UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

AUTHORSHIP ATTRIBUTION OF ANONYMOUS TEXTS USING MACHINE LEARNING

Diploma Thesis

Mavromati Galateia

Supervisor: Vassilakopoulos Michael

Volos 2021

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

AUTHORSHIP ATTRIBUTION OF ANONYMOUS TEXTS USING MACHINE LEARNING

Diploma Thesis

Mavromati Galateia

Supervisor: Vassilakopoulos Michael

Volos 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΑΠΟΔΟΣΗ ΣΥΓΓΡΑΦΕΑ ΑΝΩΝΥΜΩΝ ΚΕΙΜΕΝΩΝ ΜΕ ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

Διπλωματική Εργασία

Μαυρομάτη Γαλάτεια

Επιβλέπων: Βασιλακόπουλος Μιχαήλ

Βόλος 2021

Approved by the Examination Committee:

Supervisor **Vassilakopoulos Michael**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Tsompanopoulou Panagiota**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Daskalopoulou Aspasia**
Assistant Professor, Department of Electrical and Computer Engineering, University of Thessaly

Date of approval: 30-6-2021

Acknowledgements

I wish to show my appreciation to my professors Michael Vassilakopoulos and Elias Houstis, for introducing the world of machine learning and imparting their knowledge. I would also like to thank my parents, Claire and Dimitris, and brother, Costas, for their support, patience and encouragement during the compilation of this dissertation.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Mavromati Galateia
30-6-2021

Abstract

Authorship attribution is the task of identifying the author of a given text. This task is equivalent to a ”classification problem”, which attempts to attribute a document to an author from a set of authors. In this Thesis, we performed classification experiments on novels from the Victorian era. The novels were extracted from GDELT database using several feature extraction techniques, including Part-of-Speech tag, TF-IDF, N-grams. The classification techniques applied are Support Vector Machine, Multilayer Perceptron, Logistic Regression, XGBoost, Bayes, and DistilBert. The main objective of this work is to study the effect of feature set selection, classification model, and size of the dataset in the implementation of the authorship attribution task and compare its behavior with state-of-the-art models like DistilBert. The experiments obtained indicate that the size of the dataset plays a significant role in the performance of the models considered; the classifiers are more important than the feature selection, while SVM classifiers combined with TF-IDF performed the best, achieving a remarkable 98% accuracy. The state-of-the-art models did not perform according to our expectations, but we proposed some ideas for further experimentation and improvements.

Περίληψη

Το Authorship Attribution ή αλλιώς η απόδοση συγγραφέα ανώνυμων κειμένων αποσκοπεί στην αναγνώριση του νόμιμου συγγραφέα ενός κειμένου από ένα σύνολο υποψηφίων συγγραφέων και αποτελεί πρόβλημα κατηγοριοποιήσης. Σε αυτή τη διατριβή, πραγματοποιήθηκαν πειράματα σε μυθιστορήματα από τη βικτοριανή εποχή. Αυτά τα μυθιστορήματα εξήχθησαν από τη βάση δεδομένων GDELT χρησιμοποιώντας διάφορες τεχνικές εξαγωγής χαρακτηριστικών συμπεριλαμβανομένου των Part-of-Speech tag, TF-IDF, N-grams. Οι μέθοδοι ταξινόμησης που χρησιμοποιήθηκαν ειναι οι: Support Vector Machine, Multilayer Perceptron, Logistic Regression, XGBoost, Bayes, and DistilBert. Ο κύριος σκοπός αυτής της εργασίας ήταν είναι να μελετήσει την επίδραση της επιλογής συνόλου χαρακτηριστικών, του μοντέλου ταξινόμησης και του μεγέθους του συνόλου δεδομένων στην υλοποίηση της εργασίας απόδοσης συγγραφέα και να συγκρίνει τη συμπεριφορά του με μοντέλα της σύγχρονης τεχνολογίας όπως το DistilBert. Τα αποτελέσματα που προέκυψαν από τα πειράματα είναι ότι το μέγεθος του συνόλου δεδομένων παίζει σημαντικό ρόλο στην απόδοση των μοντέλων. Οι ταξινομητές είναι πιο σημαντικοί από την επιλογή χαρακτηριστικών, ενώ η καλύτερη ακρίβεια επιτεύχθηκε από τους ταξινομητές SVM σε συνδυασμό με το TF-IDF καθώς πέτυχαν ένα αξιοσημείωτο 98%. Τα σύγχρονα μοντέλα δεν ανταποκρίθηκαν στις προσδοκίες μας, αλλά προτείνουμε ορισμένες ιδέες για περαιτέρω πειραματισμό και βελτιώσεις.

Table of contents

Acknowledgements **ix**

Abstract **xi**

Περίληψη **xiii**

Table of contents **xv**

List of figures **xvii**

List of tables **xix**

Abbreviations **xxi**

1 Introduction **1**

1.1 Problem Statement 2

1.1.1 Contribution of Thesis 2

1.2 Content organization 3

2 Related Work **5**

3 Feature Extraction **9**

3.1 Part-Of-Speech tag 9

3.2 N-grams 11

3.3 TF-IDF 12

3.4 Count Vectorizer 13

3.5 Hashing Vectorizer 15

4 Pretrained Models **17**

4.1 Bert 18

4.2 DistilBert 20

5 Experiments **23**

5.1 Dataset Information 23

5.2 Hyperparameters 25

5.3 Classification Methods 27

5.3.1 Support Vector Machine 27

5.3.2 Gradient Boosting 28

5.3.3 Logistic Regression 30

5.3.4 Artificial Neural Network 32

5.3.5 Naive Bayes 34

5.3.6 Random Forest 36

5.4 Results 37

6 Conclusions **43**

6.1 Summary 43

6.2 Future Work 44

Bibliography **45**

List of figures

3.1 Penn treebank POS tags [1]. 10
3.2 Word frequency map. 12
3.3 Count Vectorizer example. 14
3.4 Stemming vs Lemmatization. 14

4.1 Bert architecture [2]. 19

5.1 Authors' distribution in training set. The x-axis represents author I.D. and y-axis the number of occurrences in training set. 25
5.2 Word frequency. 26
5.3 Support Vector Machine. 28
5.4 Logistic Regression 30
5.5 Artificial Neural Network Architecture. 33
5.6 Random Forest 36

List of tables

2.1 Classification methods used in related work. 8

3.1 N-grams for author 1. 11

4.1 Glue test results on Bert. 20
4.2 Glue test results on DistilBert. 21

5.1 Authors list 24
5.2 Reduced dataset results (4500 instances). 38
5.3 Reduced dataset results (4500 instances) with POS Tag. 39
5.4 DistilBert results for 11146 instances. 40
5.5 Full dataset results. 41

Abbreviations

A.A.	Authorship Attribution
SVM	Support Vector Machine
D.T.	Decision Tree
KNN	K-Nearest Neighbour
GDELT	Global Database of Events, Language, and Tone
POS	Part-of-Speech
ID	Identity Document
TF-IDF	Term frequency–inverse document frequency
Bert	Bidirectional Encoder Representations from Transformers
TPU	Tensor Processing Unit
GLUE	General Language Understanding Evaluation
LDA	Linear discriminant analysis
MLP	Multilayer Perceptron
NB	Naive Bayes
i.e.	id est, that is
e.g.	for example
MDA	Multiple Discriminant Analysis

Chapter 1

Introduction

Authorship attribution (A.A.) is the process of attempting to identify the true author of a document or text, given a collection of documents whose authorship is known. A.A. constructs a classification problem, for which machine learning techniques are used in order to solve it. It is important to mention that it differs from text classification. In text classification only the text context is responsible for the classification process, but in A.A. also the writing style plays a significant role in classifying the text correctly to the author. The authorship attribution question [3, 4] is trying to determine the correct author of a given document based on text samples written by known authors. Pseudepigrapha has falsely attributed works, texts whose claimed author is not the true author, or a work whose real author attributed it to a figure of the past [5]. The Shakespeare authorship question argues that a collaborator or another author wrote the works attributed to Shakespeare. This A.A. problem has troubled literal scholars since the middle of the 19th century [6, 7, 8, 9, 10]. It is estimated that about 80 authors have cooperated with Shakespeare or be the rightful authors for some of his plays. The most well recognized are John Fletcher, Sir Francis Bacon, Christopher Marlowe, and Thomas Kyd. Moreover, some unidentified plays have been attributed to Shakespeare but for various reasons the authorship is questionable. Some of them are Edward III, The Two Noble Kinsmen and Pericles, Prince of Tyre. This playgroup is called Shakespeare apocrypha and has troubled researchers into finding the real author of these plays. A similar question has been raised for the oldest European literary documents: The Homeric Epics. The Homeric Question – concerning by whom, when, where and under what circumstances the Iliad and Odyssey were composed – continues to be debated [11, 12, 13]. This question was stated by Milman Parry [14], and now most classicists agree that, whether or not there was ever such

a composer as Homer, the poems attributed to him are to some degree dependent on oral tradition. Oral tradition was a generations-old technique that was the collective inheritance of many singer-poets [15].

A problematic field that A.A. has application to is forensic linguistics. It is the case when the number of suspected writers might be very large, possibly numbering in the many thousands. Also, there is often no guarantee that the true author of an anonymous text is among the known suspects. Another challenging aspect is when a candidate's amount of writing might be very limited, and the untitled text itself might be short [16]. It has various applications such as social media, fake news, plagiarism, death row statements, and suicide letters [17, 18, 19, 20].

1.1 Problem Statement

This Thesis will conduct a systematic approach to authorship attribution by performing experiments on 50 Victorian Era novels extracted from GDELT database. The primary purpose of this survey is to answer the following questions:

- Which data augmentation (feature extraction) techniques benefit the text classification models for this specific dataset?
- Does the size of the dataset play a significant role in achieving high accuracy?
- Which is more important? The feature set or the classification model?
- Is it better to combine feature extractions or use one solely?

To answer the above questions, we need to a) identify the feature extraction techniques and the classifiers and b) conduct experiments including both combined feature extractors with each classifier. We will perform the experiments on the full dataset and in a reduced dataset of 4500 instances.

1.1.1 Contribution of Thesis

The contribution of the Thesis is summarized as follows:

1. Analysis features versus classifier performance:
   Model interpretation is a crucial benefit of feature selection as it reduces the number

of features in the model and tries to optimize the model performance. The real query, though, is whether the selection of the features plays a more significant role than the classifier itself.

2. Feature extraction effect, either augmented or individually used:
   There is a variety of feature extraction methods that can be applied to an A.A. task. There are lexical, semantic, syntactic, and many more approaches can be used. We address whether it is preferable to use an augmented set of features or to use each one individually to receive the best possible outcome.

3. State-of-the-art pre-trained model performance in authorship attribution:
   Pretrained language models have shown excellent performance in several text classification tasks, including sentiment analysis, emotion classification, and topic classification, but little research has been done on the authorship attribution problem. In this Thesis, the goal is to extend this research and verify that pretrained model achieves the best results as proposed in [21, 22].

