                                    sparse')
pipeline_optimizer.fit(tfidf_train, df_train['target'])
print(pipeline_optimizer.score(tfidf_test, df_test['target']))
pipeline_optimizer.export('tpot_exported_pipeline.py')
```

We chose the most appropriate hyperparameters and ran TPOT optimization pipelines on all 3 datasets with 50 authors [3]. The result is shown in Code Listing 5.8 for all 3 single domain selected datasets, thus proving that SVM is the best choice as a model classifier for this task.

[3]45 in the case of GDELT

Code Listing 5.8: TPOT pipeline extracted.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC

# NOTE: Make sure that the outcome column is labeled 'target' in the
                                  data file
tpot_data = pd.read_csv('PATH/TO/DATA/FILE', sep='COLUMN_SEPARATOR',
                                  dtype=np.float64)
features = tpot_data.drop('target', axis=1)
training_features, testing_features, training_target, testing_target
                                  = \
train_test_split(features, tpot_data['target'], random_state=42)

# Average CV score on the training set was: 0.6912
exported_pipeline = LinearSVC(C=0.5, dual=True, loss="squared_hinge",
                                   penalty="l2", tol=1e-05)
# Fix random state in exported estimator
if hasattr(exported_pipeline, 'random_state'):
setattr(exported_pipeline, 'random_state', 42)

exported_pipeline.fit(training_features, training_target)
results = exported_pipeline.predict(testing_features)
```

CHAPTER 6

Results and Evaluation

In this chapter we are going to show the best results obtained by setting up the training phase as described in the previous chapters. Most of the parameter tuning was done on just one dataset[1], and then the same setup was used to produce results for all our datasets.

6.1 Confusion matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. To help the reader better understand what this is all about, we can give a practical example with authorship attribution. Suppose we have an unknown document and we have a verification problem where we have to indicate whether Shakespeare is the author of the unknown document or not. The confusion matrix takes the form it has in Figure 6.1 and 4 types of values show up:

- **true positives (TP)**: These are cases in which we predicted yes (Shakespeare is the author of the unknown document), and he really was the author of that document.
- **true negatives (TN)**: We predicted no, and he didn't write that document.
- **false positives (FP)**: We predicted yes, but he didn't actually write that document. (Also known as a "Type I error.")
- **false negatives (FN)**: We predicted no, but in reality he did write that document. (Also known as a "Type II error.")

[1]The Reuters Corpus, RCV1

These aforementioned definitions will come in very handy to better understand all the metrics used to evaluate the model we built.

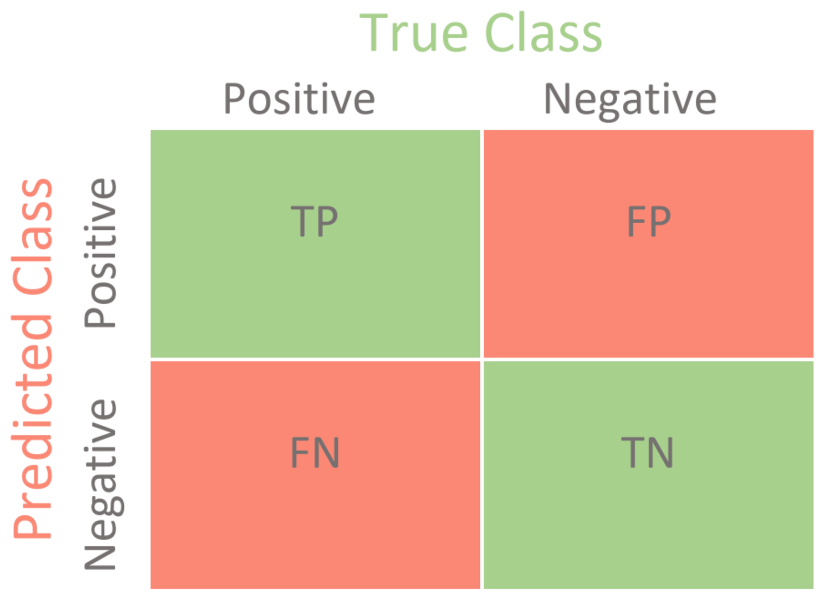

Figure 6.1: A confusion matrix example with 4 classes of values: true positive, true negative, false positive and false negative.

6.2 Metrics used

In order to evaluate the performance of the model being built, we computed for each training phase, 2 different scores on the testing set: the *accuracy score* and the *F1 score*[2]. The accuracy score, one of the more obvious metrics, is the measure of all the correctly identified authors of each documents. It's mostly used when all the classes are equally important.

$$Accuracy = \frac{TruePositive + TrueNegative}{(TruePositive + FalsePositive + TrueNegative + FalseNegative)} \tag{6.1}$$

The F1 score can be interpreted as a weighted average of the precision[3] and recall[4], where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal.
The formula for the F1 score is:

[2]`http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html`
[3]The measure of the correctly identified positive cases from all the predicted positive cases.
[4]The measure of the correctly identified positive cases from all the actual positive cases.

$$F1 = \frac{2 * (precision * recall)}{(precision + recall)} \tag{6.2}$$

where:

- **Precision:** is the fraction of relevant instances among the retrieved instances.
- **Recall:** is the fraction of relevant instances that have been retrieved over the total amount of relevant instances.

To summarise the differences between the F1 and the accuracy:

- Accuracy is used when the True Positives and True negatives are more important while F1 is used when the False Negatives and False Positives are crucial.
- Accuracy can be used when the class distribution is similar while F1 is a better metric when there are imbalanced classes.
- In most real-life classification problems, imbalanced class distribution exists and thus F1 is a better metric to evaluate our model on.

