Utilising a Dataset for Two Text Classification Tasks: Sentiment Analysis and Author Identification

Thesis submitted in partial fulfillment of the requirements of the degree of

Masters of Science
in
Computational Linguistics by Research

by
Ojaswi Binnani
20161006

International Institute of Information Technology
Hyderabad - 500032, India
January 2023

International Institute of Information Technology, Hyderabad

Certificate

It is certified that the work contained in this thesis, titled **"Utilising a Dataset for Two Text Classification Tasks: Sentiment Analysis and Author Identification"** by **Ojaswi Binnani**, has been carried out under my supervision and is not submitted elsewhere for a degree.

date	Adviser: Prof. Radhika Mamidi

Dedication

To my family who provided me endless support to complete this. To Radhika ma'am who was always there for me during this entire process. To the pandemic for taking distractions away.

Abstract

The Internet has many tools that connect people from different ends of the globe. Geographical boundaries do not hinder the connection between two people. Communication is easily facilitated, and news is easily spread. However, this connectivity can be exploited and used negatively.

Bullying, hate speech, and fake news can all be spread across the internet, and the common denominator in combating these issues is Sentiment Analysis. Sentiment Analysis is the task of finding the sentiment (positivity or negativity) of a text. This process can be an essential feature for bullying detection, hate speech detection, and fake news detection models. However, there are many libraries and functions available to use, and choosing the optimal one that has the most accuracy to use as a feature for other Natural Language Processing (NLP) tasks is essential. During our search to find functions that calculate sentiment, otherwise known as pre-trained sentiment models, we find that most of the models are trained on polarised texts and datasets that already have sentiment annotated. Hence, we use a non-polarised text form: news articles and we use a dataset that is not meant to be used with Sentiment Analysis. This helps us test different models. Through our experiments we find that VADER, Stanza, and Transformer's distilBERT work with a good accuracy, and are ideal to use as the basis for calculating a sentiment feature whereas lexical-based models such as TextBlob and using SentiWordNet are less accurate in its calculations.

The dataset we use is the Reuter_50_50 dataset that is meant to be used for another NLP task: Author Identification/Author Profiling. [60] Working with the dataset led to our second task: to create a model that works with minimal requirements for data and computational power but still gives a comparable accuracy to other deep learning methods of Author Identification. Author Identification is that task of being given a random text that could have been written by a group of suspect authors, and finding with rea-

sonable accuracy, which author wrote the text. This is useful to stop other people taking credit for works they have not written and giving credit to the rightful author. We find that using a mixture of word-based features, stylometric feature, and syntax/punctuation features are good features to train a model for Author Identification. We also find that the Boosting class of algorithms outperforms other classes such as Linear models, Nearest Neighbour-Based Models, and Tree Models. XGBoost performs the best out of all the algorithms we test. Linear Models such as SVMs, the Naive Bayes, and the K-Nearest Neighbour algorithm perform abysmally.

Contents

Chapter 1

Introduction

1.1 Aim

For sentiment analysis, we aim to compare and find the most accurate pre-trained sentiment model while using news articles. All of the models are trained and tested with polarised text forms such as reviews, so using news articles shows the sensitivity of the models. We also draw a correlation between the sentence-level sentiment of the first sentence and document-level sentiment.

For Author Identification, we aim to train a traditional machine learning model that works with comparable accuracy to other deep learning based models without the disadvantages of a deep learning model. We aim to find features that can be good differentiators between authors.

1.2 Motivation

Sentiment may be used as a feature to improve models that detect bullying, racism, sexism, hate speech, etc and can be used positively for stock prediction models and consolidation reviews for products and services. Sentiment is also the basis for other NLP tasks such as sarcasm detection and sentiment reasoning. Additionally, when one does not have enough data/ annotated data, or have the time to train a model from scratch for sentiment analysis, so using a pre-trained sentiment analysis model is ideal.

Author Identification is useful for giving the correct author of a given work due credit. It can be used to prevent plagiarism since every person

has their own unique style of choosing vocabulary and structuring sentences, paragraphs, and documents. Additionally, Author Identification can be used in Forensic Linguistics such as the case against the Unabomber (Section 2.2.1). Some Author Identification research papers we came across either have a methodology for Author Identification without proof or results of the working of the proposed model, or a model that is a computationally expensive model such as deep learning model LSTMs, and require a lot of data to make the model accurate. Our model works by using features to overcome the disadvantages of traditional machine learning models. Our model does not require as much data or time as the deep learning methods.

1.3 Key Contributions

Our key contributions are as follows:

- A dataset consisting of over 70,000 words was annotated for sentiment into Positive and Negative. These annotations are used to compare the accuracy of all the pre-trained sentiment models.
- Experiments were conducted using a variety of different available pre-trained sentiment models from python libraries that use different algorithms - from rule-based approaches to deep learning approaches - to find a pre-trained sentiment model that works with the highest accuracy for news articles.
- A novel idea that the sentiment of the first sentence has a strong correlation with the sentiment of the text, and can be used to accurately determine the sentiment of the whole document was presented.
- Features that can be easily computed and do not require human annotation are experimented with to find a set of features that can enhance a traditional machine learning model to work with similar or better accuracy than deep learning methods of Author Identification.
- A set of experiments was conducted with different types of traditional machine learning models viz. Linear models, Nearest-Neighbour based models, Tree-based models, and Boosting models. Within each group, multiple algorithms were tested to find the algorithms and category that are most effective in Author Identification with the chosen feature

set. A conclusion was drawn that the XGBoost algorithm within the Boosting category is the most effective amongst all other algorithms and is comparable to deep learning methods of Author Identification.

1.4 Overview

The thesis is organised into six chapters:

Chapter 2: Related Work and Case Studies In this chapter, we discuss work that has been done previously in the fields of Sentiment Analysis and Author Identification. We also look at some case studies of Author Identification being used such as the case against the Unabomber, the authorship of the disputed Fedaralist Papers, and the authorship of the book Wuthering Heights.

Chapter 3: Dataset In this chapter, we discuss the dataset we used for both Sentiment Analysis and Author Identification tasks. We discuss the reasons for choosing the dataset and the modifications we made to suit our experiments.

Chapter 4 In this chapter, we discuss our study of pre-trained sentiment analysis models and the comparison between sentence-level sentiment and document-level sentiment. We show the models we test and the working behind the models followed by an analysis of why they performed the way they did. We also conclude which models are ideal to use when one needs the sentiment of a text without wanting to train a model.

Chapter 5 In this chapter, we discuss the features that can be used to differentiate between authors and can be used to improve the working of a traditional machine learning model. We look at three different types of features: Word-Based, Stylometric, and Syntax-Based, and choose at least one from each category. Additionally, we examine the algorithms we experiment with to find the one with the best accuracy. We also try different types of Machine Learning models: Linear, Nearest Neighbour Based, Tree Based, and Boosting. We conclude with the features that are important and the model that produces maximum accuracy.

Chapter 6 In this chapter, we recap the conclusions of the previous two chapters and look at future directions possible in these fields.

Chapter 2

Related Work

2.1 Sentiment Analysis

Nguyen et al. [54] performs aspect-level sentiment analysis on news articles. They conclude that using a neural network leads to overfitting, so they create a new approach to finding sentiment of news articles. This approach consists of using a word embedding model, an LSTM model and a Convolutional Neural Network (CNN).

The main lexical-based approach is to pre-process the text, assign sentiments to the words, average it and predict the sentiment. We do this method in section 4.1. This method is used by citetstock2014, shah.Kalyanaraman et al. [34] finds the sentiment for stock news articles. They created a sentiment dictionary that has accurate sentiment values for words used in news articles about stocks. Shah et al. [66] finds the relation between the sentiment of news articles and the short-term prediction of the stock price. They use a lexical-based approach and create a dictionary of words used in the financial market.

Derakhshan and Beigy [17] tests multiple methods of predicting stock movement on social media interaction including a human-based sentiment method and an aspect-based sentiment method.

Multiple machine learning models including maximum entropy classifier, support vector machines, bagging, boosting, random forest and decision trees are tested for predicting sentiment on polarised texts.[33] The data is amazon reviews, yelp reviews and imdb reviews. Text pre-processing is done on the datasets, the words are used as features for the various machine learning

models.

2.2 Author Identification

Khan et al. [36] discussed the differences between the three tasks in author analysis - one of which is author identification. The paper discusses different types of features and types of models used in author identification and proposes a method to identify an author of an email.

Zheng et al. [78] looked at authorship identification in online messaging. One of the languages they looked at was English and came up with 270 features categorized into five types: lexical features, word-based features (not similar to our word-based feature classification in section 5), syntactic features, structural features and content specific features. Using these 270 features, Li et al. [42] found that a Support Vector Machine (SVM) outperformed the other models. This proves that a traditional model with sufficient meaningful features can be used in author identification. Mohsen et al. [52]; De Vel [16] also used SVMs for author idenitication.

Zhou and Wang [79]; Qian et al. [60] worked with three stylometric features: average sentence length, average word length and Hapax Legomenon Ratio and applied Glove Vector Embeddings to news articles. They worked with RNN and LSTM respectively. Their models suffered from an overfitting problem due to the computational complexity of their models and their insufficient dataset.

Iqbal et al. [29] has a list of features, and for any two authors has a subset of features that are applicable to one author and not the other. They call this the write print between two authors, and use these features to classify the author of a malicious email.

2.2.1 Author Identification Case Studies

Theodore John Kaczynski (Ted Kaczynski) - Forensic Linguistics

Ted Kacynski was a former mathematics professor who left his academic career in favor of a primitive lifestyle without electricity or running water in 1971. He had a huge hatred towards industrialisation and used terrorism to fight against it. In 1978, he began sending mail bombs to locations that he believed to be supporting industrialisation. The FBI referred to the terrorist and UNABOM - university and airline bomber. This caused the media to

refer to him as the Unabomber. During the course of the FBI's investigation, they were led with many false clues that the Unabomber purposefully left in the mail packages and a lack of real clues to lead to the terrorist. In 1995, the Unabomber approached several big newspapers to print his essay titled Industrial Society and Its Future (also known as the Unabomber manifesto). Upon publishing, the FBI asked the public for tips about who the author of the manifesto could be and found an essay written by Ted Kacynski in 1971. FBI profiler James R Fitzgerald compared the essay and the manifesto and found linguistic similarities between both. While reading the essay, he came across a phrase "the sphere of freedom", a phrase that was also present in the manifesto. Ted Kaczynski's brother David Kacynski then came forward with the letters that Ted had sent him, so the FBI were able to match the timeline of all of Kaczynski's activities. They found that the bomb packages and the letters sent to his brother followed a similar timeline. The similarities in the texts was the basis for the search warrant for Kaczynski's cabin, and the agents were able to arrest Kaczynski in his cabin in 1996. In 1996, his lawyers attempted to use an insanity defense but Kazynski refused this defense and changed lawyers. He was found fit to stand trial and was convicted of ten counts of illegally transporting, mailing and using bombs in 1998. He had pleaded guilty and took a plea bargain accepting eight life sentences without the possibility of parole to avoid the death sentence.[69; 68; 30; 73; 62]

Authorship of the Fedaralist Papers

From 1787 to 1788, a person or some people used the pseudonym Publius to write 77 articles around 900-3500 words for a variety of Newspapers to convince the US citizens to ratify the US constitution after some articles that opposed the proposition. The authors were Alexander Hamilton, James Madison and John Jay. They had tried to hide their identities since they were at the convention where the constitution was proposed. However, some observers correctly identified their identities. The 77 articles along with another 8 articles were published into a book called *the Federalist*. Amongst those articles, there was a consensus that Jay wrote 5 articles, Hamilton was the sole author of 51 articles, Madison was the sole author of 14 articles and that they collaborated on 3 articles. The remaining twelve articles are written by either Hamilton or Madison, but there was no consensus about which were written by whom. The articles are referred to as the *disputed papers*. This dispute exists because after Hamilton's death, Hamilton's lawyer had

a list which credited 63 papers to Hamilton (including the 3 he wrote with Madison). However, it has a mistake which credited article no.54 to Jay, though the fact is that Jay wrote no.64 instead. Madison disputed the list and claimed 29 essays to himself.