4. Machine learning classifiers' accuracy in A.A. tasks:
   Most surveys report that SVM is the best classifier for authorship attribution problems. We apply and compare six algorithms (SVM, MLP, Naive Bayes, Logistic Regression, Random Forest, and XGBoost) to discover the best classifier for the A.A. problem in the case of large datasets.

5. Optimal choice of feature selection and classifier:
   Different datasets provide different answers to the question: Which are the best feature extractor and classifier options for A.A. tasks. For the specific large dataset of 50 authors we have selected, we conducted various experiments and applied each feature extraction technique to every classifier to find the optimal choice.

1.2 Content organization

We organize this Thesis as follows. Chapter 2 discusses related work and authorship attribution's fields of application. Chapter 3 presents and analyses the feature extraction techniques used for the experiments. Chapter 4 introduces the stateoftheart pretrained models

Bert and DistilBert. In Chapter 5, we provide information about the dataset and the classification methods used in our experiments, and present the performance analysis results. Finally, Chapter 6 includes the conclusions and remarks on the proposed future work for further experimentation.

Chapter 2

Related Work

In this Chapter, we present a survey of the various techniques for solving the authorship attribution problem. Authorship attribution surveys focus on stylometric features to extract the author's writing signature and can be applied to several topics. One application of A.A. that has troubled scholars is around Shakespeare. It is a common belief that in the Eliza bethan era (1558–1603), authors used to collaborate. From time to time, scholars embrace the view that Shakespeare did not solely write some of his plays because of his lack of education and aristocratic sensibility, but other authors were involved in the writing.

On the other hand, most modernday scholars reject this claim, arguing that there is strong evidence that Shakespeare wrote the plays and poems attributed to him. This conflict has led to an endless academic debate with unsettling answers. Identifying the real author of the Shakespearean works is known as the ”Shakespeare authorship question.” Scholars use different techniques in determining Shakespeare's authorship. Plechac [23] uses Support Vector Machine (SVM) as a classifier together with rhythmic types to identify writing style, while Merriam [24] chooses Multilayer Perceptron combined with function words. Boyd [25] uses Linear discriminant analysis (LDA), Decision Trees (D.T.s), and SVM together with five types of measures to recognize the authors writing signature. Aljumily [7] uses hierarchical and non hierarchical clustering together with function words, bi-gram, and tri-grams. However, these different classification methods and datasets result in contrasting outcomes, and the results given are inconsistent with each other and can not provide a final answer.

Similar research took place with several books which had a doubtful attribution. One of them is the 15th Oz book ”The Royal Book of Oz”, which has initially been credited to L. Frank Baum, the writer of the previous 14 Oz books. However, it was later recognized to be

written entirely by Ruth Plumly Thompson and Binongo's statistical analysis using function words proves that assumption [26]. Gungor experimented with 50-author dataset derived from the Victorian Era and tested traditional feature extraction methods (bag of words, n-grams etc) and new techniques like word2Vec [27]. However, it was found that the latter didn't outperform traditional ways. "The Letter to the Hebrews" also has a debated authorship and using function words, trigram Markov method, and multiple discriminant analysis. It was discovered that Barnabas was stylistically closer to the writing style than Paul, Matthew, Mark, Luke, or John [28]. Another survey focuses on the use of alliteration in historical texts [29]. Alliteration did not provide a suitable feature extractor but improved the total accuracy when combined with lexical stress and traditional stylistic features. However, the main problem was that there is no accurate time-period pronunciation dictionary and used present-day pronunciations of written words.

Another major topic that A.A. finds application to is forensic linguistics. There have been various cases where some documents could be the key to solve an issue. An example based on real events is when an employee was fired for a racist email sent from his (open) cubicle, which he denied sending [3]. The main difference in forensic authorship attribution is that the document belongs to one or more authors from an available set of authors, or an unknown author. Social media conceal several digital crimes such as pedophilia and identity theft. In [17], the problem that arises in social media is the size of the messages. To encounter this, they computed very low-level lexical statistics and emphasized punctuation, abbreviations, and character-based signifiers common in Internet culture. They also examined supervised learning-based methods that are effective for small sample sizes. Specifically, the efficient classification method was Power Mean SVM (PMSVM). Another approach to identify the social media author and distinguishing between fake and real tweets was proposed in [30]. The dataset used was streamed from Twitter, with a maximum of three thousand tweets per author, and consisted of two features, the author and the tweets. They extracted features using a bag of words and vector space model and tested the classification techniques Logistic Regression and Naive Bayes where they resulted in 91% and 90% accuracy, respectively.

Moreover, a vast majority of papers focus on fake news detection using several machine learning methods. Supervised learning algorithms such as KNN, Naive Bayes, Random Forest, SVM and XGBoost are used in [31] using several feature sets including language, lexical, psycholinguistic, and semantic features. The dataset consisted of 2282 BuzzFeed news

articles related to 2016 U.S. election labeled by journalists and enriched with comments associated with the news stories and shares and reactions from Facebook users. The experiments showed that the best classifiers were Random Forest with 85% and XGBoost with 86% accuracy, respectively. In Table 2.1, we summarize the classification methods used and point to the corresponding reference. .

Classifier	Reference
SVM	[23] [25] [27] [29] [17] [31]
MLP	[10] [29]
Naive Bayes	[27] [30] [31]
Clustering	[7] [27]
Ensemble methods	[27] [31]
LDA	[25]
D.T.	[25]
PCA	[26]
CNN	[27]
MDA	[28]
Logistic Regression	[30]
Random Forest	[31]
KNN	[31]

Table 2.1: Classification methods used in related work.

Chapter 3

Feature Extraction

The most important task in authorship attribution is to find distinctive features that would indicate an author's writing style. There are several types of stylometric features. The main categories are lexical, character, syntactic, semantic, and application-specific features. This Thesis will focus on Part-of-Speech tag, which belongs to the syntactic category, lexical features, i.e., function words and N-grams, and several vectorizers.

3.1 Part-Of-Speech tag

Dionysius Thrax (c. 100 B.C.) is considered to be the author of the earliest grammatical text in the Greek language and the first man to distinguish between the eight-word classes: noun, verb, pronoun, preposition, adverb, conjunction, participle, and article. This set of eight words has become the basis for descriptions of European languages for the next 2000 years. Part-of-speech tagging aims at labeling each word with a unique tag that indicates its syntactic role. This technique is essential because authors tend to use similar syntactic patterns unconsciously thus, we can extract a unique writing signature from each author. Part-of-speech (POS) tagging is the process of assigning a part-of-speech to each word in a text. The input is a sequence $x_1, x_2, ..., x_n$ of (tokenized) words and a tagset, and the output is a sequence $y_1, y_2, ..., y_n$ of tags, each output y_i corresponding exactly to one input x_i [32]. However, sometimes words may have more than one possible part-of-speech.Consider the several different uses of the word "back":

earnings growth took a **back**/J.J. seat

a small building in the **back**/N.N.

a clear majority of senators **back**/VBP the bill

Dave began to **back**/V.B. toward the door

enable the country to buy **back**/R.P. about debt

I was twenty-one **back**/R.B. then

POS tagger is responsible for solving this problem of ambiguity and assign the proper tag for the context.

An important tagset containing corpus tagged sentences is the Penn treebank. The Penn treebank is perhaps one of the most popular NLP datasets. It operated for eight years (1989-1996) encompassed a vast array of natural language sources ranging from technical handbooks to Wall Street Journal articles. The majority of the output of the Penn Treebank consists of POS tagged and syntactically bracketed versions of written texts. In the early years of the project, bracketing was done using a relatively simple skeletal parse. In later years of the project, phases made use of a richer predicate-argument bracketing schema using common parenthesized tree representations. In Penn treebank, parts of speech are generally represented by placing the tag after each word, delimited by a slash. The table in Figure 3.1 displays the part-of-speech tags contained in the corpus.

Tag	Description	Example	Tag	Description	Example
CC	coord. conjunction	*and, or*	RB	adverb	*extremely*
CD	cardinal number	*one, two*	RBR	adverb, comparative	*never*
DT	determiner	*a, the*	RBS	adverb, superlative	*fastest*
EX	existential there	*there*	RP	particle	*up, off*
FW	foreign word	*noire*	SYM	symbol	*+, %*
IN	preposition or sub-conjunction	*of, in*	TO	"to"	*to*
JJ	adjective	*small*	UH	interjection	*oops, oh*
JJR	adject., comparative	*smaller*	VB	verb, base form	*fly*
JJS	adject., superlative	*smallest*	VBD	verb, past tense	*flew*
LS	list item marker	*1, one*	VBG	verb, gerund	*flying*
MD	modal	*can, could*	VBN	verb, past participle	*flown*
NN	noun, singular or mass	*dog*	VBP	verb, non-3sg pres	*fly*
NNS	noun, plural	*dogs*	VBZ	verb, 3sg pres	*flies*
NNP	proper noun, sing.	*London*	WDT	wh-determiner	*which, that*
NNPS	proper noun, plural	*Azores*	WP	wh-pronoun	*who, what*
PDT	predeterminer	*both, lot of*	WP$	possessive wh-	*whose*
POS	possessive ending	*'s*	WRB	wh-adverb	*where, how*
PRP	personal pronoun	*he, she*			

Figure 3.1: Penn treebank POS tags [1].

In this survey the tags we used to distinguish the writing style of an author are verb,

adjective, adverb, and noun.

3.2 N-grams

The N-gram model is constructed by calculating the frequency of the word sequence in a corpus text and then estimating the probability. In other words, given a sequence of N-1 words, the N-gram model predicts the words most likely to follow this sequence. A model that only relies on the frequency of words without looking at previous words is called unigram ($N = 1$). Word N-grams are sequences of "N" adjacent words. If a model only considers the previous word to predict the current word, then it is called bigram ($N = 2$). If one considers the first two words, it is a trigram model ($N = 3$). Word n-grams have been excessively used in many NLP applications. Guessing the next word (or word prediction) is an essential subtask of speech, hand-writing recognition, augmentative communication for the disabled, and spelling error detection. In such tasks, word identification is difficult because the input is very noisy and ambiguous. Thus, looking at previous words can give us an important cue about what the next ones will be. To extract word N-grams text has to be tokenized thus making the method language-dependent and complicated [33].

Traditional N-grams are sequences of elements as they appear in texts. These elements can be words, characters, POS tags, or any other elements as they appear one after another in texts. A standard convention is that "n" in N-grams corresponds to the number of elements in a sequence [34]. An example of N-grams derived from the dataset used for in this Thesis is shown in Table 3.1 and shows the unigram, bigram, and trigram for author with $I.D. = 1$.

1-Gram	Occurrence	2-Gram	Occurrence	3-Gram	Occurrence
would	5004	young man	421	love gone astray	64
saI.D.	4420	sugar princess	295	sugar princess chapter	26
one	3695	marriage bond	265	drew long breath	25
could	3571	new york	241	young miss giddy	25
mr	3142	long time	154	yo fa tt	22

Table 3.1: N-grams for author 1.