6.3 Results obtained over different datasets

In this section we will present and discuss the best results obtained for the different datasets selected, with particular attention to the type of feature representation, the number of authors, the choice of division between training and testing sets, and whether or not the authors' documents belong to different categories.

6.3.1 Reuters Corpus results

The first results we show refer to the Reuters Corpus as it was the dataset on which we did cross validation and hyperparameters tuning. In fact both the choices of features representation, the parameters and the tokenizer, the classifier parameters were all chosen according to the performance of this corpus, especially in the portion that we called RCV1-50, that is the selection of 100 documents per author, for the 50 most prolific authors of the Reuters corpus, for the CCAT category.
Precisely for this reason in Table 6.1 are shown the 3 best results on this portion of the dataset in terms of both accuracy score and F1. As can be seen from the table, the best results were obtained with the linear SVM classifier, with tfidf as the method of feature representation, and gradually increasing results were obtained thanks to the study of a better selection of words on the tokenizer. In fact we denote *"stock_tokenizer"* the standard tokenizer of the python library *sklearn.feature_extraction.text.TfidfVectorizer*.

While in the other two approaches we changed the tokenizer, setting a "custom" one, at first leaving all words except the text between double quotes. The best result was obtained with a variation of this custom tokenizer, called *"only-remove-quotes-tokenizer"*, removing the words between double quotes but with a length of greater than one word (i.e. the single words between double quotes were left intact) and in addition we used not only the representation of a word ngram, but we also chose to represent the pair of neighboring words to better identify the author of a text. The reasoning behind this type of representation and the reason why we got the best result is because the use of words between double quotes could better distinguish the author of a text, while the sentences enclosed by double quotes were discarded because they certainly belong to a quote, and therefore do not distinguish the style of an author.

Table 6.1: Accuracy score and F1 macro score for Reuters Corpus 50 authors CCAT category.

Model	Accuracy	F1
LinearSVC (combinedDFs), tfidf, stock tokenizer	0.7644	0.7600
LinearSVC (combinedDFs), tfidf, only-remove-quotes-tokenizer	0.7884	0.7842
LinearSVC (combinedDFs), tfidf, only-remove-quotes-tokenizer (threshold 1), ngram=(1,2)	**0.7984**	**0.7949**

In Table 6.2 we can see the results of the same models on the portion of the reuters corpus dataset, of the 100 documents extracted by each author, selecting the 10 most prolific authors with the documents belonging to the CCAT category.

The Reuters dataset being frequently used in other authorship attribution studies, also allows us to compare the results in terms of accuracy with other models that we are aware of. In Table 6.3 we can see some of the models that achieved the best results for the authorship attribution task on the Reuters Corpus both for the RCV1-10 portion and for the RCV1-50 portion. At the end of the table we reported the best result achieved with our method, that for both of the portion of the dataset (i.e. the 10 authors and the 50 authors) showed the highest accuracy score to the best of our knowledge.

For the RCV1-10 portion of the reuters corpus the results of our approach improved the best accuracy score achieved with local histograms method in [55] by $6,71\%$ and for the RCV1-50 we improved by $6,11\%$ the results obtained in [49] with the doc2vec words model.

Table 6.2: Accuracy score and F1 macro score for Reuters Corpus 10 authors CCAT category.

Model	Accuracy	F1
LinearSVC (combinedDFs), tfidf, simple tokenizer	0.8780	0.8760
LinearSVC (combinedDFs), tfidf, only-remove-quotes-tokenizer	0.9080	0.9060
LinearSVC (combinedDFs), tfidf, only-remove-quotes-tokenizer (threshold 1), ngram=(1,2)	**0.9220**	**0.9210**

Table 6.3: Accuracy score for Reuters Corpus 10 and 50 authors in the CCAT category.

Model	RCV1-10	RCV1-50
D2V words[49]	0.8280	0.7524
Local histograms [55]	0.8640	-
Tensor space models [48]	0.8080	-
Character and word n-grams [54]	0.7940	-
N-gram feature selection [22]	-	0.7404
Our approach	**0.9220**	**0.7984**

6.3.2 GDELT Corpus results

Regarding the dataset downloaded from the GDELT project, composed of documents belonging to Victorian era books belonging to 45 authors, the 3 best results obtained are shown in Table 6.4.

Table 6.4: Accuracy score and F1 macro score for GDELT 45 authors.

Model	Accuracy	F1
LinearSVC (combinedDFs), tfidf, stock tokenizer	0.7355	0.7090
LinearSVC (combinedDFs), tfidf, only-remove-quotes-tokenizer	0.7426	0.7173
LinearSVC (combinedDFs), d2v dmm	**0.7716**	**0.7489**

As can be seen from the results, in this only case for this type of dataset the best method for text representation was not tfidf, although the tfidf model along with the "custom" tokenizer of *only-remove-quotes-tokenizer* still performed well. The best method for the representation of documents in vectors, fell on the Distributed Memory Mean (dmm) model that obtained the highest accuracy score. The reason why tfidf didn't get the best result is not clear to us, but we point out that this type of dataset differs from the others for the style of writing (Old English of the mid-nineteenth century), the length of the documents, on average they contain about 1000 characters more than the reuters dataset and about 3000 more than the amazon reviews dataset. In addition, the authors' documents were extracted from books, whose style is very different from the style of news or reviews, where grammatical errors or abbreviations are nil and where quotations are hardly present, except in the form of dialogue of the characters in the story. Unfortunately, to the best of our knowledge we do not have similar previous studies to compare the results obtained on the same type of task and dataset. However, being a similar context (50 vs 45 authors) and similar training methodology (100 documents per author, 50 documents for the training phase and 50 for the testing phase), we can safely say that the results obtained with this type of dataset are lower than those obtained with the same methodology with the Reuters corpus and proved to be the lowest results in terms of accuracy compared to all the datasets that we previously selected for this task.