Through stylometric analysis of the disputed papers, it is widely believed that the twelve disputed papers were written by Madison. [53] concluded: *"In summary we can say with better foundation than ever before that Madison was the author of the 12 disputed papers"*. They also discussed the differences in Hamilton's and Madison's writing style. For example, Madison used the word "whilst" while Hamilton used "while". Also, Madison used the word "by" much more frequently than Madison. [8] calculated a hyperplane on function word features that separates all of Hamilton's and Madison's works and found that all twelve of the disputed papers belonged on Madison's side of the hyperplane. [23] also found that the author of all twelve disputed papers were written by Madison.

Authorship of Wuthering Heights

Wuthering Heights was a book that was published in 1847 under a pseudonym of Ellis Bell. It could be classified as a dark romance with highly unlikable and selfish characters. In the second edition, the book was published under the real author's name: Emily Brontë. She chose to write under a pseudonym to give her book a chance to succeed in Victorian times as female authors were not taken seriously. After her death, her sister Charlotte Brontë credited the book to Emily. Though, throughout the years to come, the authorship of the book came into question multiple times. In 1911, John Malham-Dembleby claimed that the author was not Emily, rather it was Charlotte and she was not able to claim authorship due to politics with the publisher. In 1942, Alice Law (in her book *Patrick Branwell Brontë*) claimed Emily's brother Branwell was the author of the book simply because she believed a woman in Victorian times could not come up with and write such a story. Grundy, a friend of Branwell's, wrote in his book *Pictures of the Past "...well-nigh incredible that a book so marvellous in its strength, and in its dissection of the most morbid passions of diseased minds, could have been written by a young girl like Emily Brontë, who never saw much of the world or knew much of mankind"*. In 1867, the Halifax Guardian published a letter written by William Deardon (under a pseudonym William Oakendale) claiming that he saw similarities between Wuthering Heights and works written by Branwell.

He claimed that he knew Branwell and they sent each other poetry verses and he saw common themes between his poetry and the book. He also claimed that it was impossible that a shy, antisocial girl such as Emily would be able to write this book. Notably the questioning around the authorship of Wuthering Heights started with Branwell's friends, especially with Deardon's article. [39; 26; 10; 13; 48; 49; 51]

McCarthy and O'Sullivan [48] confronts the issue of who wrote Wuthering Heights. They acknowledge that by doing a rolling stylometry analysis of Wuthering Heights compared to Branwell's unfinished novel and texts written by Emily Brontë, Branwell comes as the likely author of the book. However, the paper attributes this result to the lack of data available for Emily. The written pieces that exist for Emily apart from the book itself are poetry. Poetry is not used for authorship identification. They claim that the written texts are not suitable for the author identification task. Additionally, they claim that Branwell's unfinished novel is a piece of fiction and hence going to be more similar in style to Wuthering Heights. To dismiss Branwell as the author or contributor in Wuthering Heights, stylometric analysis is done with Emily's sisters: Charlotte and Anne. Charlotte and Anne both have a better dataset for author identification, and after doing analysis, the authorship style matches with Charlotte and Anne much more than it does with Branwell. "In other words, Branwell's style only shows up in Wuthering Heights when competing with work by Emily which cannot be considered suitable for analysing the likely authorship of a piece of fiction"[48]. This study does not dissuade the possibility that Charlotte wrote Wuthering Heights, but most critics support the idea that Branwell was the true author of the book. Also, it was Charlotte herself who credited the book to Emily Brontë after her death.

Chapter 3

Dataset

For both of the tasks, we use the Reuter_50_50 dataset. This dataset was developed by ZhiLiu in 2011 and has been used in Author Identification experiments.

It is a subset of Reuters Corpus Volume 1 (RCV1)[41] dataset. RCV1 is a dataset that was made available by Reuters Ltd. from English news articles published in Reuters news agency from August 1996 to August 1997. The articles were annotated with three different categories: Topics, Industries and Regions. The topics were CCAT(corporate/industrial), ECAT (economics), GCAT (government/social) and MCAT(markets).

Reuters_50_50 dataset [60] used news articles from the CCAT (corporate/industrial) topic to make sure that a topic change would not impact the working of the author identification model. This dataset has been used in many Author Identification experiments and we perform our own experiments on it (Chapter 5).

There are 5,000 articles in total, with half in the testing set and half in the training set. There are 50 authors in total, and each author has 50 articles in the testing set, and 50 in the training set. There is no overlap between the articles in the testing and training set.

For each author, the number of words per article ranges from 468 to 569 words, averaging around 512 words per article. The sizes of the articles range from 2.7 kB to 3.6kB, averaging at 3.13 kB per article.

This dataset was used for testing sentiment analysis pre-trained models (Chapter 4) because the dataset has not been used in sentiment analysis. It was important to choose a dataset that has not been used to train any of the pre-trained sentiment models, and this one was created for Author

Identification and had no sentiment annotation. We annotated 150 articles from the dataset into Positive and Negative, and used the annotated articles to test the various pre-trained models as well as test sentence-level versus document-level sentiment.

This dataset was combined and then redistributed into a 90-10 split for training to testing for the Author Identification experiments (Chapter 5).

	Testing Data	**Training Data**
Number of Authors	50	50
Original Number of Articles	2500	2500
Topic	Corporate/Industrial	
Average Words	512 words	
Average Size	3.13kB	

Table 3.1: Dataset values

Chapter 4

A Comparative Study of Pre-trained Sentiment Analysis Models and Sentence-Level and Document-Level Sentiment

4.1 Introduction

Sentiment Analysis or *Opinion Mining* is the process of evaluating a text as positive or negative. It is a useful task in Natural Language Processing (NLP) since it can speed up the process of getting the overarching opinion of a text or series of texts.

Additionally, sentiment analysis is the stepping stone to many other useful NLP tasks that can prevent unsavory language such as bullying detection and hate speech detection, or can be used in prediction in other domains such as stock movement and social media trends. It is also the basis of sentiment reasoning (finding the reason behind the sentiment) and sarcasm analysis.

In certain scenarios, the sentiment should be calculated with minimum effort. This could be because the sentiment is being used as a feature in an another task or the amount of data that needs analysis is lower or is not annotated. Training a model in these scenarios is unfeasible, so a pre-trained sentiment model can be used to do the analysis. In this task, we compare several easily available pre-trained sentiment models in predicting the sentiment of news articles. Most of these models are trained over polarised texts

such as reviews, so news articles can show the sensitivity of the model. This is discussed further in section 4.3. We also aim to find if there is a correlation between the sentiment of the first sentence of the article and the sentiment of the overall article. We discuss our hypothesis further in section 4.2.

4.2 Sentence-Level vs Document-Level

According to Liu [43], sentiment analysis is in three levels: sentence-level, document-level and aspect-level. Sentence-level is doing sentiment analysis on each sentence, document-level is doing sentiment analysis on the whole document. While reading some of the documents in the dataset, it seems that news articles have a format that is mostly followed. The hypothesis is that while writing a news article, the writer expresses a sentiment in the first sentence, shows opinions or facts agreeing with that sentiment and also shows the opposing view. This means that if the overall article is positive, there may be the first sentence showing a positive sentiment, then positive facts or quotes while also showing the negative side of the news. Hence, the first sentence's sentiment can be used to predict the sentiment of the whole article. To test this hypothesis, we apply the sentiment analysis models to the whole document as well as to the first sentence, and test which of the two may be more accurate in predicting the sentiment of the whole document.

4.3 Pre-trained Models

In different areas of Natural Language Processing (NLP), sentiment analysis of the dataset can be used as a feature for the classification problem. Tasks in which sentiment analysis can be used as a feature can be hate speech detection[5], bullying detection [76], stock forecasting [72; 75], etc. In these scenarios, the features should be easy to obtain and be accurate. Rather than the researcher having to annotate their dataset, gather a set of features, train multiple models to obtain reasonable accuracy and use precious time to be able to use sentiment as a feature, this task aims to find a pre-trained sentiment model that can be used that results in reasonable accuracy regardless of the fact that the model is not trained on the dataset. Additionally, these pre-trained models can be used even with a small dataset, since it is not being used in the training stage.

According to Poria et al. [57], the past work of sentiment analysis can be categorized into Rule-Based, Lexical-Based, Machine Learning and Deep Learning. Except for Machine Learning (Naivye Bayes, SVMs, etc), we test at least one pre-trained sentimenet analysis model from each category.

4.3.1 Vader

VADER [28] is an in-built function in the NLTK Library in Python. It stands for Valence Aware Dictionary for sEntiment Reasoning. It was a human-based approach to create a dictionary that has sentiment scores to certain linguistic features on a scale of -4 (extremely negative) to +4 (extremely positive) where 0 was neutral. They had a team of human evaluators who had to prove their English capabilities through a series of tests and then get a series of linguistic features and assign a valence score to them. These features were words, acronyms, slang and emoticons.

Then, two experts assign a 400 positive and 400 negative tweets scrutinizing at other characteristics in the text such as:

- the usage of punctuation *e.g "The food was great!"* vs *"The food was great."*,
- specific capitalisation *e.g "The food was GREAT"* vs *"The food was great"*,
- modifiers *e.g. "The food was extremely good"* vs *"The food was good"*,
- contrastive conjunction which signifies a change in polarity, for example the word *"but"* in *"The food was good but the service was bad"* and
- evaluating specific tri-grams.

All of these features changed the intensity of a sentiment by a value, and those changes were reflected in the VADER model. The model was trained on Twitter's tweets, movie reviews from rotten tomatoes, technical product reviews and opinion news articles.

It is a rule-based approach that uses a bag-of-words model to extract the words in the input, and assigns a sentiment value to each word, punctuation usage and other characteristics in the input text. However, it still maintains the negation of words. For example, *"not good "* will have a negative sentiment score. The total of the sentiment values is normalised between -1

(most negative) and +1 (most positive) using an equation and is the assigned sentiment score to the input text.

We can see that VADER is a lexical-based approach that relied on WotC (Wisdom of the crowd) to come up with a rule-based model to assign sentiment to a gien input text. It has been modified upon and used in other research papers. [2]

4.3.2 TextBlob

TextBlob is a function of the textblob python library (there is a difference in capitalisation). It is a library for doing linguistic tasks such as noun phrase extraction, sentiment analysis, part of speech tagging etc. It is heavily based on the NLTK library and the pattern library. Upon going through their code in an attempt to find out how the sentiment analysis is calculated, we find that the sentiment analysis is done using the pattern library's sentiment analysis without any modification.

Pattern [14; 15] is an open-source python library that has tools for various natural language processing tasks including sentiment analysis. It also has tools for other tasks such as machine learning and data mining. It was developed and is maintained by the CLiPs Computational Linguistics group at Universiteit Awerpen [24]. Pattern uses a dictionary of adjectives having the following information: the Cornetto Id and the Cornetto synset id which is from the Cornetto lexical database for Dutch, a WordNet Id, the part of speech tag from the Penn Treebank, sense which is a definition of the word, a polarity from -1 to 1 with -1 being the most negative and +1 being the most positive, a subjectivity score from 0 to 1 where 0 is unbiased and 1 is biased and a confidence score based on whether the sentiment and subjectivity value was inferred (0.7) or was hand-tagged by an expert (1.0). This dictionary is used in its algorithm and outputs a tuple with the input text's sentiment score from -1 to +1 and a subjectivity score from 0 to 1.