Arthur Conan Doyle, e.g. author "1", has 16 books registered in the dataset with 1394980

total words. As we can see, the most used word is ”would” with 5004 occurences. In bigrams ”young man” is displayed 421 times, while the highest trigram sequence is ”love gone astray” with 64 occurrences. We used N-grams in TF-IDF, Count Vectorizer, and Hashing Vectorizer, as explained below.

3.3 TF-IDF

Word frequencies in a document contain important information about the meaning of a specific linguistic item. A simple way of encoding sentences in numerical form is using a bag-of-words model, creating dictionaries of word to word-frequency mappings for each item.

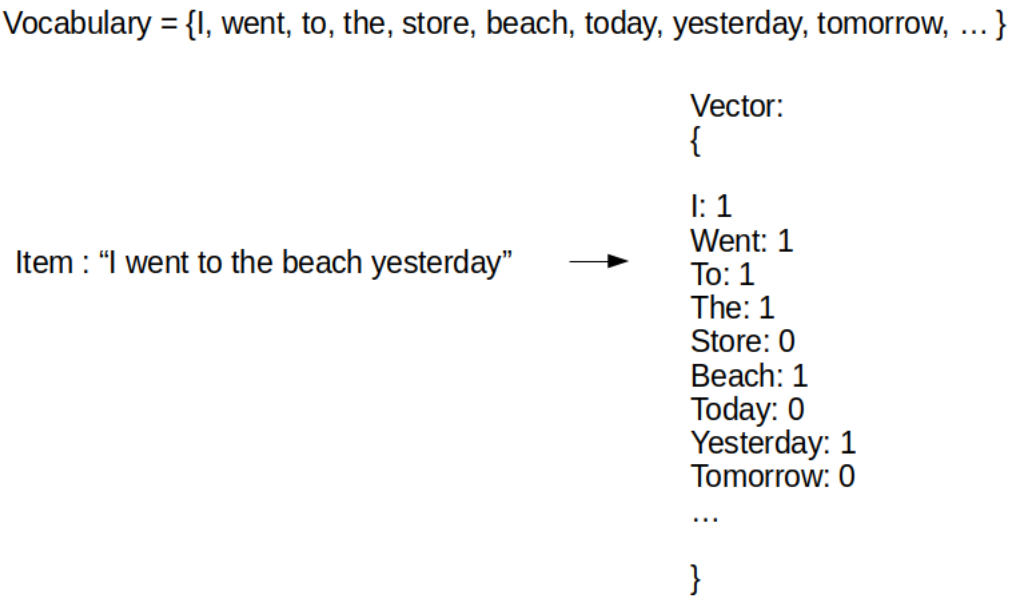

Figure 3.2: Word frequency map.

As we can see in Figure 3.2, this type of vectorization produces a highly sparse vector, meaning that most of the dictionary values are set to zero; thus, the mere presence of a word in the vector object is more important than the number of words that it appears in.

However, this mode of vectorization contains a very significant pitfall: Several frequently occurring words such as pronouns that don’t contain any information significant to class categorization induce a skewed dataset that will affect our category predictions down the line. To avoid this, we employ TF-IDF Normalization.

The statistical measure TF-IDF stands for Term Frequency Inverse Document Frequency, and it examines how relevant a word is to a document in a collection of documents. This is accomplished by multiplying two metrics over a group of documents: the number of times a word appears in the document and the inverse document frequency. In feature extraction techniques, it is frequently employed as a weight. The equation of TF-IDF is:

$$TFIDF(t, d, D) = TF(t, d) * IDF(t, D)$$

where:

(t, d, D): term, a document and the corpus respectively.

$T.F.(t, d)$: the number of times that term t appears in document d.

If a term appears very often across the corpus, it means it does not carry special information about a particular document. We need a way to weigh down the effects of too frequently occurring terms. Also, the terms that occur less in the document can be more relevant. We need a way to weigh up the effects of less frequently occurring terms. Inverse document frequency is a numerical measure of how much information a term provides:

$$IDF(t, D) = \log \frac{|D| + 1}{DF(t, D) + 1}$$

where the total number of documents is represented by $|D|$ and the number of documents that contain the term t is represented by $D.F.(t, D)$.

In our experiments for the computation of TF-IDF, we took into account N-grams with the range of $(1, 3)$, i.e., unigrams, bigrams, and trigrams.

3.4 Count Vectorizer

Count Vectorizer is used to transform a given text into a vector, based on the frequency (count) of each word that occurs in the entire text. It creates a matrix in which a column of the matrix represents each unique word, and each text sample from the document is a row in the matrix. The value of each cell represents the count of the word in that particular text sample. Count-Vectorizer counts the number of words occurring more frequently in the document, which may overshadow the words which occur less frequently but may have more importance to the document feature [35], this limitation is handled in TF-IDF. An example of Count Vectorizer is shown in Figure 3.3.

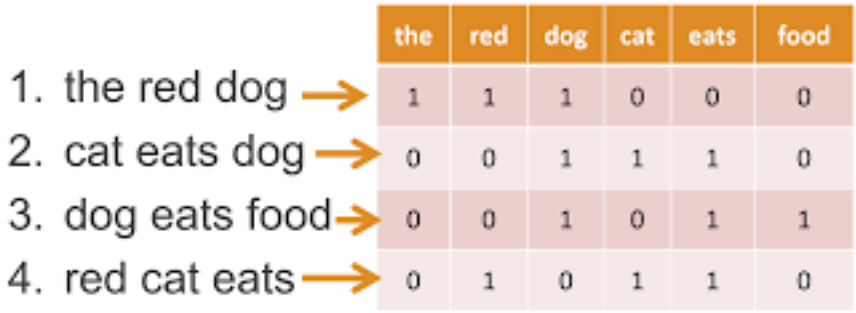

Figure 3.3: Count Vectorizer example.

It is important to point out that Count Vectorizer does not do the Lemmatization or Stem ming automatically. Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes or the roots of words known as a lemma. For example, if we stemmed ” loving” it will produce the result of ” lov”. Lemmatization is the grouping together of different forms of the same word. In Figure 3.4, there are several examples that indicate the difference between Stemming and Lemmatization.

Original	Stemming	Lemmatization
New	New	New
York	York	York
is	is	be
the	the	the
most	most	most
densely	dens	densely
populated	popul	populated
city	citi	city
in	in	in
the	the	the
United	Unite	United
States	State	States

Figure 3.4: Stemming vs Lemmatization.

3.5 Hashing Vectorizer

Hashing vectorizer uses the hashing trick to map the token string name to an integer index. Conversion of text documents into a matrix is done by this vectorizer where it turns the documents into a sparse matrix that counts how many times the token appears. Hashing Vectorizer and Count Vectorizer are meant to do the same thing: convert some text documents to a matrix of token counts. The difference is that Hashing Vectorizer does not store the resulting vocabulary (i.e., the unique tokens). Each token in Hashing Vectorizer directly maps to a column position in a matrix, where its size is predefined. The benefit of not storing the vocabulary is that the resulting HashingVectorizer object, when saved, would be much smaller and thus faster to load back into memory when needed. The downside of doing this is that it will not be possible to retrieve the actual token given the column position. This would be especially important in tasks like keyword extraction, where you want to retrieve and use the actual tokens.

Chapter 4

Pretrained Models

Transfer learning is a machine learning research subject that focuses on storing and transferring knowledge learned while addressing one problem to a different but related problem [36]. In this way, the model created for one task can be reused as a starting point for another task. This model is called pre-trained. Pre-trained language modeling has achieved state-of-the-art results on various downstream NLP tasks such as sentiment analysis, data classification, and question answering [37]. As proposed in [38], the typical approach for pre-trained models consists of three steps:

1. Selection of Source Model.
   There is a wide variety of pre-trained models to find an appropriate choice for a specific classification problem.

2. Reuse Model.
   Then use that model as a starting point building another model. Depending on the modeling technique employed, this could include using all or part of the model.

3. Fine tuning the model.
   The model may need to adjust or improve the data based on the input-output available for the task of interest. This step is not mandatory.

The advantage of implementing this approach is that only a few parameters need to be learned from scratch.

The Hugging Face transformers package is a popular Python library providing pre-trained models for several natural language processing (NLP) tasks. These tasks include text classification, information extraction, question answering, and text generation. Some transformers

examples are OpenAI's Generative pre-training Transformer (GPT) [39], Bi-directional Encoder Representations from Transformers (BERT) [40], DistilBert [41] and Robustly Optimized BERT Pretraining Approach (RoBERTa) [42].

In this Chapter, we introduce the transformers Bert and DistilBert derived from the Hugging Face.

4.1 Bert

BERT(Bidirectional Encoder Representations from Transformers) is a new language representation model proposed in [40]. It pre-trains deep bidirectional representations from an unlabeled text by jointly adjusting all layers of the left and right context. In contrast to the directional model that reads text input sequentially (from left to right or right to left), the Transformer encoder reads the entire sequence of words at once. Bert is a transformer model pre-trained on large corpus from Wikipedia and the Bookcorpus [43] using two training objectives. The training of Bert is conducted in two phases: pre-training and fine-tuning. In pre-training, the model is trying to understand the language and there are two main tasks. The first one is Masked Language Model (MLM). In MLM, Bert takes in sentence with random words filled with masks. The goal is to output these mask tokens. The second is Next Sentence Prediction (NSP). Bert receives two sentences and determines whether the second sentence follows the first. In other words, it learns the relationship between this two sentences and could be seen as a "binary classification" problem. In fine-tuning, the model is trying to learn a specific task. One or more fully-connected layers are typically added on top of the final encoder layer [44]. BERT decomposes its input sentences into Token Embeddings, the WordPiece tokens [45] that have a vocabulary of 30.000 tokens. The use of WordPieces downsizes the vocabulary, and Bert becomes more powerful against out-of-vocabulary words. The Sentence Embeddings is the sentence number that is encoded into a vector. The Position Embeddings is the position of a word within a sentence that is encoded into a vector. These three vectors are summed together to get an embedding vector that is passed through the main body of the model. This produces the output representations that are fed to the final, application-dependent layer, e.g., a classifier [46]. Although out of all the models, BERT seems to perform very well, it still has some loopholes in its implementation. First of all, it uses [MASK] token during the pre-training process, but these token are missing from real

data during the fine tuning phase, which results in a pre-train-finetune discrepancy. Another weakness of BERT is that it assumes that masked tokens are independent of each other and are only predicted using unmasked tokens [37]. The architecture of Bert can be seen in Figure 4.1.