6.3.3 Amazon Food Reviews Corpus results

The dataset of amazon product review collections in the food category showed the best results in the context of author attribution for a group of 50 authors with 100 documents each, 50 in the training phase and 50 in the testing phase. The best results obtained in terms of accuracy and F1 are shown in Table 6.5.

Table 6.5: Accuracy score and F1 macro score for Amazon Food Reviews 50 authors dataset.

Model	Accuracy	F1
LinearSVC (combinedDFs), tfidf, simple tokenizer	0.7704	0.7674
LinearSVC (combinedDFs), tfidf, stock tokenizer	0.7836	0.7817
LinearSVC (combinedDFs), tfidf, only-remove-quotes-tokenizer, ngram=(1,2)	**0.8388**	**0.8368**

Why our proposed model resulted in better performance on this dataset is not 100% clear to us, but we may give the reader some observation we made after evaluating this results. It is a given and established fact now that the length of the text in a task such as authorship attribution is crucial for the successful authorship attribution of an anonymous text. Yet the average length of this dataset is much shorter than that of the other datasets compared; the average length of a review (i.e. an author's document in this dataset) is in fact about 66% shorter than a news document in the Reuters corpus and about 75% shorter than a text in the GDELT corpus. The lexicon of the reviews in this corpus is more mundane, including abbreviations, punctuation, grammatical errors, and short, disconnected periods. Probably because of the characteristic of being short texts, the use of one or two common keywords across product reviews means that the author is better recognized than in other contexts. Also for this dataset, to the best of our knowledge, there are no studies on authorship attribution that would allow us to compare the metrics of our approach in terms of performance.

6.3.4 The Guardian Corpus results

Although our study primarily focused on authorship attribution on single-category documents, we wanted to collect one of the datasets widely used in previous studies to compare how our approach performs on cross-topic datasets compared to single-topic datasets and how much better or worse it performs compared to models from previous studies specifically designed to address the problem of authorship attribution in the context of cross-topic documents. The dataset in question is selected from The Guardian newspaper. The context is very different from previous ones for the number of authors in the entire dataset (13) and the selection of author papers (that was described in subsection 5.1.4). We followed this approach by using the authors' papers in the Politics category as training and then performed testing on 4 portions of the testing set with the authors' papers belonging to the Books, World, Uk, and Society categories.

Table 6.6: Accuracy score and F1 macro score for The Guardian Corpus with LinearSVC (combinedDFs), tfidf, only-remove-quotes-tokenizer.

Training topic vs Test topic	Accuracy	F1
Politics vs Books	0.7446	0.7640
Politics vs World	0.7560	0.7470
Politics vs Uk	0.7890	0.7010
Politics vs Society	0.8863	0.7430
Average	**0.7940**	**0.7388**

In the Table 6.6 we can see the results obtained with the linearSVC(combinedDFs) model, with feature extraction method tfidf and with the "custom" tokenizer *only-remove-quotes-tokenizer*. In the last row we can see the calculated average accuracy score and F1 obtained. Taking the work of [49] as a benchmark for this dataset, we can state that the results obtained with our approach in the context of cross-topic authorship attribution did not come close to the benchmark, thus demonstrating that the two types of authorship attribution tasks probably need different approaches in order to obtain performances adequate to the case study and the dataset with which they are to be compared.

In Figure 6.2 we can see the metrics evaluated for each dataset on the best scoring models. The metrics shown are: accuracy, precision macro, recall macro, F1 macro. In blue we can see the result of the reuters dataset on the portion composed of 50 authors, while in yellow the best result obtained on the dataset of amazon food reviews.

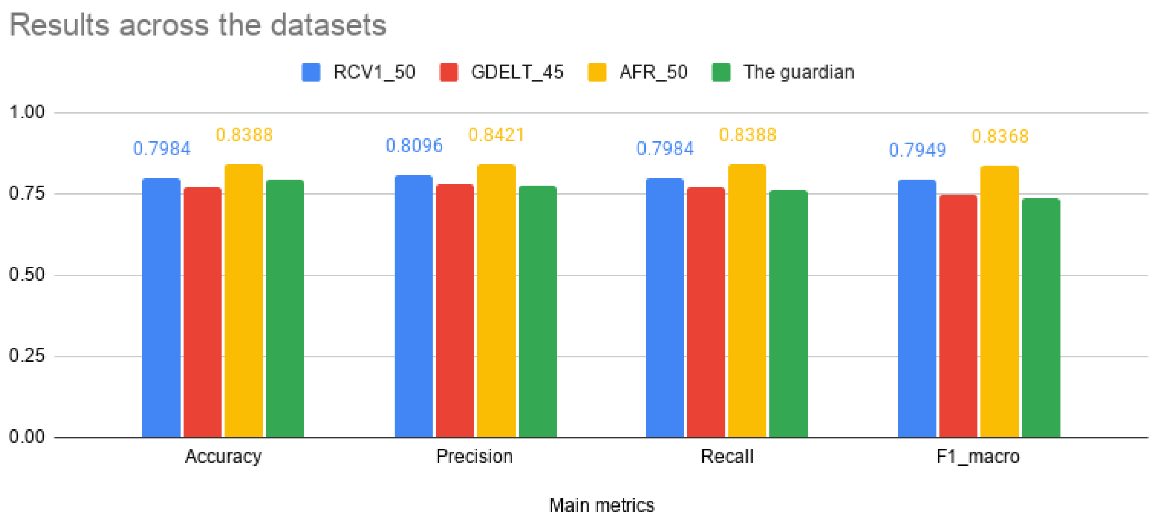

Figure 6.2: Accuracy, precision, recall and F1 for every dataset showing only the best result achieved for each one.