4.3.3 SentiWordNet

SentiWordNet can be accessed through NLTK interface. SentiWordNet is a resource for opinion mining created by Esuli and Sebastiani [19]. It is built upon WordNet, and assigns every sunset in WordNet a sentiment score: Positive, Negative or Objective.

This approach is not like the other approaches mentioned ealier since SentiWordNet is not a pre-trained model, rather it acts like a dictionary. We use an algorithm similar to Taj, Meghji, and Shaikh [67]; Kalyanaraman, Kazi, Tondulkar, and Oswal [34]. Taj et al. [67] use an inbuilt sentiment dictionary in the programming language they are working in and all of the text preprocessing is done by the same programming language. Kalyanaraman et al. [34] manually come up with a dictionary with the sentiment scores of stock jargon. In this experiment, we will be using the SentiWordNet as a sentiment dictionary. Apart from the sentiment dictionary, the overall steps remain the same (Figure 4.1).

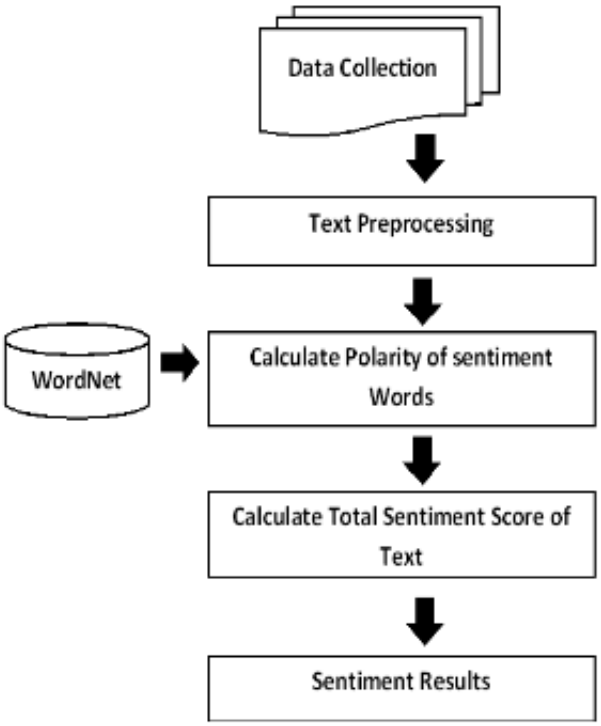

Figure 4.1: Flowchart showing the steps of this approach, from Taj et al. [67]

First we have to do the preprocessing of the text as described below:

- Removing dashes in between words. For example, *"hand-tagged"* be-

comes "hand tagged".

- Removing all non-alphabetic characters (except for spaces) so that numbers and other punctuation do not intrude with our words.
- POS tagging the text since many of the operations require the part of speech of the word it is applying a function on.
- Removing all proper nouns since they do not contribute to the sentiment of the overall text
- Making the whole text lower-case, capitalisation in news articles are basic grammar which do not contribute to sentiment, and SentiWordNet does not have capitalisation
- Removing stop words that do not have a contribution to the sentiment of the text.
- Finding the lemma of every word using NLTK's WordNetLemmatizer
- Setting each word in the correct format to use the SentiWordNet interface.

SentiWordNet requires three pieces of information for every word, the word, the part of speech it is being used in and the variation number of the usage. For example, *"chair"* can be used both as a noun or a verb and it is important to specify which it is being used as. Additionally, words can have different usages which can have different sentiment scores. However, it is difficult to interpret which usage the word is being used in, so as a default, the first and most common version is selected.

After all the text preprocessing is done, the sentiment score is found for the article by adding all the positive sentiments, adding all the negative sentiments and subtracting the negative score from the positive score. If the value of the final sentiment score is above 0, it is classified as positive, else it is classified as negative.

4.3.4 Flair

Flair [1] is a framework of natural language processing that is known for its implementations of word embeddings. Flair has many different types of word

embeddings such as GloVe, FastText and a unique Flair embedding too. It also has functionality to stack embeddings together to create an embedding that has the values of more than one embedding type. It also has a sentiment analysis pre-trained model. Upon going into their source code as well as reading the release updates, the pre-trained model is a distilBERT (section 4.3.5) model (trained by using the HuggingFace's Transformer Library) and is trained on amazon product reviews and movie reviews.

4.3.5 Transformer

HuggingFace's Transformer [74] can be used in python that can train state-of-the-art NLP models. It uses a transformer architecture [70]. It has implementations of ALBERT, BART, BERT, RoBERTa, etc and can be used with ease to train models over any dataset. It also has a variety of pre-trained models in different tasks and in different languages. There are pre-trained models in sentiment analysis as well, and we use two such pre-trained models.

BERT

BERT is not a model we use directly in our experiments, but it is the basis for the two models we do use in the Transformer architecture. Both distilBERT and roBERTa are based on BERT [44; 64].

BERT: Bidirectional Encoder Representations from Transformers [18] is a language representation model created by Google that can be used for various natural language processing tasks such as question answering and Natural Language Inference. BERT's model architecture is a multi-layer bidirectional Transformer encoder. The input of the model can be a single sentence or a pair of two sentences which is known as a single token sequence. The 'sentence' is not necessarily a linguistic sentence, it is simply a span of continuous words. This is so that the model can handle a variety of tasks. The model is trained over two unsupervised tasks: Masked Language Modeling (Masked LM) and Next Sentence Prediction (NSP). The Masked LM is used to get the biredictionality of the model. It randomly masks 15% of the words in the input and then the model has to predict the masked words after reading the whole masked input. NSP is important for Natural Language Inference and Question Answering. BERT is pre-trained on BookCorpus (a dataset of 11,038 unpublished books) which has 800 million words and English Wikipedia which has 2,500 million words not including tables, lists and

headers.

distilBERT

Hugging Face's default sentiment analysis model is the distilBERT-base-uncased-finetuned-sst-2-english. It uses the distilBERT algorithm [64] which is fine-tuned on Stanford's Sentiment Treebank. It classified a given text into Positive or Negative.

DistilBERT is a distilled version of BERT [18] which was introduced by Sanh et al. [64] by Hugging Face. It is a smaller and faster version of BERT which uses BERT as the teacher to self-supervise. It was trained to have similar probabilities as BERT, it also has the task of Masked Language Modelling (Masked LM) which randomly masks 15% of the words in the sequence, and the model predicts the words in the mask. The model is also trained to generate hidden states similar to the BERT model.

The changes it makes to the BERT architecture to achieve the faster model is that it reduces the number of layers in the transformer by a factor of 2. It also makes a number of other smaller changes to reduce the computational complexity and achieve 97% of BERT's performance. It is 40% smaller and 60% faster than BERT.

RoBERTa

Hugging Face has a series of pre-trained models one can use for their purposes, and one of the sentiment analysis models we used is cardiffnlp/twitter-roBERTa-sentiment. This model was created by Barbieri et al. [4] on a dataset of around 58 million tweets categorized into Positive, Negative and Neutral.

TweetEval [4] consists of seven tasks to be done over twitter data. This task is defined as multi-class tweet classification. These tasks are Emotion Recognition, Emoji Prediction, Irony Detection, Hate Speech Detection, Offensive Language Identification, Sentiment Analysis and Stance Detection. All of these tasks are done using various models such as RoBERTa, FastText, LSTM and SVM. They found that their retrained RoBERTa model performed the best out of all the models with 72.6% accuracy for the sentiment analysis task. This performs better than FastText's 62.9%, LSTM's 58.3% and SVM's 62.9%. They published this model to be used with HuggingFace's Transformer Library.

The algorithm used is RoBERTa : Robustly optimized BERT approach [44]. It is based on BERT (explained in section 4.3.5). The model is run more times in effort to increase its efficiency. Liu et al. [44] experiment with the various variables in the BERT architecture, and fine-tune some to optimize the model. They also do not inject short sequences unlike Devlin et al. [18]. "RoBERTa is trained with dynamic masking, full sentences without NSP loss, large mini-batches and larger byte-level Byte-Pair Encoding." [44]. These improvements are combined into RoBERTa.

4.3.6 Stanza

Stanza [59] is an open-source python library that was created by the Stanford CoreNLP Group and has a plethora of natural language processing tools and supports a wide variety of human languages. It currently supports 66 languages, has raw text processing, is fully neural and has pre-trained models like the one we use in this task. It has a python interface to the Stanford CoreNLP Java Package [47] and inherits functionality from it.

Stanza has a sentiment analysis pretrained model that is based off of Kim [37]. The text classification is done using a Convolutional Neural Network (CNN).

For English, the sentiment analyser is trained on Stanford Sentiment Treebank (SST), MELD's (Multimodal EmotionLines Dataset) text from the TV show "Friends", SLSD (Sentiment Labelled Sentences Dataset) which has reviews from imdb, amazon and yelp, Arguana data and Airline Review Twitter data.

The model gives an output of 0, 1 or 2 where 0 corresponds to negative, 1 corresponds to neutral and 2 corresponds to positive.

4.4 Results

Pre-Trained Model	Document-Level	Sentence-Level	Neutral Handling
VADER	0.593	0.62	NO
TextBlob	0.5	0.467	NO
SentiWordNet	0.5	0.533	NO
Flair	0.507	0.607	NO
distilBERT	NA	0.613	NO
RoBERTa	NA	0.843	YES (66%)
Stanza	NA	0.617	YES (28.7%)

Table 4.1: Results showing the effect of using the first sentence vs the full article and the differences between the various pre-trained models

4.5 Error Analysis

4.5.1 Sentence-Level vs Document-Level

Our hypothesis is proven correct with the fact that in most cases (except TextBlob), the sentence-level sentiment analysis is more accurate than the document-level sentiment analysis. This, as seen from the table 4.1, does not include the outputs of transformer distilBERT, transformer RoBERTa and Stanza. The transformer library does not allow an input of the size of the whole document, and the Stanza library applies sentiment analysis at sentence-level, so even after inputting the whole document, we get an array of sentiments for each sentence in the document. We did not want to apply any kind of function to the array of sentiments such as averaging to find the sentiment of the document since it is not pre-decided by the makers of the model.

A possible reason why using the first sentence's sentiment was more accurate in general than using the whole document can be because the first sentence gives an overview of the article while the document will have opposing viewpoints included as well. The opposing viewpoints may confuse the model, and add the opposite sentiment into the overall score.

4.5.2 Pre-trained models

One issue that comes up while doing the evaluations of the models is that some models give Positive and Negative as outputs while others have Positive, Negative and Neutral as outputs. VADER, TextBlob and SentiWordNet approaches all give a value which can be converted into either type of model. The problem occurs when annotating the dataset and deciding whether to classify articles with a Neutral Tag at all.

If we choose to have a Neutral Tag, models that do not output Neutral, Transformer distilBERT and Flair, will surely show a lower accuracy than actual. This is because the model itself is incapable of realising that an article can be Neutral.

If we choose not to have a Neutral tag, the task becomes simpler. VADER, TextBlob and SentiWordNet are classified such that a value above 0 is classified as positive otherwise it is Negative. Then Flair and Transformer distilBERT becomes easy to compute the accuracy and working of the model. However with Transformer RoBERTa and Stanza, computing accuracy becomes more difficult. We decided to not include the Neutral tags in the accuracy computation altogether. There are some errors in this approach too.