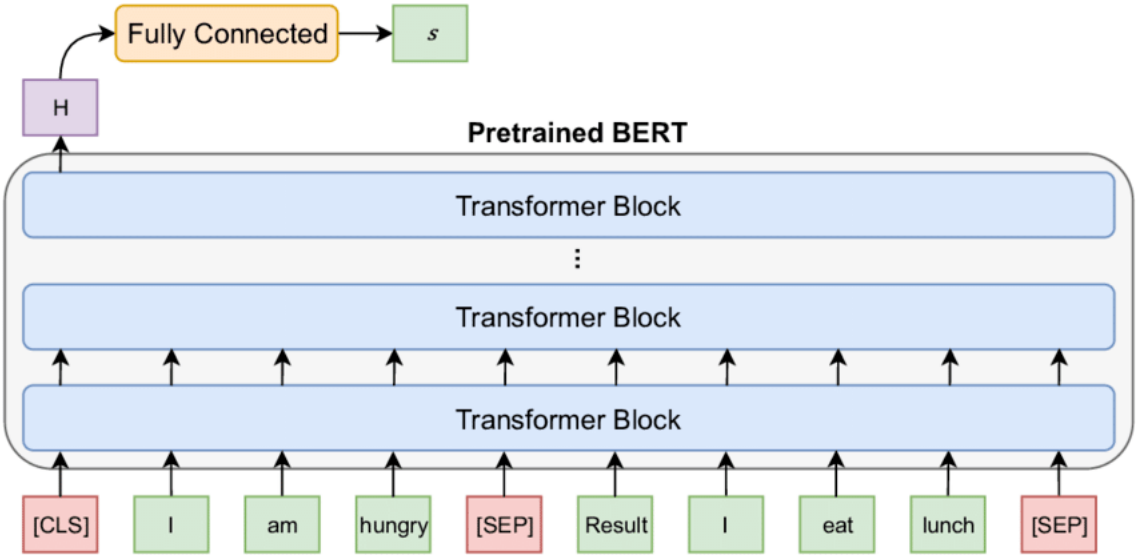

Figure 4.1: Bert architecture [2].

There are two versions of Bert: base and large. The difference between them lies in the number of layers, their hidden size, and the number of attention heads. In this thesis, we used Bert base with the specifications of 12 layers,768 hidden size, 12 self-attention heads, and Total Parameters= 110M.

Generally, Transformer-based models that are pre-trained on large datasets achieve state-of-the-art accuracy for NLP tasks. However, they demand too many resources and a lot of computation to suit low capability devices.

The model was trained on four cloud TPUs in Pod configuration (16 TPU chips total) for one million steps with a batch size of 256. The sequence length was limited to 128 tokens for 90% of the steps and 512 for the remaining 10%. The optimizer used is Adam with a learning rate of 1e-4, $\beta_1 = 0.9$ and $\beta_2 = 0.999$, a weight decay of 0.01, learning rate warmup for 10,000 steps and linear decay of the learning rate after. When fine-tuned on downstream tasks, this model achieves the Glue test results shown in Table 4.1.

MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6

Table 4.1: Glue test results on Bert.

4.2 DistilBert

DistilBERT is a small, fast, cheap, and light Transformer model trained by distilling BERT base [41]. Bert-base-uncased has 40% more parameters than DistilBert, runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark. Knowledge distillation refers to the idea of model compression by teaching a smaller network, step by step, exactly what to do using a bigger already trained network. To pass the knowledge from the teacher to the student, cross-entropy over the soft targets is used. The training loss becomes:

$$L = -\sum_i t_i * log(s_i)$$

where t_i is a probability estimated by the teacher and s_i is the probability estimated by the student.

Temperature is a hyperparameter that is applied to logits to affect the final probabilities from the softmax. A low temperature (below 1) makes the model more confident, while a high temperature (above 1) makes the model less confident. A softmax temperature is used as follows:

$$p_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

where T controls the smoothness of the output distribution and z_i is the model score for class i. At training, the same T is applied both to the student and the teacher. At inference, T is set to 1 to recover a standard softmax.

Bert and the student (DistilBert) have the same basic architecture. However, the token-type embeddings and the pooler does not exist in DistilBert while the number of layers is reduced by a factor of 2. The main purpose is to reduce the layer size. The student is initialized from the teacher by taking one layer out of two. Finally, DistilBERT is distilled in huge batches (up to 4K instances per batch) using dynamic masking and without the next sentence prediction target. DistilBERT was trained on the same corpus as the original BERT model.

When fine-tuned on downstream tasks, this model achieves the Glue test results are shown in Table 4.2

MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE
82.2	88.5	89.2	91.3	51.3	85.8	87.5	59.9

Table 4.2: Glue test results on DistilBert.

Chapter 5

Experiments

5.1 Dataset Information

The GDELT Project is an open platform for research and analysis of global society. 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 a fee. For our experiments, we extracted a dataset[1] from the GDELT database consisting of 50 authors that satisfy the following criteria: English language writing authors, authors who have enough books available (at least 5), and belong to the 19th century. These requirements were chosen to decrease the bias and create a reliable dataset. To make the classification problem more challenging, the first and the last 500 words have been removed to take out specific features such as the author's name, the name of the book, and other word-specific features.

Moreover, we chose the top 10.000 words that occurred in the whole 50 authors' text data corpus, and we split the entire book into text fragments with 1000 words each. The 50 authors selected are shown in Table 5.1 and in Figure 5.1.The distribution of authors in the training set, 45 authors in total, can be seen with the x-axis representing the author I.D. and the y-axis the number of occurrences in the training set. The author with I.D. = 8, i.e., James Balwin, has the highest number of occurrences with 6914 total, while the lowest belongs to I.D. = 16, i.e., Rudyard Kipling, with 183 instances. The author list that was not included in the training set, consists of authors with I.D.s 5 (George Eliot), 7 (Jack London), 31 (Frances Hodgson Burnett), 47 (Sarah Stickney Ellis), and 49 (Thomas Nelson Page). We added them to the test set, which finally contained 50 authors.

[1] https://archive.ics.uci.edu/ml/datasets/Victorian+Era+Authorship+Attribution

I.D.	Author name	I.D.	Author name
1	Arthur Conan Doyle	26	Bret Harte
2	Charles Darwin	27	Catharine Maria Sedgwick
3	Charles Dickens	28	Charles Reade
4	Edith Wharton	29	Edward Eggleston
5	George Eliot	30	Fergus Hume
6	Horace Greeley	31	Frances Hodgson Burnett
7	Jack London	32	George Moore
8	James Baldwin	33	George William Curtis
9	Jane Austen	34	Helen Mathers
10	John Muir	35	Henry Rider Haggard
11	Joseph Conrad	36	Isabella Lucy Bird
12	Mark Twain	37	Jacob Abbott
13	Nathaniel Hawthorne	38	James Grant
14	Ralph Emerson	39	James Payn
15	Robert Louis Stevenson	40	John Kendrick Bangs
16	Rudyard Kipling	41	John Pendleton Kennedy
17	Sinclair Lewis	42	John Strange Winter
18	Theodore Dreiser	43	Lucas Malet
19	Thomas Hardy	44	Marie Corelli
20	Walt Whitman	45	Oliver Optic
21	Washington Irving	46	Sarah Orne Jewett
22	William Carleton	47	Sarah Stickney Ellis
23	Albert Ross	48	Thomas Anstey Guthrie
24	Anne Manning	49	Thomas Nelson Page
25	Arlo Bates	50	William Black

Table 5.1: Authors list

Finally, the full dataset consists of 93.600 instances, where 53.678 of them belong in the training set, and 39.922 comprise the test set. The large-scale dataset makes the project time-consuming and often generates memory and runtime errors. To solve this problem, we

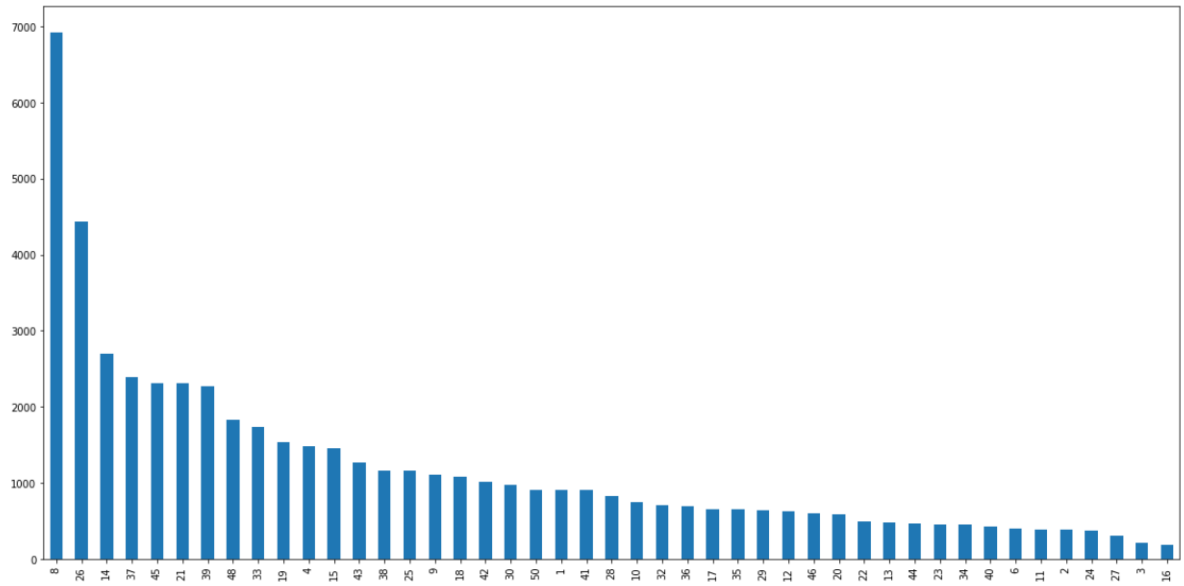

Figure 5.1: Authors' distribution in training set. The x-axis represents author I.D. and y-axis the number of occurrences in training set.

reduced the dataset. We conducted the first experiments on a dataset with 4500 instances extracting some first impressions on the classifiers. We applied the best of them in the full dataset where possible.

In Figure 5.2, we can see the word frequency before removing stop words. As we observe, the most common words do not provide any significant information. Hence we need to apply text clean-up when using the feature extractors.

5.2 Hyperparameters

GridSearchCV is a function that comes in Scikit-learn's model selection package. This function helps to traverse the predefined hyperparameters and adapt the model to the training set. So, in the end, we can select the best parameters from the listed hyperparameters. Generally, a hyperparameter is a parameter whose value is used to control the learning process. Selecting the appropriate hyperparameters for the classifier results in a high accuracy and success rate. A low learning rate results to ignoring important patterns in the data. While a high learning rate may result to collisions. Grid search measures the performance using the "Cross Validation" technique. Cross-validation is mainly used to apply machine learning to estimate the skill of the machine learning model on unknown data. That is, when used to predict data that is not used during model training, use a limited number of samples to esti-

	Common_words	count
0	the	3142432
1	and	1898968
2	of	1782356
3	to	1547337
4	a	1278583
5	i	966878
6	in	957556
7	that	702345
8	he	699602
9	it	676919
10	was	674696
11	his	529741
12	you	489528
13	â	478332
14	with	463788
15	her	463259
16	as	460455
17	for	433169
18	had	406999
19	is	405003

Figure 5.2: Word frequency.

mate the overall expected performance of the model. The general procedure of k-fold Cross Validation explained in [47] is as follows:

1. Shuffle the dataset in random.

2. Partition the dataset into k equal sized subsamples.

3. For each k subsample:

    - Take the subsample as the validation data for the test dataset.
    - Take the rest of the subsamples as a training dataset.
    - Fit a model on the training set and make an evaluation on test dataset.
    - Keep the evaluation score and discard the model.