CHAPTER 7

Conclusion

In this thesis work we have given a general overview of what authorship attribution is and how it can be addressed with acceptable results nowadays. We first saw the various types of tasks that can be addressed: instance-based versus profile-based approach. We also saw the differences of single domain documents or previous work on cross domain datasets. We also saw the difference between closed and open set of authors. We listed the main information retrieval techniques for representing and extracting useful text information, including tf-idf and bag of words, as well as doc2vec. In Chapter 4 we then listed in detail the state of the art work on authorship attribution and the corpora used, as well as the main classifiers. We have seen how SVM is widely used among machine learning models for some of its peculiar features (including the speed of training and the scalability as features increase). We also showed how Reuters Corpus and The Guardian datasets are the most used datasets for autorship attribution work. In chapter 5 we have described the work we have done starting from the preparation of the dataset, the selection and extraction of features and finally the choice of the model using an automatic optimization tool (TPOT). In the last chapter 6 we showed the best metrics obtained with our approach, comparing them also with results obtained reproducing the same scenario of related works. We showed how for the RCV1-10 dataset we improved to the best of our knowledge by 6.71% the accuracy of the model compared to the best result obtained with using local histograms. We also improved the RCV1-50 dataset by 6.11% over the best result obtained using a doc2vec word model. We then also reported results for 4 total datasets, including a dataset of Victorian-era books collected by the GDELT project and the amazon food reviews dataset, which showed the best results in terms of accuracy among the top 3 datasets that are united by number of authors and number of documents in training and testing. We then also showed how our approach did not show significant improvements on The Guardian corpus dataset, thus proving suboptimal cross domain and cross topic performance.

In this work we mainly focused our efforts towards author attribution in its most straightforward form, i.e. we are given examples of the writing of a number of candidate authors and are asked to determine which of them authored a given anonymous text [32]. This approach to author attribution has been the only one studied until the last decade, as it is already quite complex. The enormous steps forward both in terms of modern computing power and in terms of studies of new models of machine learning, have allowed us to leave the concept of classical attribution and explore some of the tasks still unsolved.

The open set authorship attribution One of the closest problems to being solved in this field of authorship attribution is undoubtedly moving from a closed set group of authors to an open set group of authors. This revolution in approach allows us to have a group of authors in the supervised training phase and a classifier who must take into account that in the testing phase it may have to deal with labels (i.e. authors in this case) that it has never seen in the training phase and classify them as unknown. This approach is complex because it puts together similarities between the verification problem and the needle-in-haystack problem with the classic approach of author attribution in a closed set. During the study of our work we have tried to validate our approach by dealing also with a type of open set of author attribution. In fact, we removed 10% of the authors from each dataset in the training phase and moved them to the testing phase. To give the reader a better understanding we show a practical example: for the dataset RCV1-50 with 50 authors, for each of them we collected 100 documents. In the closed set authorship attribution approach we would have had a 50/50 split, i.e. 50 documents of each author would have gone in the training phase and 50 documents of each author in the testing phase, thus leading to a balanced splitting between the classes. In the open set authorship attribution approach, we selected 5 of the 50 authors whose 100% of the documents (i.e., 100 documents) we placed in the testing phase by moving them out of the training phase. We thus obtained an unbalanced splitting of classes with a total of 50 documents for 45 authors in the training phase (a total of 2250 documents) and 50 documents for 45 authors, adding 100 documents for each of the 5 remaining authors in the testing phase, ending up with a total of 2750 documents in the testing phase. For the open set authorship attribution approach we used a One Vs Rest classifier from the python library $sklearn.multiclass.OneVsRestClassifier$ i.e. we built a classifier for each author, giving his 50 documents as positive examples and all other documents belonging to the other authors as negative examples. We then inserted an additional label in the training phase, marking it as "unknown author" for the testing phase. The results obtained with the datasets selected in this work are shown in Figure 7.1. We excluded the dataset from The Guardian Corpus as the number of authors in the full dataset was too small to allow for reliable results without introducing learning bias.