As we can see from the results, Transformer RoBERTa seems to have the best accuracy, however, this is after removing the Neutral outputs. 66% articles were removed from this accuracy score since RoBERTa classified them as Neutral, and hence the number of articles actually tested are far less. Additionally, it is not ideal that these many articles are classified as neutral since at least the majority should have a positive/negative classification. This could be due to the fact that Transformer RoBERTa was trained on twitter data with a high polarity in writing while news articles have a formal style of writing with less polarity. Stanza also had a Neutral Tag as output, but only 28.7% were classified as Neutral, and so the majority were classified with Positive or Negative.

Other models that did not do so well are TextBlob and SentiWordNet. This could be because TextBlob and SentiWordNet both have dictionaries that have polarity scores of each word. News articles do not typically have highly polarising words. However, VADER, which has a dictionary of polarity, it also has rules for assigning sentiment to different word usages, punctuation usage etc. This could be why VADER performs better than TextBlob and SentiWordNet.

4.6 Conclusion

From our results, Transformer's RoBERTa seems like the most accurate model for sentiment analysis in news articles. However, this is not the case since the model was not able to classify *most* of the articles as either positive or negative, it took the easy route of classifying the majority of articles as neutral. VADER, Transformer's distilBERT and Stanford's Stanza all perform with similar accuracies. If the neutral tag should be included in the data, it makes the most sense to either use VADER or Stanza, since VADER outputs a sentiment value from 0 to 1, while Stanza has an output tag of neutral.

4.7 Future Work

- To find the best pre-trained model for polarised texts such as reviews, Twitter data, etc. one could test those data on all of these pre-trained models. Most of the models are trained solely on polarised data, so the results could be different from the ones presented in this task.
- Instead of relying solely on the whole document or the first sentence, a new hybrid approach could be created where to find the sentiment of a news article where extra emphasis or weightage is given to the first sentence of a news article. This could work with a good accuracy since we have proven that there is a correlation between the sentiment of the first sentence and the sentiment of the whole document in the case of news articles.
- A sentence-level vs document-level sentiment analysis could be done for other types of long texts such as blogs. Blogs too have a "catchy" first sentence that correlates to the content of the whole document.
- With more annotated data, more models can be trained, so we need not be limited to pre-trained sentiment models, rather other pre-trained models (that are not trained for sentiment) such as BERT, GraphBERT, FastText, etc can be used, or even other models can be trained from scratch.

Chapter 5

Author Identification using Traditional Machine Learning Models

5.1 Introduction

Authorship analysis [36; 16] is a category of tasks that relate to finding the author of a text that falls under the stylometry field in linguistics. There are three main tasks: Author Characterisation, Text Similarity, and Author Identification/Attribution. Author Characterisation is the task of identifying a set of characteristics that are relatively common for all pieces of work written by the same person or a group of people *e.g* people of the same gender, or cultural background. [29; 16; 36] Text Similarity is the task of comparing two texts, and calculating how similar the two texts are while not necessarily identifying the author of the text. [16; 36]. However, we decided to focus on Author Identification as the main task.

Author Identification is also known as Author Attribution and is the task of extracting the author of a text from a group of suspect authors. We aim to create a computationally non-complex model that works in minimal time to find the author of a given text. This task focuses on the use of various stylometric and word-based features as well as different traditional machine learning models to create a classifier that gives the best accuracy.

5.2 Features

The features are the values derived from the text to be used by the classifier to predict the class or the author it belongs to. We use a similar categorization as Khan et al. [36].

5.2.1 Word-Based

This group of features are those features that are directly derived from the words in a text. The words may be individual (Bag Of Words), or clubbed together with other words (n-grams) or can be represented as a vector (Word2Vec, GloVe embeddings [56], Doc2Vec).

Bag Of Words

Bag of Words (BOW) [77] is a way of representing a text by having a count of each word in the text. This can be represented by a dictionary where the key is each distinct word in the text and the value for each key is the number of times it occurs in the text. To use this as a feature, one-hot encoding is used. For example, the first position of an array will be the first word, and the value in that position will be its frequency. By doing this to all of the texts in the training set, the array will be as long as the total number of distinct words in the whole training set. Then this array can be used as a feature in the machine learning models [45].

This feature is the simplest to understand, but there are a few disadvantages to using this feature. Firstly, any jargon used by an author will be heavily represented as the author's writing style. For example, if in the training set, author A writes about NASA, then the word "NASA" will have a high count for author A. So if in the testing set another author writes about NASA, that text could be misclassified as having been written by Author A because of the use of that particular word. Another issue that arises while using Bag of Words is that the word ordering is not preserved. Two sentences with the same words will have the same representation [40], Additionally, an author could have a distinctive style of writing and constantly uses a particular phrase, but that information will not be captured in the feature. This is not the optimal feature to use in Author Identification.

N-Grams

N-grams [32] is a way of representing a text by having a count of each phrase in the text. The "n" represents the number of words in the phrase, so by equating n to 1, we get the bag of words model. By having high values of n, the number of phrases becomes high, and the count of each phrase is minimal. With low values of n, similar issues to the BOW model occur. The phrases are represented similarly to the BOW model explained in the previous section.

The disadvantages of the n-grams are also similar to the BOW model. Even if a good n-value is found, the issue of jargon still arises. Also, as the window of the phrase moves along, all previous information is lost. It only keeps track of words directly after and before it, the words that may have connections earlier in the sentence or paragraph are lost. The only way to capture this information is to increase the value of n, which is not ideal. This is also not an optimal feature to use in Author Identification.

Word Embeddings

Word Embeddings, specifically Doc2Vec [40], is the word-based feature used finally in this task. Doc2Vec is a modification on Word2Vec [50] word embeddings. Word2Vec represents each word as a vector. These vectors aim to also show the word associations. Doc2Vec modifies the Word2Vec vectors to have additional functionality such as the position of words with respect to each other, and the position of words in paragraphs.

This technique combats the problem of jargon being associated with one author far better than the above two options, and it also does not lose information of words read earlier or later in the text. Hence, we use this in our final experiments.

5.2.2 Stylometric

These features are those that relate to an author's subconscious choice as compared to the particular words or phrases the author uses. This can be an author's preference towards writing long sentences, short paragraphs or using a huge variety of vocabulary. Many stylometric features are featured in Zheng et al. [78], though that work does not use the same categories for classifying the features. We use the features used by Zhou and Wang [79]; Qian, He,

and Zhang [60] (average word length, average sentence length and Hapax Legomenon Ratio) and the length of the article in words and sentences.

Article-Level

These are the stylometric features that show an author's choice in how they choose to structure or write their articles.

e.g the number of words in the total article, the total number of sentences in the article, the number of paragraphs, the total number of lines.

Paragraph-Level

These are the stylometric features that show how an author structures their paragraphs. For example, one author may prefer writing big paragraphs with long, run-on sentences, while another may prefer shorter sentences while still keeping a large paragraph, while another may just prefer shorter paragraphs.

e.g Number of characters per paragraph, number of words per paragraph, number of sentences per paragraph, average number of words per sentence.

Word Level

These are the stylometric features that show the types of words an author may prefer.

e.g. total number of distinct words, total number of short words (words with less that four characters), average length of word in characters.

Vocabulary Richness

Vocabulary Richness shows the variety of words that the author uses.

e.g Hapax Legomenon Ratio - frequency of once-occuring words, Hapax Dialegomenon Ratio- frequency of twice-occurring words, Yule's K measure, Simpson's D measure, etc.

5.2.3 Syntax

Syntax Features are those that are either functional word features such as the frequency of using the word 'by' (This particular function word feature was used in finding out the author of the disputed works in the Federalist Papers (section 2.2.1) or punctuation features such as the frequency of the

usage of dashes. Clearly, using dashes is individualistic to an author, not all authors use dashes, and some may use them more than others. Similarly, some authors who may write longer sentences may use more commas than an author who writes small sentences. According to Zheng et al. [78], there are 8 different types of punctuation features, we use four: quotation marks, commas, dashes and exclamation marks. As further work in this area, functional word features can be added to our work.

5.3 Machine Learning Models

As mentioned in our introduction, we are using computationally inexpensive machine learning models to do the classification. We divide the models into four types: Linear, Nearest-Neighbour Based, Trees and Boosting.

5.3.1 Linear Models

Linear Models are those that attempt to linearly separate the data points. In this category we have Naive Bayes, Linear Regression and Support Vector Machines. Even though at first glance Naive Bayes does not look like a linear model, it is linear in the logarithmic feature space, so we place it here.

Naive Bayes

Naive Bayes [35] is one of the simplest machine learning models. It is based on Bayes Theorem and relies on the assumption that every feature is independent from the other. This is a huge assumption to make because all of the features are extracted from the same article which makes it highly likely that the features are all correlated. This is not expected to perform well, rather it is a baseline for our results.

Baye's Theorem:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

where

- $P(A|B)$ is the probability of event A given that event B has already happened

- $P(B|A)$ is the probability of event B given that event A has already happened
- $P(A)$ is the probability of event A happening
- $P(B)$ is the probability of event B happening.

Bayes theorem is a theorem developed by Thomas Bayes which gives the probability of an event happening given that another event has already occurred. If these two events are independent, then they have no dependence so $P(e_2|e_1)$ and $P(e_2)$ become equal.

With classification problems we can represent the problem as

$$P(C_k|x_1, x_2....x_n)$$

where C_k is the class K and $x_1, x_2...x_n$ are n features.

Then using Bayes Theorem:

$$P(C_k|x_1, x_2...x_n) = \frac{P(x_1|x_2....x_n, C_k) * P(C_k)}{P(x_1)} * \frac{P(x_2|x_3....x_n, C_k)}{P(x_2)} * ...$$

We remove the denominator values since they do not affect the class C_k's probability ($x_1, x_2...x_n$ will all have the same denominator value).

So we get:

$$P(C_k|x_1....x_n) = P(C_k) * P(x_1|x_2...x_n, C_k) * P(x_2|x_3...x_n, C_k) * .....$$

Now the assumption that $x_1, x_2....x_n$ are all linearly independent comes in. This assumption makes the equation much easier to solve.

$$P(C_k|x_1....x_n) = P(C_k)P(x_1|C_k)P(x_2|C_k)....P(x_n|C_k)$$

The class with the highest value will be the predicted class of that data point.

We use this algorithm in our experiments as a baseline, ideally all other models will perform with a better accuracy than Naive Bayes.

Linear Regression

As seen by the name, Linear Regression [38; 7] is usually used for regression problems. It is the regression counterpart of Linear Classification. However, since Support Vector Machines optimise the decision boundary of Linear Classification and it is a part of our experiments, we decided to try this algorithm instead.

Linear Regression places all of the data points in a feature space and attemps to find a line that best represents the data clustering. Let the line have an equation:

$$w^T x + b = y$$

where w^T is the vector with weights $[w_1, w_2..w_n]$, x^T is the feature vector $[x_1, x_2..x_n]$ and n is the number of features.

Then a loss function is calculated by calculating the *residuals*. Residuals are the individual errors, usually they are the vertical distance between the line and the point.

Then using this loss function, the weights w are changed using the gradient descent algorithm.