4. Summarize the evaluation scores of the model.

5.3 Classification Methods

As mentioned before, authorship attribution is equivalent to a ”classification problem”. It is important to find the right classifier to provide the best possible accuracy. In this Chapter, we present the classification methods used for this dataset. We used only supervised learning techniques, including Support Vector Machine, Gradient Boosting, Logistic Regression, Artificial Neural Network, Naive Bayes, and Random Forest. We tested each classifier with different hyperparameters to find the optimal choice.

5.3.1 Support Vector Machine

Support Vector Machine (Figure 5.3) can be used as a classification or regression method, maintaining in both cases the main concept that characterizes the algorithm which is to draw a hyperplane of maximum margin.

In the classification case, the task is pretty simple: find the hyperplane that separates the classes and has maximum margin, i.e., maximum distance from the closest point of each class, which is where the support vectors are drawn. For new data, the algorithm decides the class by comparing its position relative to the support vectors. In Regression, the predictions lie on the hyperplane.

One can add a trade-off parameter to the objective function:

$$min\frac{1}{2}||w||^2 + C\sum|\xi_i|$$

where ξ denotes the deviation from the support vector, and tune the hyperparameter C to match the data’s requirements and achieve higher performance. A higher C value means less tolerance for the points inside the margin in classification and outside the margin in regression.

When choosing Radial Basis Function (RBF) as a kernel, the parameters taht are taken into account are C and gamma. ”C” weigh the misclassification of training samples. A low C smoothes the decision surface, while a high C aims to correctly classify all training examples. Gamma defines what will be the influence of a single training example. The larger the gamma, the closer the other samples must be affected [48]. Grid search tested the following values:

- ’C’: [0.1, 1, 10, 100, 1000]
- ’gamma’: [1, 0.1, 0.01, 0.001, 0.0001]

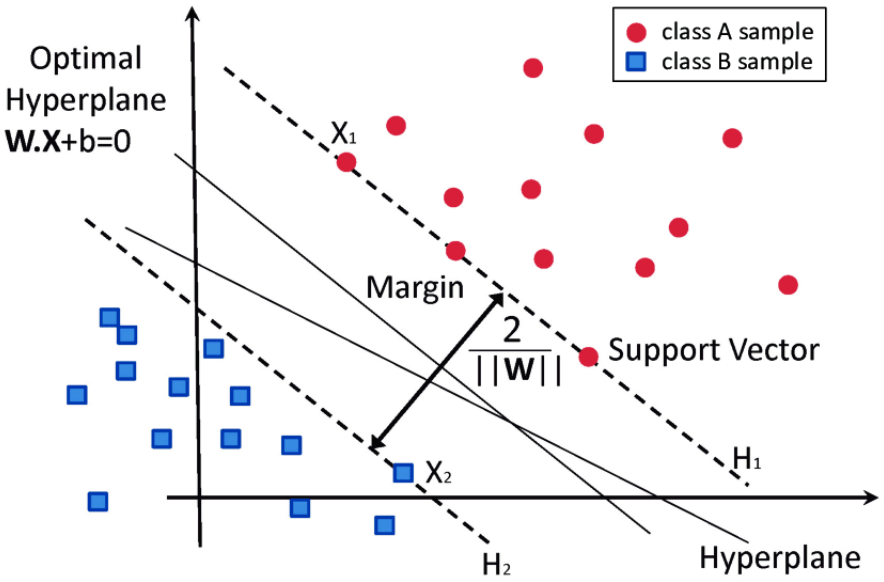

Figure 5.3: Support Vector Machine.

- 'kernel': ['rbf']

In this specific dataset, GridSearchCV found that the best parameters for SVM are C = 1000, kernel="rbf", random state=0 and gamma = 0.001.

SVM is used as a classification method for authorship attribution problems in [49, 50, 51, 52, 53, 54, 21, 55, 56].

5.3.2 Gradient Boosting

Gradient boosting classifiers are similar to ensemble classifiers. They use a collection of weak classifiers to make predictions, except that they are trained in succession to improve upon the false classification the previous classifiers have made.

In gradient boosting, to produce a more accurate estimate of the response variable, the learning procedure fits new models in a sequential manner. The principle idea behind this algorithm is to construct the new base-learners that is related to the maximum negative gradient of the loss function associated with the entire ensemble. The choice of the loss function is up to the researcher, with both a rich variety of loss functions derived so far and with the possibility of implementing one's own task-specific loss [57].

The basis for any gradient boosting algorithm is the following: Suppose an imperfect classifier is trained (e.g. a simple decision tree with low depth), the predicted class values for a single sample x are

$$F_0(x) = y_p$$

This weak classifier could be any model, in our case a tree model fitted to minimize a loss function $L(y_i, F_0(x))$. A way to improve upon this classifier would be to calculate the residuals of these approximations:

$$h_0(x) = y - F_0(x)$$

where y are the ground truth labels, and fit a new model to predict these values, thus obtaining a more efficient classifier

$$F_1(x) = F_0(x) + h_0(x)$$

This can be done for an arbitrary number of steps until we are satisfied with our approximation, in general at the m-th step of approximation:

$$F_m(x) = F_{m-1}(x) + h_{m-1}(x)$$

More accurately, we use a weighted sum of successive predictors

$$F_m(x) = F_{m-1}(x) + \gamma_{m-1} h_0(x)$$

Where the γ_{m-1} coefficients are set by performing ”line search” to find the value that minimizes the aforementioned loss function:

$$\gamma_m = argmin_\gamma \{\sum_{i=1}^{n} [L(y_i, F_{m-1}(x_i) - \gamma h_{m-1}(x_i)]\}$$

We implemented the gradient boosting classifier using the XGBoost API. XGBoost is designed for speed and performance and is an applicaion of gradient boosted decision trees. XGBoost provides a parallel tree boosting that provide a fast and accurate solution for several data science problems. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples [58].

We tested the following XGBoost’s hyperparameters:

- ’max depth’: [2,4,6]
- ’n estimators’: [50,100,200]

Where max depth refers to a tree's maximum depth. As this value is increased, the model becomes more comlex and more likely to overfit. N estimators are the number of gradient boosted trees. Equivalent to the number of boosting rounds. The parameters, finally, used were max depth = 2 and n-estimators= 200. Some paper examples of Gradient Boosting in A.A. can be found here [27, 21, 59, 60, 55, 61].

5.3.3 Logistic Regression

A logistic regression model simulates how brain neurons work to make decisions. Essentially stimuli from different nerve endings are aggregated in a neuron that fires only if the total intensity of the stimulation is significant enough to surpass a certain threshold (Figure 5.4).

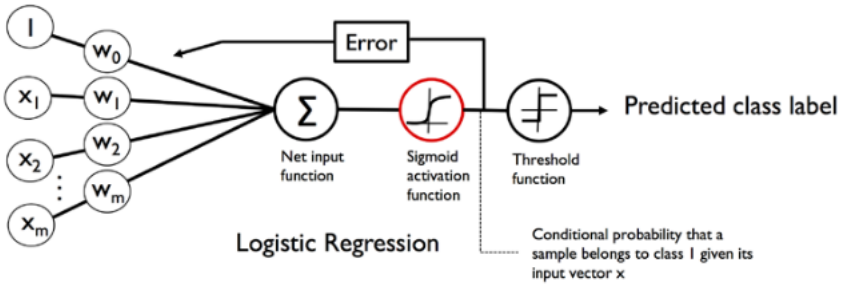

Figure 5.4: Logistic Regression

Similarly to the process described previously, an input vector $x = [x_1, x_2, \ldots, x_n]$ (stimulus) is applied to a weight vector w to produce a stimulus output:

$$z = w_0 x_0 + w_1 x_1 + ... + w_n x_n$$

This output is passed through an *activation function* $\phi(z)$ that transforms the real number output to represent a probability between $[0, 1]$. This function is usually either the sigmoid function:

$$\phi(z) = \frac{1}{1 + e^{-}z}$$

Another often used function in Logistic Regression is the reLU function and its derivative, leaky reLU, boasting simplicity that results in smaller evaluation time.

The final output from the activation function is compared to a threshold value and a decision about the objects class is made

$$y = g(z) = \begin{cases} 1 & \text{if } \phi(z) \geq 0.5 \\ 0 & \text{otherwise.} \end{cases}$$

The subsequent model is optimized by maximizing the subsequent likelihood function

$$L(\mathbf{w}) = P(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) = \prod_{i=1}^{n} P\big(y^{(i)} \mid x^{(i)}; \mathbf{w}\big) = \prod_{i=1}^{n} \Big(\phi\big(z^{(i)}\big)\Big)^{y^{(i)}} \Big(1 - \phi\big(z^{(i)}\big)\Big)^{1-y^{(i)}},$$

Which is the same as maximizing log-likelihood:

$$l(\mathbf{w}) = \log L(\mathbf{w}) = \sum_{i=1}^{n} y^{(i)} \log\Big(\phi\big(z^{(i)}\big)\Big) + \big(1 - y^{(i)}\big) \log\big(1 - \phi\big(z^{(i)}\big)\big)$$

For better optimization performance, this loss function is amended with an **l2 normalization** term

$$J(\mathbf{w}) = \sum_{i=1}^{m} \Bigg[- y^{(i)} log\Big(\phi\big(z^{(i)}\big)\Big) - \big(1 - y^{(i)}\big) log\Big(1 - \phi\big(z^{(i)}\big)\Big)\Bigg] + \frac{\lambda}{2} \sum_{j=1}^{m} w_j^2$$

The most famous methods to solve the above unconstrained optimization problem are **gradient-based** methods, such as batch gradient descent which involves iteratively improving the model by iteratively setting the weights:

$$w_j := w_j + \eta \sum_{i=1}^{n} \big(y^{(i)} - \phi\big(z^{(i)}\big)\big) x_j^{(i)}$$

Factoring in the normalization term:

$$w_j := w_j + \eta \sum_{i=1}^{n} \big(y^{(i)} - \phi\big(z^{(i)}\big)\big) x_j^{(i)} - \eta\lambda w_j$$

Where η is a parameter called “learning rate”,the value of which corresponds to the “step size” when progressing towards a final estimation. This procedure iteratively proceeds to a classifier vector that correctly categorizes each class sample.