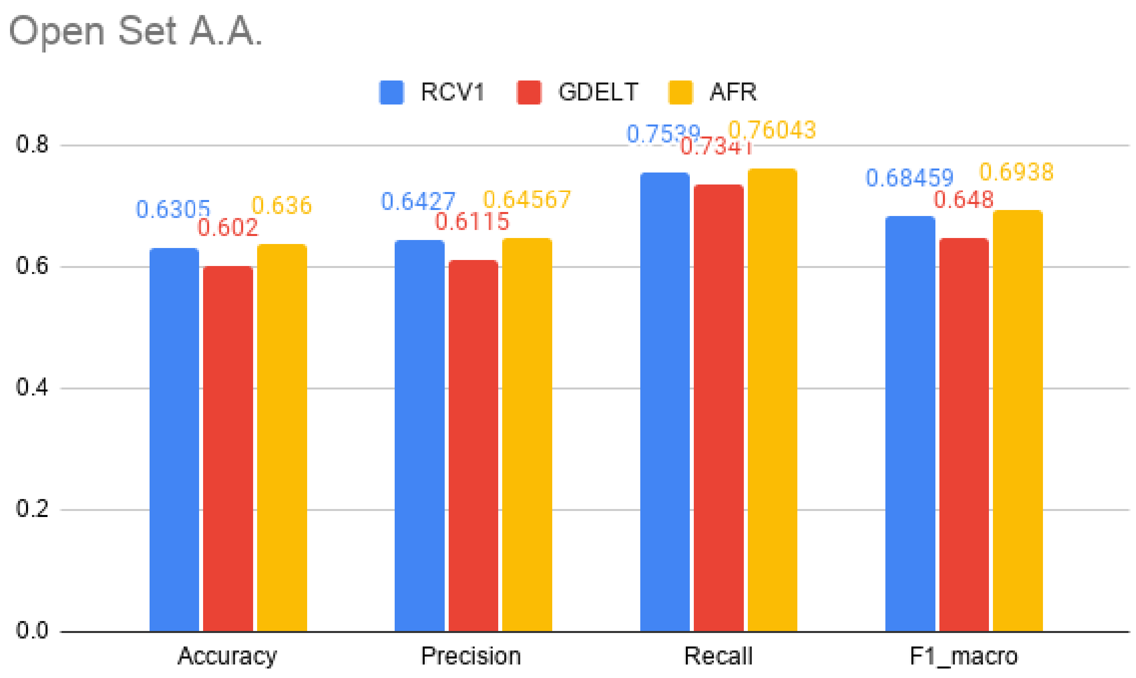

Figure 7.1: Accuracy, precision, recall and f1-macro for Reuters Corpus, GDELT corpus and Amazon Food Reviews corpus for the open set authorship attribution task (10% of unknown authors).

As we can see from the results obtained, there is a lot of room for improvement in terms of accuracy. In fact, we obtained results up to almost 25% lower than the values obtained for the datasets considering the closed set authorship attribution problem. These considerations made us mainly focus on the classical approach, but they give us the idea that many studies could come out on this particular type of authorship attribution subtask in the next years, as we have not yet found a valid approach that works for all types of datasets and for different authors set size.

Bibliography

[1] Ahmed Abbasi and Hsinchun Chen. Applying authorship analysis to extremist-group web forum messages. *IEEE Intelligent Systems*, 20(5):67–75, 2005.

[2] Shlomo Argamon and Shlomo Levitan. Measuring the usefulness of function words for authorship attribution. In *Proceedings of the 2005 ACH/ALLC Conference*, pages 4–7, 2005.

[3] Harald Baayen, Hans van Halteren, Anneke Neijt, and Fiona Tweedie. An experiment in authorship attribution. In *6th JADT*, volume 1, pages 69–75. Citeseer, 2002.

[4] Sarkhan Badirli, Mary Borgo Ton, Abdulmecit Gungor, and Murat Dundar. Open set authorship attribution toward demystifying victorian periodicals. *arXiv preprint arXiv:1912.08259*, 2019.

[5] John Burrows. 'delta': a measure of stylistic difference and a guide to likely authorship. *Literary and linguistic computing*, 17(3):267–287, 2002.

[6] JK Chambers, P Trudgill, and Natalie Schilling-Estes. The handbook of language variation and change (2nd). *Victoria: Blackwell Publishing*, 2004.

[7] Carole E Chaski. Who's at the keyboard? authorship attribution in digital evidence investigations. *International journal of digital evidence*, 4(1):1–13, 2005.

[8] Na Cheng, Rajarathnam Chandramouli, and KP Subbalakshmi. Author gender identification from text. *Digital Investigation*, 8(1):78–88, 2011.

[9] Cindy Chung and James W Pennebaker. The psychological functions of function words. *Social communication*, 1:343–359, 2007.

[10] Joachim Diederich, Jörg Kindermann, Edda Leopold, and Gerhard Paass. Authorship attribution with support vector machines. *Applied intelligence*, 19(1):109–123, 2003.

[11] Harris Drucker, Donghui Wu, and Vladimir N Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural networks*, 10(5):1048–1054, 1999.

[12] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on Information and knowledge management*, pages 148–155, 1998.

[13] Neal Fox, Omran Ehmoda, and Eugene Charniak. Statistical stylometrics and the marlowe-shakespeare authorship debate. *Proceedings of the Georgetown University Roundtable on Language and Linguistics (GURT), Washington, DC, USA*, 2012.

[14] Georgia Frantzeskou, Efstathios Stamatatos, Stefanos Gritzalis, and Sokratis Katsikas. Effective identification of source code authors using byte-level information. In *Proceedings of the 28th international conference on Software engineering*, pages 893–896, 2006.

[15] Helena Gómez-Adorno, Juan-Pablo Posadas-Durán, Grigori Sidorov, and David Pinto. Document embeddings learned on various types of n-grams for cross-topic authorship attribution. *Computing*, 100(7):741–756, 2018.

[16] Jack Grieve. Quantitative authorship attribution: An evaluation of techniques. *Literary and linguistic computing*, 22(3):251–270, 2007.

[17] Abdulmecit Gungor. *Benchmarking authorship attribution techniques using over a thousand books by fifty Victorian era novelists.* PhD thesis, 2018.

[18] H van Halteren. Linguistic profiling for authorship recognition and verification. 2004.

[19] Graeme Hirst and Ol'ga Feiguina. Bigrams of syntactic labels for authorship discrimination of short texts. *Literary and Linguistic Computing*, 22(4):405–417, 2007.

[20] David I Holmes. The evolution of stylometry in humanities scholarship. *Literary and linguistic computing*, 13(3):111–117, 1998.

[21] David I Holmes and Fiona J Tweedie. Forensic stylometry: A review of the cusum controversy. *Revue Informatique et Statistique dans les Sciences Humaines*, 31(1): 19–47, 1995.

[22] John Houvardas and Efstathios Stamatatos. N-gram feature selection for authorship identification. In *International conference on artificial intelligence: Methodology, systems, and applications*, pages 77–86. Springer, 2006.