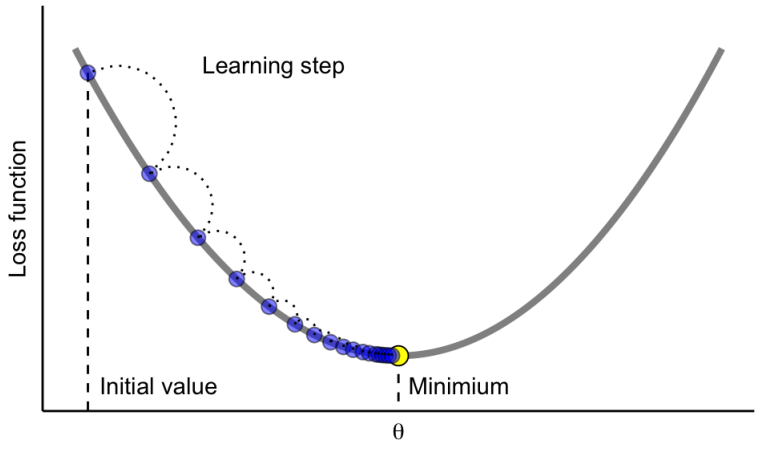

Figure 5.1: Reducing the loss function to reach a minima point by changing parameters (learning rate) [6]

If the loss function has a high value, the slope of the loss function is steep, so the learning rate is increased. If the loss function has a low value, the value

is close to the minima point and the learning rate is decreased. Until the minima point is reached, the weights keep getting changed.

If the learning rate is too high the minima point might be skipped, if the learning rate it too low, the algorithm does not move fast enough.

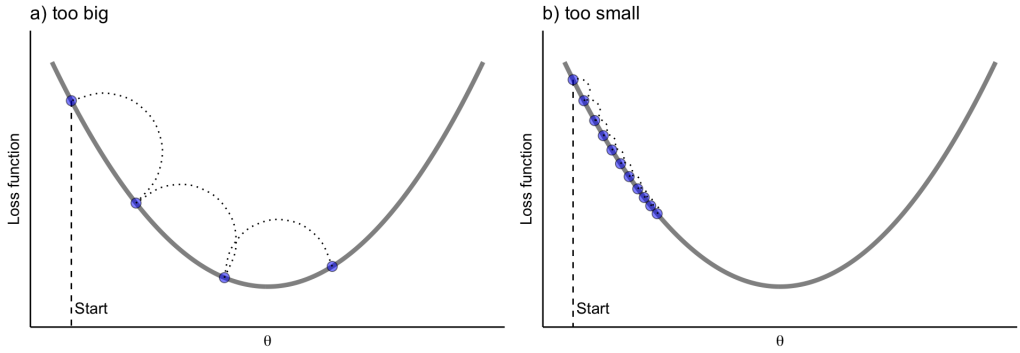

Figure 5.2: Effect of the learning rate being too large or too little. [6]

This new line with the new weight values is used to calculate the value of a new data point.

Support Vector Machines

Support Vector Machine (SVMs) [55; 20; 3] is a supervised machine learning model that can be used to separate linearly separable classes.

Caption: There are an infinite number of hyperplanes that separate two linearly separable classes.

In the diagram 5.3, we can see that there are infinite hyperplanes (lines in a 2-D space) possible to separate two linearly separable classes. Linear Classification Model would find the first hyperplane that linearly separates both classes and consider that as the classification hyperplane. From then on, the points above the hyperplane would be one class and the points below the hyperplane would be the other class. However, out of infinite such hyperplanes, there is a high probability that Linear Classification will not choose the optimal hyperplane.

The SVM finds this optimal hyperplane by choosing the hyperplane that does two things: The hyperplane must linearly separate the two classes The hyperplane must have the greatest distance between the hyperplane and each of the classes of all the other hyperplanes.

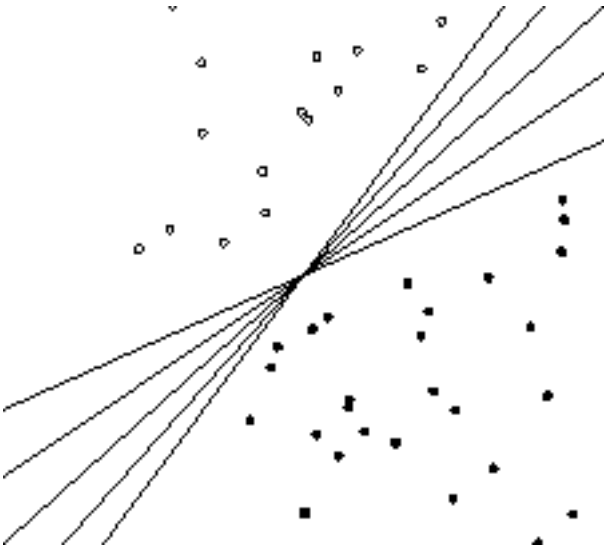

Figure 5.3: There are an infinite number of hyperplanes that can separate these points [46]

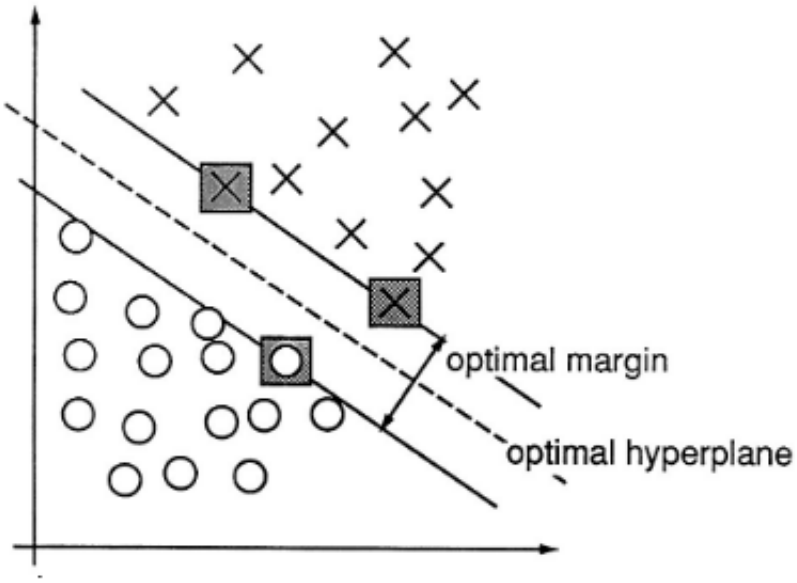

Figure 5.4: The optimal hyperplane would have maximum difference between two parallel hyperplanes [11]

These two conditions can be represented mathematically: If

$$w^T x + b = 0$$

is the equation of the hyperplane where $w^T = [w_1, w_2, w_3....w_n]$ and $x^T = [x_1, x_2, x_3...W_n]$ Then there are two parallel lines

$$L1 : w^T x + b = 1$$

$$L2 : w^T x + b = -1$$

such that any point on L1 or above L1 will be given the class 1 and any point on L2 or below L2 will be given the class -1. In other words, points on or above L1 will have a $y_i = 1$, and points on or below L2 will have $y_i = -1$ Putting these together we get:

$$y_i(w^T x_i + b) \geq 1$$

The distance between these two lines should be maximised to support the second condition of SVMs. Since the distance should be maximised, $||w||$ should be minimised.

The above calculations are for a linear separation, but by using kernels, the algorithm can be modified to separate different types of classes such as polynomial and sigmoidal. During our experiments, we found that the other kernels perform worse than the linear kernel so we use the linear kernel function for our purpose.

This algorithm is used to separate two classes, so for a multi-class problem such as author identification, we can use two different methods to classify. One way is to create a classifier between every two classes and then see which class the data point has the highest probability in. Another way is to create a one-vs-all classifier for each class and see which class has the data point in the "one" class rather than the "all" class.

SVMs have been used in a variety of machine learning tasks such as for medical diagnosis [71], shallow semantic parsing [58] and image classification.

We used the SVM for Author Identification because if the classes are linearly separable, the SVM could be the optimal classifier. Additionally, the SVM gives decent results with less data to analyse, so the classifier could give better results than a model that requires more data. SVMs have also been used by other authors in the author identification task.

5.3.2 Nearest-Neighbour Models

Nearest-Neighbour Based Models are interesting in the fact that they do not attempt to generalise how the data points are placed, rather they remember the training data points. This means that if the training data is representative of the data, the model should be able to classify the test data points.

K-Nearest Neighbour Model

The K-Nearest Neighbour (KNN) [12] Algorithm is a classification model that is based on the Nearest-Neighbour Algorithm.

The training data points are all plotted in the feature space. Then while testing one data point, it is plotted in that feature space. Then with a defined value k (chosen optimally during the training phase in this algorithm), the k closest data points are noted. Then a Majority Vote is taken to choose the predicted class. When using an even k-value, it is possible that there may be an equal number of points of both classes, so ideally, k will be odd in a two class classification.

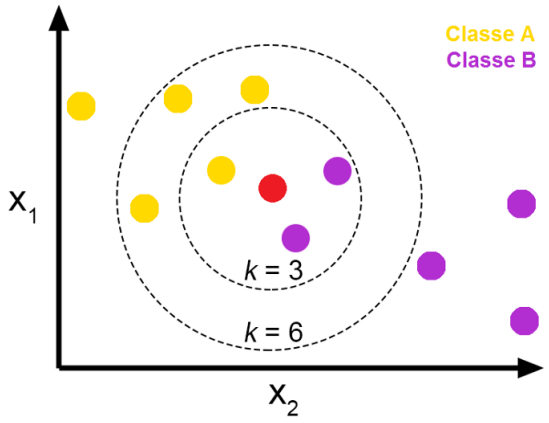

Figure 5.5: Different k-values affecting the classification of the unknown point. [31]

In the figure 5.5, the circles (in a 2-D space) represent different k values. For the first k value: k=3 there are three points within the circle. There are two points of class B and one point of Class A, so the point gets classified as Class B. However, when the value of k increases to 6, there are 4 points of Class A and 2 points of class B, so the point gets classified as Class A.

In this algorithm, the classes do not need to be linearly separable or have any discernible pattern to the class separation, but there is an assumption that data points in the same class will be close together. Additionally, the dataset needs to be balanced for a possible accurate prediction. We used this algorithm in author identification because it can be said that each authors' work will be similar in some sense, so their data points may be close together. Also, our dataset is balanced with each author having 100 works in total (training and testing).

5.3.3 Tree Models

Tree Models have one or many trees: graph-like structures with a root node that has no incoming edges, and all other nodes having one incoming edge. Tree models have no underlying assumptions about the distribution of the data or about the relationships between the features. It only has the assumption that the dataset is balanced.

Since there is no underlying assumption about the distribution of the data points, the tree models are good at representing complex relationships between the classes where the distribution of the data points is non-linear.

Decision Trees

Decision Trees [61; 63] is a classifier that has a root node with no incoming edges, and all other nodes have one incoming edge. If the decision tree is a binary tree, then each node can have 0,1,2 outgoing edges. A leaf node has 0 outgoing edges and 1 incoming edge from its parent node. Each leaf node is assigned to one class, but one class can have multiple leaf nodes.