The parameter tested for logistic Regression using sklearn’s Grid Search CV was ”C” with values (0.0001, 100, 20). Smaller values indicates stronger regularization, as proposed and in SVM. We discovered that C = 5.26325263157894 is the optimum parameter for logistic regression. Logistic regression is used in [21, 62, 56, 63, 30]

5.3.4 Artificial Neural Network

Similar to logistic Regression described before, artificial neural networks simulate biological neurons firing in tandem to encode information and make predictions. Each individual neuron is described by a vector of weights w_j where j is the neuron index in the layer. The neuron outputs, consist of the dot product of the input vector filtered through an activation function, usually the logistic sigmoid function we discussed previously:

$$a_j = \sigma(z_j) = \sigma(\sum_{i=1}^{D} w_{ji}x_i + w_{j0}) \tag{5.1}$$

Where z_j is our "activation" dot product plus a bias term $z_j = w_j x + w_{j0}$, or if we use extended vector notation, i.e $x' = [1, x_1, x_2, \dots]$ then $z_j = [w_j, w_{j0}]x'$. We can express the mapping of input feature vector into layer activations as a matrix multiplication with the weight matrix W formed by stacking the neuron weight vectors as columns:

$$\mathbf{a} = \sigma(\mathbf{z} = W\mathbf{x}) \tag{5.2}$$

In feed-forward neural networks layers of neurons are stacked in a sequence in order for the output of the previous layer to be fed as input to the next one. These networks map any input vector (stimulus) to an output vector (response) by feeding them through a number of "hidden layers" as described before (Figure 5.5). In general for the activations at the k-th layer we derive the following formula:

$$a^L = \sigma(z^L) = \sigma(w^L a^{L-1})$$

Each layers weights are fitted to a train set of data vectors in order to learn to make classifications by a process called **backpropagation**. As any other machine learning model, in order to learn to item classification a loss function $l(x)$ needs to be minimized. Assume for now . In order to do this we use a gradient based optimizer to update the weights iteratively. To employ this, we need to first calculate the derivative of the loss function with respect to each layers weights. Suppose that we have an L-layered model with sigmoid activation functions, the derivative of the loss function with respect to the weights of the last layer (output layer) is calculated according to the chain rule as:

$$\frac{\partial l(x)}{\partial W^L} = \frac{\partial a^L}{\partial W^L}\frac{\partial l(x)}{\partial a^L} = \frac{\partial z^L}{\partial W^L}\frac{\partial a^L}{\partial z^L}\frac{\partial l(x)}{\partial a^L}$$

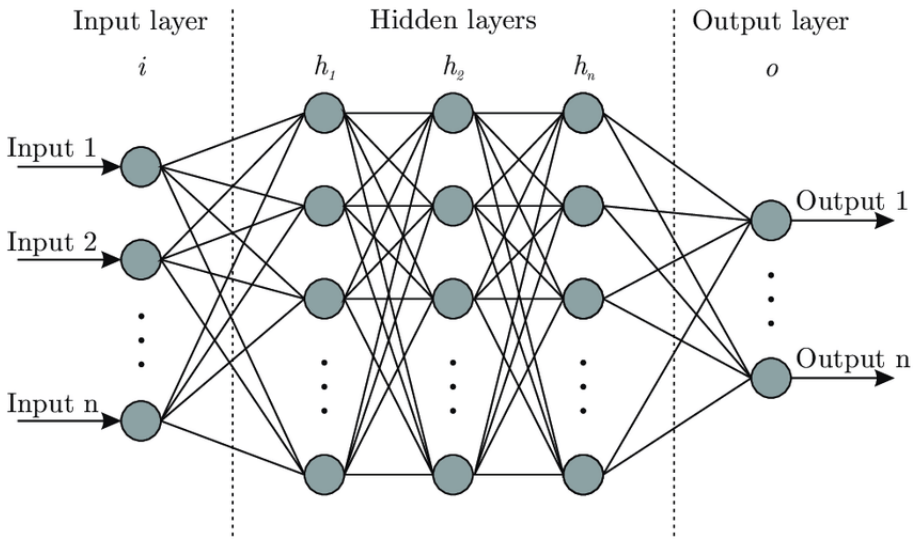

Figure 5.5: Artificial Neural Network Architecture.

If we assume for now that $l(x) = \frac{1}{2N} \sum^{N\_\{i=1\}(a} L - y)^{T(a} L - y)$, meaning our loss function is set to M.S.E. then:

$$\frac{\partial l(x)}{\partial a^L} = (a^L - y)$$

$$\frac{\partial a^L}{\partial z^L} = \sigma'(z^L) = \sigma(z^L)(1 - \sigma(z^L))$$

$$\frac{\partial z^L}{\partial W^L} = a^{L-1}$$

And finaly, putting it all together:

$$\frac{\partial l(x)}{\partial W^L} = a^{L-1}\sigma'(z^L)(a^L - y)$$

In order to propagate this result backwards we merely have to establish that for the k-th layer:

$$\frac{\partial l(x)}{\partial a^k} = \frac{\partial z^{k+1}}{\partial a^{k+1}} \frac{\partial a^{k+1}}{\partial z^{k+1}} \frac{\partial l(x)}{\partial a^{k+1}} = W^k \sigma'(z^{k+1}) \frac{\partial l(x)}{\partial a^{k+1}}$$

And:

$$\frac{\partial l(x)}{\partial W^k} = \frac{\partial z^k}{\partial W^k} \frac{\partial a^k}{\partial z^k} \frac{\partial l(x)}{\partial a^k} = a^k \sigma'(z^k) \frac{\partial l(x)}{\partial a^k}$$

This means that we can progress backwards calculating the gradients for each function by applying a matrix multiplication to the previous layer we calculated. The weights are updated at each step according to the formula:

$$W^k := W^k + \eta \frac{\partial l(x)}{\partial W^k}$$

Where η is the learning rate parameter. By this process we set up a hidden state in our network that allows it to extract hidden features from abstract items of informations and efficiently classify them to the given categories.

In this project we applied MLP, a class of feedforward artificial neural network , to classify data. An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP can easily handle the non-linear and interactive effects of explanatory variables. The major drawback of MLP is the interpretation difficulty.

GridSearchCV tested MLP on the following parameters:

- 'activation': ['tanh', 'relu'],
- 'solver': ['lbfgs','sgd', 'adam'],
- 'hidden layer sizes': [(50,100,50), (100,)]

Where activation is the activation function for the hidden layer: 'tanh' (the hyperbolic tan function) returns $f(x) = tanh(x)$, while 'relu' (the rectified linear unit function) returns $f(x) = max(0, x)$. The solver for weight optimization can be one of three types: 'lbfgs' is an optimizer in the family of quasi-Newton methods, 'sgd' refers to stochastic gradient descent and 'adam' refers to a stochastic gradient-based optimizer [64]. Finally, in hidden layer sizes, the i_{th} element represents the number of neurons in the i_{th} hidden layer.

The parameters selected are activation= 'tanh', alpha= 0.05, hidden layer sizes= (100,) and solver= 'lbfgs' were the optimal parameters for MLP. Some examples of MLP in A.A. are presented in [54, 65, 66, 67, 68]

5.3.5 Naive Bayes

The naive Bayes classifier model derives its name from the 18th century mathematician and theologian Thomas Bayes who formulated the Bayesian probability model that its based on.

Consider a set of object classes $C_j, j = 1, \ldots, n$, given a feature vector of an object $x = (x_1, x_2, \ldots, x_n)$we are interested in estimating the probability that the certain object is contained in any class given the features x_i i.e $P(C_j|x_1, x_2, \ldots, x_n)$,using the bayesian probability rule we calculate the conditional probability as follows:

$$P(C_j|x_1, x_2, \ldots, x_n) = \frac{P(x_1, x_2, \ldots, x_n|C_j)P(C_j)}{P(x_1, x_2, \ldots, x_n)} = \frac{P(x_1, x_2, \ldots, x_n|C_j)P(C_j)}{\sum_i P(x_1, x_2, \ldots, x_n|C_i)P(C_i)}$$

The "naivety" of naive bayes is attributed to the following assumption: we assume that any features x_i, x_j are statistically independant from each other, that is

$$\frac{P(x_1, x_2, \ldots, x_n|C_j)P(C_j)}{\sum_i P(x_1, x_2, \ldots, x_n|C_i)P(C_i)} = \frac{P(x_1|C_j)P(x_2|C_j)\ldots P(x_n|C_j)P(C_j)}{\sum_i P(x_1, x_2, \ldots, x_n|C_i)P(C_i)}$$

The above probability model is combined with a decision rule to make a final decision about the class that should be selected for x to be classified in. One common rule is to pick the hypothesis that is most probable; this is known as the maximum a posteriori or MAP decision rule

$$argmax_{k=1,2,\ldots,n}\{P(C_k)\prod_{i=1}^{n} P(x_i|C_k)\}$$

Training a naive bayes model is closely linked to PDF estimation. The probability densities $P(x_i|C_k)$ are not known quantities and must be approximated. Several kinds of naive bayes classifiers exist, each using a different kind of approximation technique for estimating the underlying pdf. For our model, we settled on multinomial Naive bayes which has been shown to perform well for nlp tasks. This model assumes that the underlying PDF for each class is given by the formula:

$$p(x|C_k) = \frac{\sum_{i=1}^{N} x_i}{\prod_{i=1}^{N} x_i!} \prod_{i=1}^{N} p_{ki}^{x_i}$$

Where p_{ki} are the trainable parameters, corresponding to probabilities of feature (word) i belonging to a document of class k.

According to sklearn documentation, these parameters are estimated during training time using the formula:

$$p_{ki} = \frac{N_{ki} + \alpha}{N_k + n\alpha}$$

Where:

N_{ki}: Is the total count of occurrences of feature i for class k

N_k: Is the total count of all features for class k$

$\alpha \in [0, 1]$: Is Lidstone's smoothing parameter used to avoid overfitting

This makes for a relatively efficient yet simple model with a very fast training time. For this project GaussianNB and MultinomailNB were used. Naive Bayes' classification method for solving A.A. problems are proposed in [52, 21, 30, 69, 70, 71]

5.3.6 Random Forest

Random forests are essentially multiple uncorrelated decision trees, that predict using aggregation in case of regression and majority voting in case of classification. In short, decision trees are built by recursively splitting the data according to a criterion involving its features that minimizes an error function. In order to avoid correlation between trees, each tree is constructed using a random subsample of the original dataset and a random subset of all features as split criteria.

Random forests tend to have better generalization than decision trees because of the bagging concept. On the other hand , both decision trees and random forest have high performance variance depending on the characteristics of the dataset and are susceptible to overfitting , thus need careful hyperparameter tuning (Figure 5.6).