[23] Thorsten Joachims. Making large-scale svm learning practical. Technical report, Technical report, 1998.

[24] Thorsten Joachims et al. Transductive inference for text classification using support vector machines. In *Icml*, volume 99, pages 200–209, 1999.

[25] Patrick Juola. *Authorship attribution*, volume 3. Now Publishers Inc, 2008.

[26] Brett Kessler, Geoffrey Nunberg, and Hinrich Schütze. Automatic detection of text genre. *arXiv preprint cmp-lg/9707002*, 1997.

[27] Mike Kestemont, Efstathios Stamatatos, Enrique Manjavacas, Walter Daelemans, Martin Potthast, and Benno Stein. Overview of the cross-domain authorship attribution task at pan 2019. In *CLEF (Working Notes)*, 2019.

[28] Dmitry V Khmelev and William J Teahan. A repetition based measure for verification of text collections and for text categorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 104–110, 2003.

[29] Moshe Koppel and Jonathan Schler. Exploiting stylistic idiosyncrasies for authorship attribution. In *Proceedings of IJCAI'03 Workshop on Computational Approaches to Style Analysis and Synthesis*, volume 69, pages 72–80, 2003.

[30] Moshe Koppel, Jonathan Schler, and Kfir Zigdon. Determining an author's native language by mining a text for errors. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 624–628, 2005.

[31] Moshe Koppel, Jonathan Schler, Shlomo Argamon, and Eran Messeri. Authorship attribution with thousands of candidate authors. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 659–660, 2006.

[32] Moshe Koppel, Jonathan Schler, and Shlomo Argamon. Computational methods in authorship attribution. *Journal of the American Society for information Science and Technology*, 60(1):9–26, 2009.

[33] Moshe Koppel, Jonathan Schler, and Shlomo Argamon. Authorship attribution in the wild. *Language Resources and Evaluation*, 45(1):83–94, 2011.

[34] Moshe Koppel, Jonathan Schler, Shlomo Argamon, and Yaron Winter. The "fundamental problem" of authorship attribution. *English Studies*, 93(3):284–291, 2012.

[35] Robert Layton, Paul Watters, and Richard Dazeley. Authorship attribution for twitter in 140 characters or less. In *2010 Second Cybercrime and Trustworthy Computing Workshop*, pages 1–8. IEEE, 2010.

[36] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[37] David Madigan, Alexander Genkin, David D Lewis, Shlomo Argamon, Dmitriy Fradkin, and Li Ye. Author identification on the large scale. In *Proceedings of the 2005 Meeting of the Classification Society of North America (CSNA)*, 2005.

[38] Ilia Markov, Efstathios Stamatatos, and Grigori Sidorov. Improving cross-topic authorship attribution: The role of pre-processing. In *International Conference on Computational Linguistics and Intelligent Text Processing*, pages 289–302. Springer, 2017.

[39] Yuval Marton, Ning Wu, and Lisa Hellerstein. On compression-based text classification. In *European Conference on Information Retrieval*, pages 300–314. Springer, 2005.

[40] S Michaelson and A Morton. The qsum plot. Technical report, Internal Report CSR-3, 1990.

[41] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[42] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[43] Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pages 746–751, 2013.

[44] Leonid A Mironovsky, Alexander V Nikitin, Nina N Reshetnikova, and Nikolay V Soloviev. Graphological analysis and identification of handwritten texts. In *Computer Vision in Control Systems-4*, pages 11–40. Springer, 2018.

[45] Tom M Mitchell. Artificial neural networks. *Machine learning*, 45:81–127, 1997.

[46] Frederick Mosteller and David L Wallace. *Inference and disputed authorship: The Federalist*. Stanford Univ Center for the Study, 2007.

[47] Rebekah Overdorf and Rachel Greenstadt. Blogs, twitter feeds, and reddit comments: Cross-domain authorship attribution. *Proceedings on Privacy Enhancing Technologies*, 2016(3):155–171, 2016.

[48] Spyridon Plakias and Efstathios Stamatatos. Tensor space models for authorship identification. In *Hellenic Conference on Artificial Intelligence*, pages 239–249. Springer, 2008.

[49] Juan-Pablo Posadas-Durán, Helena Gómez-Adorno, Grigori Sidorov, Ildar Batyrshin, David Pinto, and Liliana Chanona-Hernández. Application of the distributed document representation in the authorship attribution task for small corpora. *Soft Computing*, 21(3):627–639, 2017.

[50] Martin Potthast, Sarah Braun, Tolga Buz, Fabian Duffhauss, Florian Friedrich, Jörg Marvin Gülzow, Jakob Köhler, Winfried Lötzsch, Fabian Müller, Maike Elisa Müller, et al. Who wrote the web? revisiting influential author identification research applicable to information retrieval. In *European Conference on Information Retrieval*, pages 393–407. Springer, 2016.

[51] Joseph Rudman. The state of authorship attribution studies: Some problems and solutions. *Computers and the Humanities*, 31(4):351–365, 1997.

[52] Conrad Sanderson and Simon Guenter. Short text authorship attribution via sequence kernels, markov chains and author unmasking: An investigation. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, pages 482–491, 2006.

[53] Upendra Sapkota, Thamar Solorio, Manuel Montes, Steven Bethard, and Paolo Rosso. Cross-topic authorship attribution: Will out-of-topic data help? In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 1228–1237, 2014.

[54] Upendra Sapkota, Steven Bethard, Manuel Montes, and Thamar Solorio. Not all character n-grams are created equal: A study in authorship attribution. In *Proceedings of the 2015 conference of the North American chapter of the association for computational linguistics: Human language technologies*, pages 93–102, 2015.