In the figure 5.6 we can see that the root node is splitting the decision tree with a criteria of age. From the leaf nodes we can see there are two classes: Yes and No. There are two leaf nodes that contribute to the Yes class, and two that contribute to the no class. It is not necessary in every case that the same number of leaf nodes exist for every class, since the split on a parent node may not be equal along its children nodes. To find the class of a testing point, we start at the root node. At the root node, the condition is checked, and whichever child node the testing point fits in (in this example: if the testing point has age more than 30, it goes down the right subtree, else it goes down the left subtree), the testing point goes further down that tree,

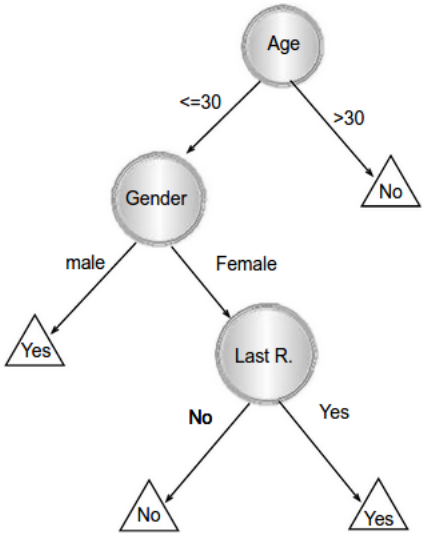

Figure 5.6: Example of a simple Decision Tree [63]

until each condition is uncovered and it reaches a leaf node. That leaf node's class is the predicted class of the testing point.

Initially, with a training set of data points (X_i, y_i) where X_i is the feature vector of point i and y_i is the class of point i, the root node needs a splitting condition that separates its two children subtrees. All splitting conditions are compared according to a metric, and the best one is chosen for the root. Then the training set is split according to the splitting condition, and the children subtrees are built with their section of the training set. This recursion happens until the training set of a subtree all belong to the same class, i.e. that subtree becomes a leaf-node, or the maximum depth of the tree is reached.

There are several improvements that have been made to this base algorithm to decrease the tree size, combat with overfitting, choose the best splitting condition, etc. which have been detailed by Rokach and Maimon [63].

Extra Trees

Extra Trees or Extremely Randomized Trees [25] is an ensemble algorithm that uses the majority vote of multiple decision trees for its final prediction. The decision trees are created parallely, and are not correlated to each other. No tree learns from the errors of another which is the case in Boosting (section 5.3.4).

The decision trees are trained over the whole training data set. However, to make sure that the trees are different (there is no meaning to a majority vote over a set of same trees), a randomized subset of the total features is given to the tree. To split the node, a splitting condition is needed. From the randomised subset, K times a random feature split is chosen with replacement. Then the best among those K splits is chosen as the split for the node. The remaining algorithm of the decision tree remains the same.

Since the best among K splits is chosen, the decision tree should be accurate, especially since the whole training dataset is used to train the tree. Additionally, because of the different feature subsets given to each tree as well as the K random splits, the trees should be different enough that the same errors should not be present in the majority of the trees.

Extra Trees algorithm works better than a sole decision tree since it reduces the overfitting and increases the accuracy. The algorithm is similar to Random Forest (explained further in section 5.3.3) but is faster since the split is somewhat randomly chosen instead of having to find the best split amongst all of the features.

Random Forest

Random Forest [9] is an ensemble algorithm that is based on the use of decision trees. Unlike Boosting (section 5.3.4), the decision trees are created parallely and do not benefit from the mistakes of the previous decision trees. Rather, they are created simultaneously and have errors that ideally are not present in all or most of the trees. Then, for a test data point, every tree predicts the class of that data point, and a majority vote is taken amongst the trees. The majority is the predicted class of the Random Forest.

For this algorithm two work accurately, there are two criteria that the trees have to satisfy.

- Each of the trees should work with a decent accuracy.

- The trees should all be uncorrelated or as less correlated to each other as possible.

The individual trees need to have a decent accuracy because each of the trees is involved in the majority vote for the final class of the test data point. If a significant number of the trees have a low accuracy, then the output of the random forest will have a low accuracy. Hence, while training the tree, the best split is chosen every time.

The trees should be uncorrelated or less correlated so that each tree's errors are not reflected in other trees. This is done by assigning a different random feature vector to each tree. When training the trees, the training data is bootstrapped original training data (the distribution of the data is kept similar), and the best split is chosen from the subset of features assigned to that tree (the random feature vector).

The Random Forest improves on the working of a sole Decision Tree since it minimises the chance of overfitting and it increases the accuracy by using many trees with different splitting conditions. According to Breiman [9], the Random Forest works as well as AdaBoost (section 5.3.4) and is faster than bagging or boosting algorithms.

Extra Trees and Random Forest have some similarities in their algorithms. Both are ensemble methods that create decision trees parallely. While creating the trees, a random subset of features are used to create the split. Both also use the majority vote for the final prediction. The differences between both are subtle:

- Random Forest Bootstraps the training dataset to train each tree. Extra Trees uses the full dataset to train each tree.
- When training each tree, the best split is chosen from the random subset of features. Extra Trees randomly selects K splits from the random subset of features, and the best amongst those K splits are chosen.
- Extra Trees have a higher variance, but lower run time (since the best split is not chosen while training each tree) than Random Forest.

5.3.4 Boosting

Some classifiers try to come up with one rule that fits the data, and then classify all following test points into that one rule. Looking at our own exam-

ples, a Support Vector Machine tries to find one line that linearly separates the data points into class A and class B. Coming up with a highly accurate rule is difficult (a SVM relies on the linear distribution of data points), so Boosting [65] tries to come up with a series of weak rules used together to correctly classify a datapoint.

First, the rough, not-so-good rules are found. This is done using a weak classifier, a classifier that works with just better than 50% accuracy or just better than a random chance and is known as the *base learning algorithm*. This base learning algorithm is called many times until the Boosting program has a series of weak classifiers that it must combine to create a strong classifier.

This method is highly effective since training each of the weak classifiers is computationally inexpensive and fast. So the Boosting algorithm itself is fast. Also, since not just one rule is being used in the algorithm, the algorithm is able to represent complex relationships, and does not need a specific layout of the data points to be able to make a good prediction.

Gradient Boosting

Gradient Boosting [22; 6] is a greedy boosting method that combines several weak models to create a strong classifier. This algorithm is a sequential ensemble algorithm. It creates one base learner, and from the error and loss function of that model, another one is trained. This happens until a stopping mechanism is reached such as the user-defined limit is reached or all the data points are correctly classified.

It is a gradient descent algorithm, it tries to find a minima point in the loss function as fast as possible, then the algorithm terminates. This may not be the most minimum point in the loss function. When the loss function has a big drop in value, the learning rate is increased to go in the same direction, if the loss function has a small drop in value, the learning rate is decreased to find the minimum point.

Gradient Boosting uses this as well, if there is a huge drop in the loss function by changing a prediction, the new model will produce a classification that will try to predict the changed prediction to reduce the error. If there is a small change in the loss function by changing a prediction, the new model does not change this prediction since the error does not decrease. Each new model tries to reduce the prediction error.

The parameters of gradient boosting are:

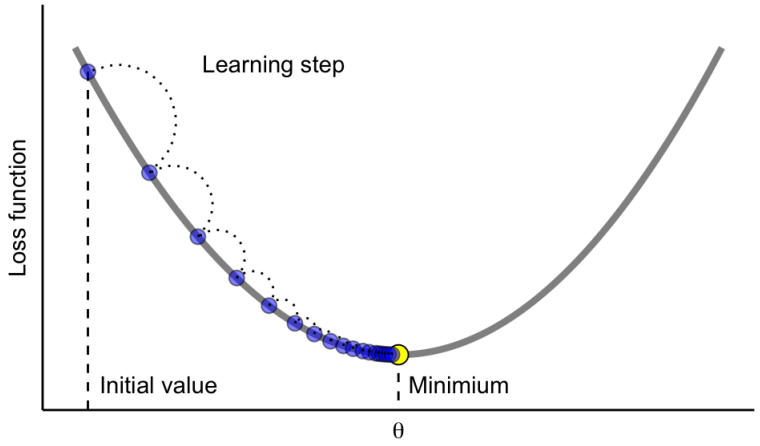

Figure 5.7: Reducing the cost function to reach a minima point by changing parameters (learning rate) [6]

- Number of Trees - having too few trees means having an over-generalised model, having too many han overfit and cost more computational time
- Learning Rate - follows the gradient descent rules

The base learners are usually decision trees with a restricted depth. Trees with higher depths risk overfitting.

Adaptive Boosting

Adaptive Boosting [21; 65] also known as AdaBoost is a boosting algorithm developed to "boost" several weak classifiers into a strong classifier. A weak classifier works with just better than 50% accuracy. This algorithm is a sequential ensemble algorithm. Sequential refers to the fact the weak classifiers are trained one after the other and aim to 'learn' from the mistakes of the previous. Ensemble refers to the fact that there are multiple classifiers used within this method.

The pseudo code in figure 5.8 shows the steps of the AdaBoost method. First, the model is initialised with some data points. An arbitrary data point (x_i, y_i) has x_i as a feature vector of the point i, and y_i refers to the class it belongs to: -1 or +1. The algorithm initialises a set of weights D_1 for each point as $\frac{1}{m}$ where m is the total number of points.

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final classifier:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

Figure 5.8: Psuedo Code for AdaBoost [65]

Then, for each iteration a weak classifier is trained based on the data points with data points with a higher weightage being given more importance. In the first iteration, all the data points have the same weightage, so all the data points are equally important.

The classifier i itself gets a weightage α_i which is dependent on how well the classifier performs. If the classifier has a higher accuracy compared to all other classifiers, its α value will be higher.

Next the weights of all the data points are updated. This means that points that are wrongly classified by the weak classifier are given higher weightage, and the points that are correctly classified are given less weightage. The weights are all normalized. The wrongly classified points are given higher weightage so that in the next iteration, the next classifier will aim to correctly classify those points.

This iterates until the user-defined iteration limit is reached, or all the data points are correctly classified.

$$H(x) = sign(\sum t = 1T\alpha_t * h_t(x))$$

where T is the number of iterations, α_i is the classifier i's weightage, and $h_i(X)$ is the ith classifier.

Mostly, the weak classifiers in AdaBoost are decision stumps: decision trees with a restricted depth. Since the algorithm assigns higher weights to misclassified samples, if a particular sample is misclassified multiple times, the weightage keeps increasing until a model is able to correctly classify it. Hence, if this point is an outlier point, the model has a possibility of overfitting.

5.4 Results

5.4.1 Features

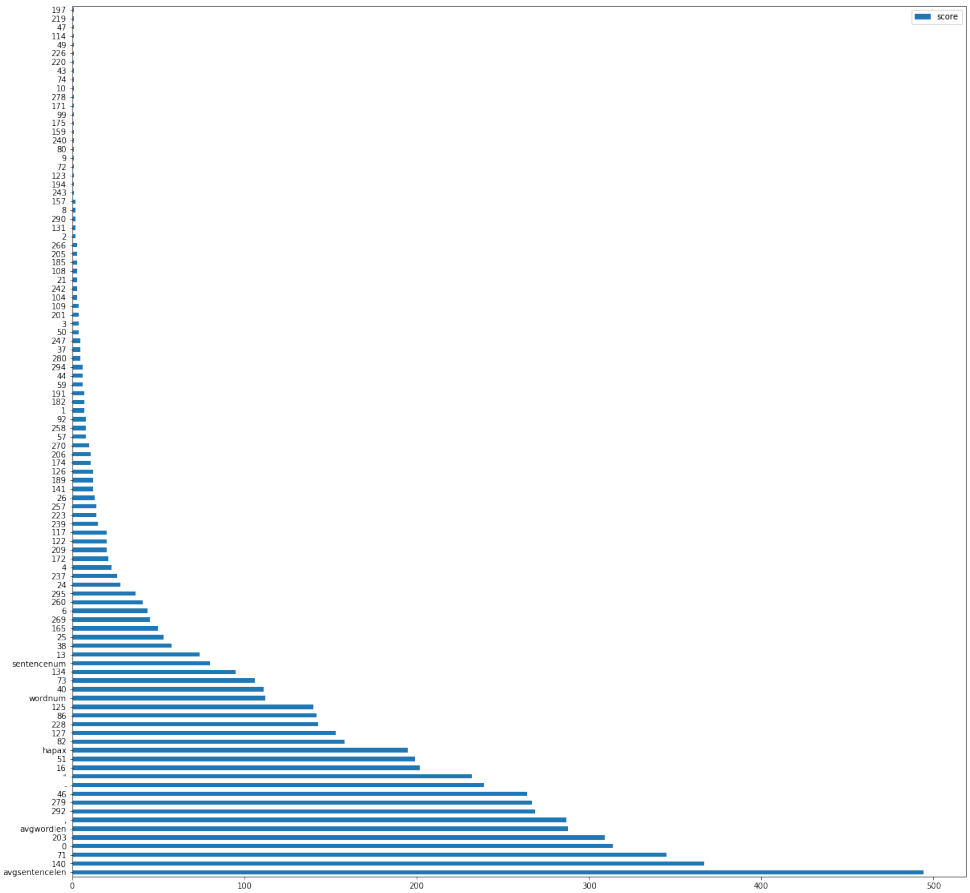

Figure 5.9: Diagram showing importance of features in XGBoost model

Figure 5.9 represents all the features and their importance in the XGBoost Model. The least important features are certain positions in the Doc2Vec vectors, while the more important features are the stylometric and syntax-based features as well as certain Doc2Vec vector positions.