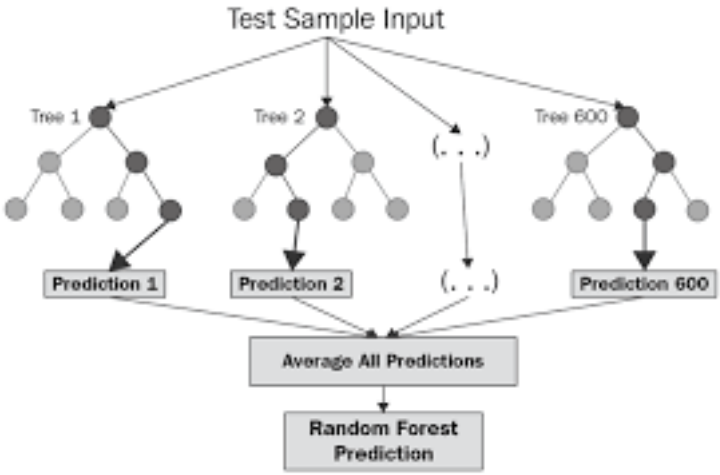

Figure 5.6: Random Forest

The hyperparameters grid tested were:

- 'n estimators': [100, 200, 700, 1000],

- 'max features': ['auto', 'sqrt', 'log2'],
- 'criterion' :['gini', 'entropy'],
- 'n jobs': [-1,1]

The number of trees in the forest is n estimators, and the number of features to examine when looking for the best split is max features. Criterion is the function to measure the quality of a split, while n jobs is the number of jobs to run in parallel. The parameters selected are n estimators=1000, criterion = 'gini', max features = 'auto'and n jobs = -1.

Random Forest is performed in [54, 21, 61, 72, 73, 74] for A.A..

5.4 Results

We tested the different classifiers with various feature extraction methods. The aim was to find which data augmentation (feature selection / feature extraction) techniques benefit the text classification models for the specific dataset and, more importantly, the feature set or the classifier model. We conducted the first experiments t with a reduced dataset of 4500 instances. To avoid the imbalanced dataset that could lead to disputed results, we selected only 100 rows randomly from every author, where each row was composed of 1000 words. Table 5.2 shows the results for the reduced dataset. The first column shows the feature extractors used, the second the classification method, and the third the accuracy extracted with the help of the classification report. TF-IDF was used together with N-grams and the best classifier was MLP with 89% accuracy followed by SVM with

88% accuracy. N-grams were very time-consuming during calculation and often caused allocation problems, especially for the large dataset. The TF-IDF, without combined with other feature extractors, had better results as it achieved accuracies of 92% using SVM and 91% using MLP, respectively. DistilBert and Bert, although we expected to have higher performance due to pre-trained, they managed to achieve accuracies of 74% and 64%, respectively. SVM combined separately with Hashing Vectorizer and N-grams resulted in 83% and 79% accuracy, respectively. Count Vectorizer managed to achieve a 86% score with MLP while only 62% with SVM combined with N-grams. The first impression on the results from Table 5.2 is that N-grams not only increase the accuracy but also slows down the computational process, significantly raising the running time and memory. Moreover, it can be seen

that the best classifiers are SVM and MLP while the best feature extraction technique is TF-IDF.

Feature extraction	Classifier	Accuracy
TF-IDF, N-grams	Random Forest	78%
	SVM	88%
	GaussianNB	83%
	MLP	89%
	XGBoost	73%
TF-IDF	SVM	92%
	XGBoost	71%
	MLP	91%
	Random Forest	70%
	GaussionNB	60%
DistilBert	Logistic Regression	73%
	SVM	73%
	MLP	74%
Bert	Logistic Regression	64%
	SVM	62%
	MLP	64%
HashingVectorizer, N-grams	SVM	79%
HashingVectorizer	SVM	83%
	XGBoost	70%
	Random Forest	71%
CountVectorizer, N-grams	SVM	62%
CountVectorizer	XGBoost	73%
	MLP	86%
	Random Forest	71%
	GaussianNB	58%
	SVM	78%

Table 5.2: Reduced dataset results (4500 instances).

The results for the reduced dataset for POS Tag are shown in Table 5.3.Pipeline consists of POS tag, TF-IDF transformer, and Count Vectorizer. The highest scores obtained are using MLP and SVM classifiers, reaching 92%, while the worst classifier is Decision Tree with 30%. TF-IDF has good accuracy for MLP and SVM while producing almost the same result with XGBoost as the Pipeline. N-grams with TF-IDF and POS Tag surprisingly didn't result in a poor score as expected and managed to maintain the high scores of the two previous feature extractors. Hashing Vectorizer with or without Ngrams didn't manage to have desired scores. Generally, POS Tag is a powerful feature extraction technique reaching high accuracy with different augmented combinations.

Feature extraction	Classifier	Accuracy
Pipeline	MLP	92%
	SVM	92%
	Decision Tree	30%
	Logistic Regression	73%
	XGBoost	69%
POS tag, TF-IDF	MLP	92%
	SVM	91%
	XGBoost	70%
POS tag, TF-IDF, N-grams	MLP	92%
	SVM	90%
	XGBoost	70%
POS tag, HashingVectorizer	MLP	84%
	SVM	83%
	XGBoost	66%
POS tag, HashingVectorizer, N-grams	MLP	75%
	SVM	81%

Table 5.3: Reduced dataset results (4500 instances) with POS Tag.

Taking into consideration the above results, the next step was to apply those feature extraction techniques and classifiers to the full dataset. Due to RAM inadequacy, DistilBert

was tested on 11146 instances. The results are shown in Table 5.4show that MLP reached the highest accuracy with 73% score that is not very satisfying. SVM achieved a 72% accuracy, resulting in a lower score than the one obtained with the smaller dataset of 4500 instances which did not satisfy our expectation of significant improvement. Random Forest reached 58%, GaussianNB 44%, and XGBoost 61% inadequate accuracies, respectively.

Classifier	Accuracy
SVM	72%
MLP	73%
Random Forest	58%
GaussianNB	44%
XGBoost	61%

Table 5.4: DistilBert results for 11146 instances.

The rest of the results for the full dataset are shown in Table 5.5. There is an apparent improvement in the accuracy even though the dataset is imbalanced. TF-IDF, POS Tag combined with TF-IDF, and Pipeline reached a remarkable 98% accuracy with MLP and SVM. TF-IDF combined with N-grams in SVM achieved 96% and in XGBoost 93% accuracies, but further experimentation with the other classifiers was not possible due to memory errors. Although Hashing Vectorizer didn't have the best performance in the reduced dataset, in full dataset it managed to reach 96% with SVM classifier and 91% with XGBoost. N-grams with Hashing Vectorizer raised memory errors thus, the computation was not possible to be performed. In Count Vectorizer with XGBoost, managed to achieve a 92% accuracy, the best result obtained with this classifier through the experiments. SVM had a great performance as it reached 96% accuracy and ranked the best classifier of Count Vectorizer, followed by MLP with 95% score. Random Forest didn't manage to have an increase in score while GaussianNB reached a disappointing 63% accuracy. However, combined with Ngrams the result is still not satisfying as SVM reached only 85% accuracy. In Pipeline, which consists of POS tag, TFIDF transformer, and Count Vectorizer, the best classifier was found to be SVM followed again by MLP achieving 98% and 97% scores, respectively. Logistic Regression noted a great score improvement as it achieved 92% while XGBoost 90%, correspondingly. Decision Tree and MultinomialNB didn't manage to improve their performance significantly.

Feature extraction	Classifier	Accuracy
TF-IDF	MLP	98%
	SVM	98%
	XGBoost	91%
TF-IDF, N-grams	SVM	96%
	XGBoost	93%
Hashing Vectorizer	XGBoost	91%
	SVM	96%
	Random Forest	58%
POS tag, TF-IDF	MLP	97%
	SVM	98%
	XGBoost	90%
	GaussianNB	58%
	Random Forest	65%
POS, Hashing Vectorizer	SVM	96%
	MLP	91%
	XGBoost	90%
Count Vectorizer, N-grams	SVM	85%
Count Vectorizer	MLP	95%
	SVM	96%
	Random Forest	69%
	GaussianNB	63%
	XGBoost	92%
Pipeline	MLP	97%
	SVM	98%
	Decision Tree	46%
	Logistic Regression	92%
	XGBoost	90%
	MultinomialNB	55%

Table 5.5: Full dataset results.

Chapter 6

Conclusions

This Chapter summarizes the results indicating the best feature extraction method and the optimal classification technique and answers the questions raised in the introduction. Furthermore, we will present some ideas for future work.

6.1 Summary

After experimenting and based on the results obtained, the best accuracies were achieved by: TF-IDF on MLP and SVM classifiers, POS Tag combined with TF-IDF on SVM and the Pipeline, which consists of POS tag, TF-IDF transformer, and CountVectorizer, on SVM. If we consider the computational complexity of the algorithms, TF-IDF is the least time-consuming feature set as the calculation time was only a few minutes while POS tag consumed 7 hours CPU time. It appears that the best feature extraction technique that benefits the text classification models for this specific dataset is TF-IDF. N-grams was found to be the most inefficient method when combined with feature extractors as not only did they not improve the accuracy but also caused a remarkable increase in wall time and several memory error problems. Little improvement was noticed in feature sets when they were augmented. To reduce computational time, it is preferable to use feature extractions solely and not combined with other techniques. SVM managed to receive the highest accuracy through all of the feature sets, achieving from 96% to 98% scores except for Count Vectorizer combined with N-grams that reached 85% accuracy. Hence, we could conclude that the model is more important than the feature set. If a feature set was randomly selected, SVM would produce more or less the same accuracy, but if a classifier was chosen randomly, the result would

have a remarkable change.

The dataset size plays a significant part in high accuracy. Almost every classifier was notably improved when we increased the dataset from 4.500 to 53.678 instances, with the most considerable difference happening in XGBoost with a 20% score rise.

6.2 Future Work

A future work direction is to use finetuning on Bert and DistilBert models and to study the effect of extending finetuning to language model layers and focus on the different layers of the language modeling representation proposed in [22]. Moreover, [75] introduces GTP-3, a pre-trained model for Q.A. answering. It would be interesting to see whether this model can expand to A.A. tasks and outperform fine-tuning approaches. Furthermore, we could explore other classification methods like unsupervised learning techniques, e.g., KNN, or automated machine learning classifiers,e.g., AutoML. The optimal feature set and classifier derived from this dataset could be used to other corpus to identify optimal machine learning models for A.A. tasks.

Bibliography

[1] Han Van der Aa. Comparing and aligning process representations. In *BPM (Dissertation/Demos/Industry)*, pages 16–20. Springer, 2018.

[2] Mutian He, Yangqiu Song, Kun Xu, and Dong Yu. On the role of conceptualization in commonsense knowledge graph construction. *arXiv preprint arXiv:2003.03239*, 2020.

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

[4] Harold Love. *Attributing authorship: An introduction*. Cambridge University Press, 2002.

[5] MS Windows NT pseudepigrapha. `https://en.wikipedia.org/wiki/Pseudepigrapha`. Accessed: 2010-09-30.