[55] Yunita Sari, Andreas Vlachos, and Mark Stevenson. Continuous n-gram representations for authorship attribution. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 267–273, 2017.

[56] Roy Schwartz, Oren Tsur, Ari Rappoport, and Moshe Koppel. Authorship attribution of micro-messages. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1880–1891, 2013.

[57] Efstathios Stamatatos. Author identification: Using text sampling to handle the class imbalance problem. *Information Processing & Management*, 44(2):790–799, 2008.

[58] Efstathios Stamatatos. A survey of modern authorship attribution methods. *Journal of the American Society for information Science and Technology*, 60(3):538–556, 2009.

[59] Efstathios Stamatatos. On the robustness of authorship attribution based on character n-gram features. *Journal of Law and Policy*, 21(2):421–439, 2013.

[60] Efstathios Stamatatos. Authorship attribution using text distortion. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1138–1149, 2017.

[61] Sean Stanko, Devin Lu, and Irving Hsu. Whose book is it anyway? using machine learning to identify the author of unknown texts. *Machine Learning Final Projects*, 2013.

[62] Andrew Tausz. Predicting the date of authorship of historical texts. *CS224N project*, 2011.

[63] Antônio Theóphilo, Luís AM Pereira, and Anderson Rocha. A needle in a haystack? harnessing onomatopoeia and user-specific stylometrics for authorship attribution of micro-messages. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2692–2696. IEEE, 2019.

[64] Chris van der Lee and Antal van den Bosch. Exploring lexical and syntactic features for language variety identification. In *Proceedings of the Fourth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial)*, pages 190–199, 2017.

[65] Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[66] Carrington B Williams. Mendenhall's studies of word-length distribution in the works of shakespeare and bacon. *Biometrika*, 62(1):207–212, 1975.

[67] Lili Yang, Chunping Li, Qiang Ding, and Li Li. Combining lexical and semantic features for short text classification. *Procedia Computer Science*, 22:78–86, 2013.

[68] G Udny Yule. A test of tippett's random sampling numbers. *Journal of the Royal Statistical Society*, 101(1):167–172, 1938.

[69] Ying Zhao and Justin Zobel. Effective and scalable authorship attribution using function words. In *Asia Information Retrieval Symposium*, pages 174–189. Springer, 2005.

[70] Rong Zheng, Jiexun Li, Hsinchun Chen, and Zan Huang. A framework for authorship identification of online messages: Writing-style features and classification techniques. *Journal of the American society for information science and technology*, 57(3): 378–393, 2006.

[71] George Kingsley Zipf. Selected studies of the principle of relative frequency in language. 1932.

[72] Sven Meyer Zu Eissen, Benno Stein, and Marion Kulig. Plagiarism detection without reference collections. In *Advances in data analysis*, pages 359–366. Springer, 2007.

Ringraziamenti

Se 10 mesi fa mi avessero detto che oggi mi sarei laureato, non ci avrei mai creduto. Avevo completato il 40% del mio percorso universitario ed ero in bilico tra una pandemia mondiale ed un calo di motivazione personale. Se oggi scrivo questa ultima pagina della mia tesi, lo devo ad alcune persone in particolare:

Ringrazio il mio relatore, per aver creduto in me anche quando non lo facevo più nemmeno io.

Ringrazio Flavio *per l'enorme pazienza e la disponibilità durante gli ultimi mesi.*

Un ringraziamento speciale ai miei genitori, che non hanno mai smesso di credere in me e nel credere che ce la potessi comunque fare nonostante tutto.

Ringrazio il mio amico, socio e compagno di studi Alberto, *anche se sicuramente un "grazie" non può essere abbastanza. E' stato fonte di motivazione durante gli ultimi mesi e sempre pronto a farmi credere possibile questo incredibile traguardo, considerando da quanto lontano provenivamo e quanto poco tempo avessimo per concluderlo.*

Ringrazio anche Pietro *per il sostegno, i consigli e l'esperienza che ci ha permesso di superare più velocemente gli ultimi esami.*

Ringrazio Ossama *e* Luca*, amici, colleghi e soci nel mondo imprenditoriale. E' bello poter contare su di voi ed è incredibilmente appagante potersi confrontare quotidianamente su problemi e soluzioni, creando un network solido di conoscenze.*

Ringrazio Beatrice, Marco *e* Claudio*: questa tesi non vi piacerà, poichè è stata la causa del nostro vederci di rado negli ultimi mesi. Grazie per il sostegno e per avermi sempre stimolato e fatto ri-credere ogni giorno che passava nell'importanza di questo traguardo.*

Ringrazio Rebecca*, per avermi supportato (e sopportato) "no matter what", per aver accettato di rileggere queste 100 pagine tecniche, per gli ultimi "odiati" weekend passati davanti al pc e per le vacanze in giro per l'Italia (quando ancora si poteva).*

Ringrazio Billo *per esser stato la miglior distrazione che potessi avere negli ultimi 4 mesi.*

Infine ringrazio il corona virus, dapprima accolto come una catastrofe ma poi sempre di più come segno che anche quando alcune porte si chiudono... beh, altre si aprono!

APPENDIX A

CODE

All the datasets, the notebooks and the results obtained are available at this public link: https://drive.google.com/drive/folders/0AM8tHyzDct5GUk9PVA

A.1 Closed set authorship attribution code

In Code Listing A.1 we can see how we built our TPOT pipeline to get the classifier model we did use later on in the training process.