By this graph, we can see that the stylometric features contributed heavily to a good classification. The average sentence length (averagesentencelen) proved to the most important feature within the whole group of features.

We can also see that the punctuation features (syntax features) are important with the usage of commas being the most important feature within this set.

5.4.2 Machine Learning Models

Model	**Training Accuracy**	**Testing Accuracy**	**F1 Score**	**Recall Score**	**Precision Score**
Naive Bayes	0.23	0.16	0.114	0.092	0.164
Linear Regression	0.42	0.36	0.330	0.333	0.344
SVM	0.59	0.48	0.463	0.464	0.486
K-Nearest Neighbour	0.54	0.26	0.246	0.247	0.267
Random Forest	1.0	0.71	0.716	0.731	0.735
Extra Trees	1.0	0.68	0.694	0.705	0.697
Decision Trees	1.0	0.57	0.577	0.578	0.588
AdaBoost	0.90	0.76	0.770	0.780	0.773
XGBoost	1.0	0.83	0.826	0.830	0.842

Table 5.1: Results of various traditional machine learning models.

Table 5.1 shows the various scores for the model using the same feature set. As predicted, Naive Bayes performs the worst with a F1 score of 0.114. We can see all the Linear Models and the K-nearest Neighbour model performed with less than 50% accuracy which makes them worse than a weak model.

The Tree models performed with higher than 50% accuracy with Random Forest being the best within the Tree algorithm group with a F1 score of 0.716.

The Boosting models performed better than all other groups of models, XGBoost performing the best with an F1 score of 0.826. AdaBoost performed second best amongst all the algorithms with a F1 score of 0.770.

5.5 Error Analysis

5.5.1 Features

In our experiments, from the set of features we chose, we did not find any that are useless in this task. All the features detailed in figure 5.9 have importance as shown in the results graph.

5.5.2 Machine Learning Models

Each group of machine learning models had an inherent assumption about how the data points would be represented in the feature space. The assumptions of one group could fail, while another could hold. Hence, the groups of models were chosen as such so that the assumptions of each group would be varied.

Linear models assume that the data points could be linearly separable. Within that group the Naive Bayes would be linearly separating the variables in the log-space, Linear Regression would separate the classes and the Support Vector Machine would try to optimize the decision boundaries. By seeing the results we can clearly say that the data points were not linearly separable since all of these models failed to classify even as a weak classifier since all had an abysmal accuracy of less than 50%.

K-Nearest Neighbour algorithm had the assumption that data points belonging to the same class would be closer together in the feature space. This we can also see is not true since the accuracy of this algorithm is also less than the accuracy of a weak classifier.

From the above two group results, we see the data was not clustered such that articles of the same author were placed near each other in the feature space. We also notice that there is no clear linear distinction between different authors. The authors' had variation in the values of their features from article to article, and hence there was no clustering occurring. Additionally, articles written by different authors may have had similar values for some features, and different values of other features. Their works were not different enough to be linearly separable, explained by the similar topics and length of the articles in the dataset.

Both Tree Models and Boosting Models do not have any assumptions about the data, but may have a problem of overfitting. We can say that there are no assumptions since Decision Trees are non-probabilistic and do a

binary split. This does not require any assumption about the data. All the other models are somewhat based on the Decision Tree, so both the groups do not rely on assumptions. However, they could have not resulted in good classifications based on the overfitting problem. Decision Trees would have had the biggest overfitting problem, and the other models such as Boosting and Random Forest, since they are improvements on Decision Trees, would have had less overfitting.

Both groups: Tree Models and Boosting perform well, but Boosting outperforms all the models. The success of both these groups can be due to the fact that while overfitting occurred (training accuracy of 1.0), the testing data had similar relationships to the training data between the classes. These groups of models are good at representing complex relationships between the classes.

5.6 Conclusion

In this task, we aimed to use traditional machine learning models to do the task of author identification using three types of features viz. word-based, stylometric and syntactic. We prove that all the three of these features are useful in author identification by seeing their importance in the XGBoost model. We also see that Boosting models perform the best amongst the traditional machine learning models since both the Boosting models outperform the remaining models. XGBoost performs the best with 0.826 F1 score and 82.5% testing accuracy. This result is found due to the non-linear relationship between the classes Hence, we prove that traditional machine learning models can be used in author identification, which reduces the complexity and time to perform. It can work with an equal or better accuracy than that of complex machine learning models.

5.7 Future Work

- There are many stylometric features that have not been tested in this work. With more stylometric features, the text may be better represented. Additionally, one can find out the best set of stylometric features to use for author identification [78; 42].
- Function word features can be included to see the author's preferred

use of functional words. This has been used to find the author of the disputed Fedaralist Papers [53; 8; 23].

- Character n-grams can be beneficial [52; 27].
- Different word embeddings could be tested for author identification.

Chapter 6

Conclusion and Future Work

Sentiment analysis is a powerful analyser and an important task in NLP. To be able to use sentiment in other tasks, a pre-trained readily available coding function is ideal. We observe that all of the pre-trained models (VADER, TextBlob, SentiWordNet, Flair, Transformer's distilBERT, Transformer's RoBERTa, and Stanza) are trained and tested on polarised text forms such as Tweets (Transformer's RoBERTa), reviews (Flair) or a combination of them. Hence, we decide to test all of the models on a non-polarised text form: News Articles. We also found that news articles typically have the first sentence depicting both the opinion of the author and the overall content of the document. Hence, the analysis was done in two parts. The first part was finding the ideal pre-trained model and we found that VADER, Transformer's distilBERT and Stanza all perform better than the other models and can be used for fast and easy sentiment analysis. The second part of our experiments was seeing if a correlation between the first sentence's sentiment and the sentiment of the whole document existed. Our hypothesis was correct: there is a strong correlation between the sentiment of the first sentence and the target sentiment of the overall document.

Using the same dataset, we use it for its intended purpose: Author Identification. We saw in our readings that many author identification programs use deep learning methods that require time, effort and a lot of data. We decided to use traditional machine learning models and combat its disadvantages in accuracy by using more features. All the features we experiment with turn out to be useful in this task. The traditional machine learning models we experimented with are categorised into Linear models, Nearest Neighbour Based models, Tree Models and Boosting Models. We found that

linear models and nearest neighbour based models do not work with a desirable accuracy for this task. Tree models and Boosting models both perform well, but Boosting models outperform all other classes. Within this category, XGBoost algorithm performs the best.

6.1 Future Work

The future work of each individual task has been discussed in section 4.7 and section 5.7. Some of the directions of future work are listed below.

- Creating a sentiment model for news articles that gives higher weightage or importance to the sentiment of the first sentence. We have already proven that the sentiment of the first sentence of a news article can show the sentiment of the full article, so giving the first sentence more importance can improve the working of a sentiment model.
- Using a larger annotated dataset to train models from scratch to incorporate a greater variety of models that can be used for sentiment analysis.
- Experimenting with more features such as function word features and other stylometric features that have not been used in our experiments. Adding more features can result in the model having higher accuracy.

Publications

- A Comparative Study of Pre-trained Sentiment Analysis Models and Sentence-Level and Document-Level Sentiment (in review at ICON 2022)
- Author Identification using Traditional Machine Learning Models

  Binnani, Ojaswi. (2022). Author Identification using Traditional Machine Learning Models. 13-20. 10.5121/csit.2022.121402.

Bibliography

[1] Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. Flair: An easy-to-use framework for state-of-the-art nlp. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59, 2019.

[2] Al Amin, Imran Hossain, Aysha Akther, and Kazi Masudul Alam. Bengali vader: A sentiment analysis approach using modified vader. In *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pages 1–6, 2019. doi: 10.1109/ECACE.2019.8679144.

[3] Mariette Awad and Rahul Khanna. *Support Vector Machines for Classification*, pages 39–66. 01 2015. ISBN 978-1-4302-5989-3. doi: 10.1007/978-1-4302-5990-9_3.

[4] Francesco Barbieri, Jose Camacho-Collados, Luis Espinosa-Anke, and Leonardo Neves. TweetEval:Unified Benchmark and Comparative Evaluation for Tweet Classification. In *Proceedings of Findings of EMNLP*, 2020.

[5] Nina Bauwelinck and Els Lefever. Measuring the impach of sentiment for hate speech detection on twitter. In Dennis Folds, Els Lefever, Ralucca Gera, and Veronique Hoste, editors, *Proceedings of HUSO 2019, The fifth international conference on human and social analytics*, pages 17–22. IARIA, International Academy, Research, and Industry Association, 2019. ISBN 9781612087252.

[6] Bradly Boehmke and Brandon Greenwall. *Chapter 12 Gradient Boosting*. CRC Press, 2020.

[7] Bradly Boehmke and Brandon Greenwall. *Chapter 4 Linear Regression*. CRC Press, 2020.

[8] Robert A. Bosch and Jason A. Smith. Separating hyperplanes and the authorship of the disputed federalist papers. *The American Mathematical Monthly*, 105(7):601–608, 1998. doi: 10.1080/00029890.1998.12004933. URL https://doi.org/10.1080/00029890.1998.12004933.

[9] L Breiman. Random forests. *Machine Learning*, 45:5–32, 10 2001. doi: 10.1023/A:1010950718922.

[10] Marie Campbell. *Strange World of the Brontës*. Sigma Leisure, 2001. ISBN 9781850587583. URL https://books.google.com.sg/books?id=g3GLSjxYUBQC.

[11] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Chem. Biol. Drug Des.*, 297:273–297, 01 2009.

[12] Padraig Cunningham and Sarah Delany. k-nearest neighbour classifiers. *Mult Classif Syst*, 04 2007.

[13] Maddalena De Leo. Who wrote wuthering heights?: The authorship issue. *The Sisters' Room*, Jun 2017. URL https://www.thesistersroom.com/2017/04/03/who-wrote-wuthering-heights-the-authorship-issue-article-by-maddalena-de-leo/?lang=en.

[14] Tom De Smedt and Walter Daelemans. Pattern for python. *J. Mach. Learn. Res.*, 13(null):2063–2067, June 2012. ISSN 1532-4435.

[15] Tom De Smedt, Lucas Nijs, and Walter Daelemans. Creative web services with pattern. 06 2013.