[6] R John Leigh, John Casson, and David Ewald. A scientific approach to the shakespeare authorship question. *SAGE Open*, 9(1):2158244018823465, 2019.

[7] Refat Aljumily. Hierarchical and non-hierarchical linear and non-linear clustering methods to "shakespeare authorship question". *Social Sciences*, 4(3):758–799, 2015.

[8] William Leahy. *Shakespeare and his authors: critical perspectives on the authorship question*. A&C Black, 2010.

[9] Scott McCrea. *The case for Shakespeare: The end of the authorship question*. Greenwood Publishing Group, 2005.

[10] Thomas VN Merriam and Robert AJ Matthews. Neural computation in stylometry ii: An application to the works of shakespeare and marlowe. *Literary and Linguistic Computing*, 9(1):1–6, 1994.

[11] Robert Fowler. The homeric question. *The Cambridge Companion to Homer*, pages 220–32, 2004.

[12] E Suzan Sherratt. 'reading the texts': archaeology and the homeric question. *Antiquity*, 64(245):807–824, 1990.

[13] Minna Skafte Jensen. *The Homeric question and the oral-formulaic theory*. Museum Tusculanum Press, 1980.

[14] Milman Parry and Adam Parry. *The making of Homeric verse: The collected papers of Milman Parry*. Oxford University Press on Demand, 1987.

[15] MS Windows NT homeric question. `https://en.wikipedia.org/wiki/Homeric_Question`. Accessed: 2010-09-30.

[16] Moshe Koppel, Jonathan Schler, and Shlomo Argamon. Authorship attribution: What's easy and what's hard? *SSRN Electronic Journal*, 21, 06 2013.

[17] Anderson Rocha, Walter J Scheirer, Christopher W Forstall, Thiago Cavalcante, Antonio Theophilo, Bingyu Shen, Ariadne RB Carvalho, and Efstathios Stamatatos. Authorship attribution for social media forensics. *IEEE transactions on information forensics and security*, 12(1):5–33, 2016.

[18] Lawrence M Solan. Intuition versus algorithm: The case of forensic authorship attribution. *JL & Pol'y*, 21:551, 2012.

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

[20] Farkhund Iqbal, Rachid Hadjidj, Benjamin CM Fung, and Mourad Debbabi. A novel approach of mining write-prints for authorship attribution in e-mail forensics. *digital investigation*, 5:S42–S51, 2008.

[21] Nizar Hirzalla. *Automating Authorship Attribution in Heterogeneous and Sparse Publication Data through Supervised Machine Learning*. PhD thesis, Vrije Universiteit Amsterdam, 2020.

[22] Georgios Barlas and Efstathios Stamatatos. Cross-domain authorship attribution using pre-trained language models. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 255–266. Springer, 2020.

[23] Petr Plecháč. Relative contributions of shakespeare and fletcher in henry viii: An analysis based on most frequent words and most frequent rhythmic patterns. *Digital Scholarship in the Humanities*, 2019.

[24] Robert AJ Matthews and Thomas VN Merriam. Neural computation in stylometry i: An application to the works of shakespeare and fletcher. *Literary and Linguistic computing*, 8(4):203–209, 1993.

[25] Ryan L Boyd and James W Pennebaker. Did shakespeare write double falsehood? identifying individuals by creating psychological signatures with text analysis. *Psychological science*, 26(5):570–582, 2015.

[26] José Nilo G Binongo. Who wrote the 15th book of oz? an application of multivariate analysis to authorship attribution. *Chance*, 16(2):9–17, 2003.

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

[28] Tālis Putniņš, Domenic J Signoriello, Samant Jain, Matthew J Berryman, and Derek Abbott. Advanced text authorship detection methods and their application to biblical texts. In *Complex Systems*, volume 6039, page 60390J. International Society for Optics and Photonics, 2006.

[29] Lubomir Ivanov. Using alliteration in authorship attribution of historical texts. In *International Conference on Text, Speech, and Dialogue*, pages 239–248. Springer, 2016.

[30] Opeyemi Aborisade and Mohd Anwar. Classification for authorship of tweets by comparing logistic regression and naive bayes classifiers. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 269–276. IEEE, 2018.

[31] Julio CS Reis, André Correia, Fabrício Murai, Adriano Veloso, and Fabrício Benevenuto. Supervised learning for fake news detection. *IEEE Intelligent Systems*, 34(2):76–81, 2019.

[32] Dan Jurafsky. *Speech & language processing*. Pearson Education India, 2000.

[33] 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.

[34] Grigori Sidorov, Francisco Velasquez, Efstathios Stamatatos, Alexander Gelbukh, and Liliana Chanona-Hernández. Syntactic n-grams as machine learning features for natural language processing. *Expert Systems with Applications*, 41(3):853–860, 2014.

[35] Sawinder Kaur, Parteek Kumar, and Ponnurangam Kumaraguru. Automating fake news detection system using multi-level voting model. *Soft Computing*, 24(12):9049–9069, 2020.

[36] Jeremy West, Dan Ventura, and Sean Warnick. Spring research presentation: A theoretical foundation for inductive transfer. *Brigham Young University, College of Physical and Mathematical Sciences*, 1(08), 2007.

[37] Munazza Zaib, Quan Z Sheng, and Wei Emma Zhang. A short survey of pre-trained language models for conversational ai-a new age in nlp. In *Proceedings of the Australasian Computer Science Week Multiconference*, pages 1–4, 2020.

[38] MS Windows NT transfer learning. `https://machinelearningmastery.com/transfer-learning-for-deep-learning/`. Accessed: 2021-06-30.

[39] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[40] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[41] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[42] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[43] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

[44] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works. *Transactions of the Association for Computational Linguistics*, 8:842–866, 2020.

[45] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[46] Prakhar Ganesh, Yao Chen, Xin Lou, Mohammad Ali Khan, Yin Yang, Deming Chen, Marianne Winslett, Hassan Sajjad, and Preslav Nakov. Compressing large-scale transformer-based models: A case study on bert. *arXiv preprint arXiv:2002.11985*, 2020.

[47] MS Windows NT cross validation. `https://machinelearningmastery.com/k-fold-cross-validation/`. Accessed: 2021-06-30.

[48] MS Windows NT svm. `https://scikit-learn.org/stable/modules/svm.html`. Accessed: 2021-06-30.

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

[50] Paulo Varela, Edson Justino, and Luiz S. Oliveira. Selecting syntactic attributes for authorship attribution. In *The 2011 International Joint Conference on Neural Networks*, pages 167–172, 2011.

[51] Siham Ouamour and Halim Sayoud. Authorship attribution of ancient texts written by ten arabic travelers using a smo-svm classifier. In *2012 International Conference on Communications and Information Technology (ICCIT)*, pages 44–47, 2012.

[52] Ilker Nadi Bozkurt, Ozgur Baghoglu, and Erkan Uyar. Authorship attribution. In *2007 22nd international symposium on computer and information sciences*, pages 1–5, 2007.

[53] Petr Plecháč, Klemens Bobenhausen, and Benjamin Hammerich. Versification and authorship attribution. a pilot study on czech, german, spanish, and english poetry. *Studia Metrica et Poetica*, 5(2):29–54, 2018.

[54] Dainis Boumber, Yifan Zhang, and Arjun Mukherjee. Experiments with convolutional neural networks for multi-label authorship attribution. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[55] Suleyman Alterkavı and Hasan Erbay. Design and analysis of a novel authorship verification framework for hijacked social media accounts compromised by a human. *Security and Communication Networks*, 2021, 2021.

[56] Tieyun Qian, Bing Liu, Li Chen, and Zhiyong Peng. Tri-training for authorship attribution with limited training data. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 345–351, 2014.

[57] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21, 2013.

[58] MS Windows NT xgboost. `https://xgboost.readthedocs.io/en/latest/`. Accessed: 2021-06-30.

[59] Chanchal Suman, Ayush Raj, Sriparna Saha, and Pushpak Bhattacharyya. Source code authorship attribution using stacked classifier. In *FIRE (Working Notes)*, pages 732–737, 2020.

[60] Suleyman Alterkavı and Hasan Erbay. Novel authorship verification model for social media accounts compromised by a human. *Multimedia Tools and Applications*, 80(9):13575–13591, 2021.

[61] Nickolay Viuginov, Petr Grachev, and Andrey Filchenkov. A machine learning based plagiarism detection in source code. In *2020 3rd International Conference on Algorithms, Computing and Artificial Intelligence*, pages 1–6, 2020.

[62] José Eleandro Custódio and Ivandré Paraboni. Each-usp ensemble cross-domain authorship attribution. In *Working notes of CLEF 2018—conference and labs of the evaluation forum*, 2018.

[63] Darren M Vescovi. *Best practices in authorship attribution of English essays*. PhD thesis, Duquesne University, 2011.

[64] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[65] Nilan Saha, Pratyush Das, and Himadri Nath Saha. Authorship attribution of short texts using multi-layer perceptron. *International Journal of Applied Pattern Recognition*, 5(3):251–259, 2018.

[66] Nikos Tsimboukakis and George Tambouratzis. A comparative study on authorship attribution classification tasks using both neural network and statistical methods. *Neural Computing and Applications*, 19(4):573–582, 2010.

[67] Shanta Phani, Shibamouli Lahiri, and Arindam Biswas. A machine learning approach for authorship attribution for bengali blogs. In *2016 International Conference on Asian Language Processing (IALP)*, pages 271–274. IEEE, 2016.

[68] Urszula Stańczyk. The class imbalance problem in construction of training datasets for authorship attribution. In *Man–Machine Interactions 4*, pages 535–547. Springer, 2016.

[69] Alaa Saleh Altheneyan and Mohamed El Bachir Menai. Naïve bayes classifiers for authorship attribution of arabic texts. *Journal of King Saud University-Computer and Information Sciences*, 26(4):473–484, 2014.

[70] Grzegorz Baron. Influence of data discretization on efficiency of bayesian classifier for authorship attribution. *Procedia Computer Science*, 35:1112–1121, 2014.

[71] Rosa María Coyotl-Morales, Luis Villaseñor-Pineda, Manuel Montes-y Gómez, and Paolo Rosso. Authorship attribution using word sequences. In *Iberoamerican Congress on Pattern Recognition*, pages 844–853. Springer, 2006.

[72] Promita Maitra, Souvick Ghosh, and Dipankar Das. Authorship verification-an approach based on random forest. *arXiv preprint arXiv:1607.08885*, 2016.

[73] Sreenivas Mekala, Raghunadha Reddy Tippireddy, and Vishnu Vardhan Bulusu. A novel document representation approach for authorship attribution. *International Journal of Intelligent Engineering and Systems*, 11(3):261–270, 2018.

[74] Bander Alsulami, Edwin Dauber, Richard Harang, Spiros Mancoridis, and Rachel Greenstadt. Source code authorship attribution using long short-term memory based networks. In *European Symposium on Research in Computer Security*, pages 65–82. Springer, 2017.

[75] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.