Code Listing A.1: TPOT pipeline generation.

```
!pip install -q tpot
from tpot import TPOTClassifier, TPOTRegressor
pipeline_optimizer = TPOTClassifier(generations=5, population_size=20
                                    , cv=5,
random_state=42, verbosity=2, scoring='accuracy', config_dict='TPOT
                                    sparse')
pipeline_optimizer.fit(tfidf_train, df_train['target'])
print(pipeline_optimizer.score(tfidf_test, df_test['target']))
pipeline_optimizer.export('tpot_exported_pipeline.py')
```

Code Listing A.2 shows how we adapted doc2vec method to our task and the hyperparameter used when training doc2vec model. In the last lines of code of Code Listing A.2 we can see how to extract features vectors from the doc2vec model and the documents.

Code Listing A.2: D2V training and features extraction.

```
from gensim.models.doc2vec import TaggedDocument
import gensim
from tqdm import tqdm
from gensim.models import Doc2Vec
```

```
def tag_dataset(df):
return df.apply(lambda r: TaggedDocument(words=
                                  only_remove_quoting_tokenizer(r['
                                  articles']), tags=[r.author]), axis
                                  =1)

df_train_tagged = tag_dataset(df_train)
df_test_tagged = tag_dataset(df_test)

import multiprocessing

cores = multiprocessing.cpu_count()
print(cores)
model_dmm = Doc2Vec(dm=1, dm_mean=1, vector_size=300, window=10,
                                  negative=5, min_count=1, workers=
                                  cores, alpha=0.065, min_alpha=0.065
                                  )
model_dmm.build_vocab([x for x in tqdm(df_train_tagged.values)])

model_dbow = Doc2Vec(dm=0, vector_size=300, negative=5, hs=0,
                                  min_count=2, sample = 0, workers=
                                  cores)
model_dbow.build_vocab([x for x in tqdm(df_train_tagged.values)])

d2v_model = model_dmm

from sklearn import utils

# time
def train_d2v_model(model, df):
  for epoch in range(30):
    model.train(utils.shuffle([x for x in tqdm(df.values)]),
                                  total_examples=len(df.values),
                                  epochs=1)
    model.alpha -= 0.002
    model.min_alpha = model_dmm.alpha
  model.save(os.path.join(base_dir, 'd2v_{}_model.vec'.format(
                                  PROJECT_NAME)))

def vec_for_learning(model, tagged_docs):
  sents = tagged_docs.values
  targets, regressors = zip(*[(doc.tags[0], model.infer_vector(doc.
                                  words, steps=20)) for doc in sents]
                                  )
  return targets, regressors
training_target, training_features = vec_for_learning(d2v_model,
                                              df_train_tagged)
testing_target, testing_features = vec_for_learning(d2v_model,
                                              df_test_tagged)
```

In Code Listing A.3 we reported the code use to train the support vector machine with the features vectors chosen and how we evaluate the main metrics shown in chapter 6.

Code Listing A.3: Training SVM model and retrieve score metrics.

```
def double_pipeline():
  exported_pipeline = make_pipeline(
  make_union(
  FunctionTransformer(copy),
  SelectFwe(score_func=f_classif, alpha=0.004)
  ),
  LinearSVC(C=10.0, dual=True, loss="squared_hinge", penalty="l2",
                                              tol=0.000001, max_iter=10)
  )
  set_param_recursive(exported_pipeline.steps, 'random_state', 42)
  return exported_pipeline

def single_pipeline():
  exported_pipeline = LinearSVC(C=20.0, dual=True, loss="hinge",
                                              penalty="l2", tol=0.0001)
  if hasattr(exported_pipeline, 'random_state'):
    setattr(exported_pipeline, 'random_state', 42)
  return exported_pipeline

exported_pipeline.fit(training_features, training_target)
predicted = exported_pipeline.predict(testing_features)

accuracy_result = accuracy_score(testing_target, predicted)
precision_result = precision_score(testing_target, predicted, average
                                              ='macro')
recall_result = recall_score(testing_target, predicted, average='
                                              macro')
f1_result = f1_score(testing_target, predicted, average='macro')
print(f"Accuracy: {accuracy_result}\nPrecision: {precision_result}\
                                              nRecall: {recall_result}\nF1_macro:
                                               {f1_result}")
```

Finally, Code Listing A.4 shows the piece of code we used to cross validate our model just trained with *StratifiedShuffleSplit*. We chose this type of split when cross validating because we wanted to end up with the same number of samples in the training set and in the testing set overall and also for each authors.

Code Listing A.4: Cross validation for balanced class.

```
from sklearn.model_selection import StratifiedKFold, KFold,
                                     StratifiedShuffleSplit
import numpy as np

skf = StratifiedShuffleSplit(n_splits=10, test_size=0.5, random_state
                                     =42)
for train, test in skf.split(X, y):
  print('train -  {}   |   test -  {}'.format(
  np.bincount(y[train]), np.bincount(y[test])))


# evaluate model
scores = cross_val_score(exported_pipeline, X, y, scoring='accuracy',
                                      cv=skf, n_jobs=-1)

# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

A.2 Open set authorship attribution code

In Code Listing A.5 we can see the code for splitting the training set from the testing set documents in the open set authorship attribution scenario.

Code Listing A.5: Training and Testing set split for open set autorship attribution.

```
# Even split 50 & 50 per author and document
df_train = dataset.groupby('author').head(N_DOCS/2).reset_index(drop=
                                     True)
if OPEN_SET:
  df_train = df_train[df_train.groupby('author').ngroup() < (
                                     NUM_AUTHORS-OPEN_SET_NUM_AUTHORS)]
print_stats_dataset(df_train)
df_train.head()

# get difference between dataset and df_train for df_test
df_test = pd.concat([dataset,df_train]).drop_duplicates(keep=False)
print_stats_dataset(df_test)
df_test.head()
```

List of Figures

List of Tables