[16] Olivier De Vel. Mining e-mail authorship. 08 2000.

[17] Ali Derakhshan and Hamid Beigy. Sentiment analysis on stock social media for stock price movement prediction. *Engineering Applications of Artificial Intelligence*, 85:569–578, 2019. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2019.07.002. URL https://www.sciencedirect.com/science/article/pii/S0952197619301666.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://www.aclweb.org/anthology/N19-1423`.

[19] Andrea Esuli and Fabrizio Sebastiani. Sentiwordnet: A publicly available lexical resource for opinion mining. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation*, Genoa, Italy, may 2006. European Language Resources Association (ELRA).

[20] Theodoros Evgeniou and Massimiliano Pontil. Support vector machines: Theory and applications. volume 2049, pages 249–257, 01 2001. doi: 10.1007/3-540-44673-7_12.

[21] Yova Freund and Robert E. Schapire. A Decision Theoretic Generalization of On-Line Learning and an Application to Boosting. In Paul M. B. Vitányi, editor, *Second European Conference on Computational Learning Theory (EuroCOLT-95)*, pages 23–37, 1995. URL `citeseer.nj.nec.com/freund95decisiontheoretic.html`.

[22] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001. doi: 10.1214/aos/1013203451. URL `https://doi.org/10.1214/aos/1013203451`.

[23] Glenn Fung. The disputed federalist papers: Svm feature selection via concave minimization. pages 42–46, 01 2003. doi: 10.1145/948542.948551.

[24] Lorenzo Gatti and Judith van Stegeren. Improving dutch sentiment analysis in pattern. *Computational linguistics in the Netherlands journal*, 10:73–89, December 2020. ISSN 2211-4009.

[25] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63:3–42, 04 2006. doi: 10.1007/s10994-006-6226-1.

[26] Francis H. Grundy. *Pictures of the Past: Memories of Men I Have Met and Places I Have Seen*. Griffith and Farran, 1879. URL https://books.google.com.sg/books?id=q0oBAAAAQAAJ.

[27] John Houvardas and Efstathios Stamatatos. N-gram feature selection for authorship identification. volume 4183, pages 77–86, 09 2006. ISBN 978-3-540-40930-4. doi: 10.1007/11861461_10.

[28] C.J. Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. 01 2015.

[29] Farkhund Iqbal, Rachid Hadjidj, Benjamin C.M. Fung, and Mourad Debbabi. A novel approach of mining write-prints for authorship attribution in e-mail forensics. *Digital Investigation*, 5:S42–S51, 2008. ISSN 1742-2876. doi: https://doi.org/10.1016/j.diin.2008.05.001. URL https://www.sciencedirect.com/science/article/pii/S1742287608000315. The Proceedings of the Eighth Annual DFRWS Conference.

[30] David Johnston. On the suspect's trail: The investigation;long and twisting trail led to unabom suspect's arrest, Apr 1996. URL https://www.nytimes.com/1996/04/05/us/suspect-s-trail-investigation-long-twisting-trail-led-unabom-suspect-s-arrest.html.

[31] Italo José. Knn (k-nearest neighbors) 1, Jun 2019. URL https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d.

[32] D. Jurafsky and J.H. Martin. *Speech and Language Processing*. Always learning. Pearson, 2014. ISBN 9781292025438. URL https://books.google.com.sg/books?id=km-kngEACAAJ.

[33] Monika Kabir, Mir Md. Jahangir Kabir, Shuxiang Xu, and Bodrunnessa Badhon. An empirical research on sentiment analysis using machine learning approaches. *International Journal of Computers and Applications*, 0(0):1–9, 2019. doi: 10.1080/1206212X.2019.1643584. URL https://doi.org/10.1080/1206212X.2019.1643584.

[34] Vaanchitha Kalyanaraman, Sarah Kazi, Rohan Tondulkar, and Sangeeta Oswal. Sentiment analysis on news articles for stocks. pages 10–15, 09 2014. doi: 10.1109/AMS.2014.14.

[35] Pouria Kaviani and Sunita Dhotre. Short survey on naive bayes algorithm. *International Journal of Advance Research in Computer Science and Management*, 04, 11 2017.

[36] Sobiya Khan, Smita Nirkhi, and Rajiv Dharaskar. Author identification for e-mail forensic. *Proceedings of National Conference On Recent Trends In Computing NCRTC*, 01 2012.

[37] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014. URL `http://arxiv.org/abs/1408.5882`.

[38] Khushbu Kumari and Suniti Yadav. Linear regression analysis study. *Journal of the Practice of Cardiovascular Sciences*, 4:33, 01 2018. doi: 10.4103/jpcs.jpcs_8_18.

[39] Alice Law. *Patrick Branwell Brontë*. A.M. Philpot, Limited, 1924. URL `https://books.google.com.sg/books?id=5WUqAAAAMAAJ`.

[40] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. *31st International Conference on Machine Learning, ICML 2014*, 4, 05 2014.

[41] David Lewis, Yiming Yang, Tony Russell-Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 04 2004.

[42] Jiexun Li, Rong Zheng, and Hsiu-chin Chen. From fingerprint to writeprint. *Commun. ACM*, 49:76–82, 04 2006. doi: 10.1145/1121949.1121951.

[43] Bing Liu. *Opinion Mining and Sentiment Anlaysis*. Springer, 2 edition, 2011.

[44] 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. *CoRR*, abs/1907.11692, 2019. URL `http://arxiv.org/abs/1907.11692`.

[45] Shuming Ma, Xu Sun, Yizhong Wang, and Junyang Lin. Bag-of-words as target for neural machine translation. pages 332–338, 01 2018. doi: 10.18653/v1/P18-2053.

[46] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Vector Space Classification*. Cambridge University Press, 2008. URL https://nlp.stanford.edu/IR-book/.

[47] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014. URL http://www.aclweb.org/anthology/P/P14/P14-5010.

[48] Rachel McCarthy and James O'Sullivan. Who wrote Wuthering Heights? *Digital Scholarship in the Humanities*, 06 2020. ISSN 2055-7671. doi: 10.1093/llc/fqaa031. URL https://doi.org/10.1093/llc/fqaa031. fqaa031.

[49] Rachel McCarthy and James O'Sullivan. Who wrote wuthering heights?, Jul 2020. URL https://www.irishtimes.com/culture/books/who-wrote-wuthering-heights-1.4305162.

[50] Tomas Mikolov, G.s Corrado, Kai Chen, and Jeffrey Dean. Efficient estimation of word representations in vector space. pages 1–12, 01 2013.

[51] L. Miller. *The Bronte Myth*. Knopf Doubleday Publishing Group, 2007. ISBN 9780307428202. URL https://books.google.com.sg/books?id=YcjjCl4ktEsC.

[52] Ahmed M. Mohsen, Nagwa M. El-Makky, and Nagia Ghanem. Author identification using deep learning. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 898–903, Los Alamitos, CA, USA, dec 2016. IEEE Computer Society. doi: 10.1109/ICMLA.2016.0161. URL https://doi.ieeecomputersociety.org/10.1109/ICMLA.2016.0161.

[53] Frederick Mosteller and David L. Wallace. *Applied Bayesian and Classical Inference: The Case of The Federalist Papers*. Springer Series in Statistics. Springer New York, 2012. ISBN 9781461252566. URL https://books.google.com.sg/books?id=LJXaBwAAQBAJ.

[54] Dinh Nguyen, Khuong Vo, Dang Pham, Mao Nguyen, and Tho Quan. A deep architecture for sentiment analysis of news articles. pages 129–140, 06 2017. ISBN 978-3-319-61910-1. doi: 10.1007/978-3-319-61911-8_12.

[55] Edgar Osuna, Robert Freund, and Federico Girosi. Support vector machines: Training and applications. *Tech Rep A.I. Memo No. 1602*, 02 1970.

[56] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

[57] Soujanya Poria, Devamanyu Hazarika, Navonil Majumder, and Rada Mihalcea. Beneath the tip of the iceberg: Current challenges and new directions in sentiment analysis research. *IEEE Transactions on Affective Computing*, pages 1–1, 2020. doi: 10.1109/TAFFC.2020.3038167.

[58] Sameer S. Pradhan, Wayne H. Ward, Kadri Hacioglu, James H. Martin, and Dan Jurafsky. Shallow semantic parsing using support vector machines. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 233–240, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N04-1030.

[59] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A python natural language processing toolkit for many human languages. *CoRR*, abs/2003.07082, 2020. URL https://arxiv.org/abs/2003.07082.

[60] Chen Qian, Ting He, and Rao Zhang. Deep learning based authorship identification. 2017.

[61] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986. ISSN 0885-6125. doi: 10.1023/A:1022643204877. URL https://doi.org/10.1023/A:1022643204877.

[62] Leigh Remizowski. Unabomber lists life sentences as achievement, May 2012. URL https://edition.cnn.com/2012/05/24/us/massachusetts-harvard-unabomber/index.html.

[63] Lior Rokach and Oded Maimon. *Decision Trees*, volume 6, pages 165–192. 01 2005. doi: 10.1007/0-387-25465-X_9.

[64] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

[65] Robert Schapire. The boosting approach to machine learning: An overview. *Nonlinear Estimation Classification*, 171, 01 2003. doi: 10.1007/978-0-387-21579-2_9.

[66] Dev Shah, Haruna Isah, and Farhana Zulkernine. Predicting the effects of news sentiments on the stock market. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4705–4708, 2018. doi: 10.1109/BigData.2018.8621884.

[67] Soonh Taj, Areej Meghji, and Baby Shaikh. Sentiment analysis of news articles: A lexicon based approach. 02 2019. doi: 10.1109/ICOMET.2019.8673428.

[68] Micheal Taylor. New details of stakeout in montana, May 1998. URL `https://www.sfgate.com/news/article/New-Details-Of-Stakeout-In-Montana-3006937.php`.

[69] Terry D Turchie. Affidavit of assistant special agent in charge. URL `https://web.archive.org/web/20081218190755/http://www.courttv.com/archive/casefiles/unabomber/documents/affidavit.html`.

[70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

[71] Konstantinos Veropoulos, N. Cristianini, and C. Campbell. The application of support vector machines to medical decision support: A case study. *Adv Course Artif Intell*, 06 1999.

[72] Mattia Vicari and Mauro Gaspari. Analysis of news sentiments using natural language processing and deep learning. *AI SOCIETY*, pages 1–7, 11 2020. doi: 10.1007/s00146-020-01111-x.

[73] Henry Weinstein. Retrial rejected for unabomber, Feb 2001. URL https://www.latimes.com/archives/la-xpm-2001-feb-13-mn-24748-story.html.

[74] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019. URL http://arxiv.org/abs/1910.03771.

[75] Frank Z Xing, Erik Cambria, and Roy E Welsch. Natural language based financial forecasting: a survey. *Artificial Intelligence Review*, 50 (1):49–73, 2018.

[76] Jun-Ming Xu, Kwang-Sung Jun, Xiaojin Zhu, and Amy Bellmore. Learning from bullying traces in social media. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 656–666, Montréal, Canada, June 2012. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N12-1084.

[77] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: A statistical framework. *International Journal of Machine Learning and Cybernetics*, 1:43–52, 12 2010. doi: 10.1007/s13042-010-0001-0.

[78] Rong Zheng, Jiexun Li, Hsiu-chin Chen, and Zan Huang. A framework for authorship identification of online messages: Writing-style features and classification techniques. *JASIST*, 57:378–393, 02 2006. doi: 10.1002/asi.20316.

[79] Liuyu Zhou and Huafei Wang. News authorship identification with deep learning. 2016.