BENCHMARKING AUTHORSHIP ATTRIBUTION TECHNIQUES USING OVER A THOUSAND BOOKS BY FIFTY VICTORIAN ERA NOVELISTS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Abdulmecit Gungor

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2018

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Murat Dundar, Chair

Department of Computer & Information Science

Dr. George Mohler

Department of Computer & Information Science

Dr. Mihran Tuceryan

Department of Computer & Information Science

Approved by:

Dr. Shiaofen Fang

Head of the Graduate Program

Dedicated to my inspiring parents, colleagues, friends and teachers,
for being the role models, cheer-leading squad and sounding boards
I have needed in my life.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my parents, Cengiz and Fatma Gungor for providing me with the support to pursue my degree in Computer Science. Without their support I may not have found myself at Purdue University, nor had the courage to engage in this task and see it through. I'm very grateful to have such a supportive family.

Importantly, I would like to thank my thesis advisor, Dr. Murat Dundar for the guidance, advice, and hours of struggling through this challenging process. Without his assistance, I wouldn't have found myself at Purdue University nor focus on the topic of Natural Language Processing. In my thesis, he has helped me to set the aim of our work as change and our methodology to achieve our aim as reflecting on experience and goal setting for future practice. Thank you Dr. Dundar, for making me feel that working with you is a fruitful, exploratory, and joyful experience.

Lastly, I would like to thank my teammates and lab-mates whom I learned a lot while working with them: Halid Ziya Yerebakan, Sarkhan Badirli, Yicheng Cheng, Ziyin Wang, Sarun Gulyanon, Huiwen Cheng, Sepehr Farhand, Asimenia Dimokranitou. Halid, Sarkhan and I have teamed up to participate in Hack-Ohio challenge. We have created a navigation system in which parking problem have been easily solved. Our project has won the Best-Hack reward among more than seven hundred participants. On another occasion, Sarun, Sarkhan and I have finished second in Code4life University Challenge organized by Roche. We have created a data driven and visually stunning chatbot. Thank you team, for your hard work and making our time at Purdue worthwhile.

PREFACE

This basis for this research originally stemmed from our idea exchanges with Dr. Dundar. Our passion has been developing better feature representation of sentences and paragraphs that can beat the performance of traditional methods. As the world moves further into the digital age, generating vast amounts of text data, there will be a greater need to create new methodologies using predefined common norms that can break down the lack of efficiency in different application domains. Hence, the aim of this work is to create a research community around it by providing them with the benchmark methodologies and a new dataset.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Gungor, Abdulmecit M.S., Purdue University, May 2018. Benchmarking Authorship Attribution Techniques Using Over a Thousand Books by Fifty Victorian Era Novelists. Major Professor: Murat Dundar.

Authorship attribution (AA) is the process of identifying the author of a given text and from the machine learning perspective, it can be seen as a classification problem. In the literature, there are a lot of classification methods for which feature extraction techniques are conducted. In this thesis, we explore information retrieval techniques such as Word2Vec, paragraph2vec, and other useful feature selection and extraction techniques for a given text with different classifiers. We have performed experiments on novels that are extracted from GDELT database by using different features such as bag of words, n-grams or newly developed techniques like Word2Vec. To improve our success rate, we have combined some useful features some of which are diversity measure of text, bag of words, bigrams, specific words that are written differently between English and American authors. Support vector machine classifiers with nu-SVC type is observed to give best success rates on the stacked useful feature set.

The main purpose of this work is to lay the foundations of feature extraction techniques in AA. These are lexical, character-level, syntactic, semantic, application specific features. We also have aimed to offer a new data resource for the author attribution research community and demonstrate how it can be used to extract features as in any kind of AA problem. The dataset we have introduced consists of works of Victorian era authors and the main feature extraction techniques are shown with exemplary code snippets for audiences in different knowledge domains. Feature extraction approaches and implementation with different classifiers are employed in

simple ways such that it would also serve as a beginner step to AA. Some feature extraction techniques introduced in this work are also meant to be employed in different NLP tasks such as sentiment analysis with Word2Vec or text summarization. Using the introduced NLP tasks and feature extraction techniques one can start to implement them on our dataset. We have also introduced several methods to implement extracted features in different methodologies such as feature stack engineering with different classifiers, or using Word2Vec to create sentence level vectors.

1. INTRODUCTION

In a general point of view, the authorship attribution problems refer to all the issues that arise in the association of an author to a specific document. A common example is that when a piece of historical text is being discovered we would like to know who wrote it and when it was written. In addition to that, we may also ask other questions such as the nationality of the author, the characteristic styles of the author, genre of the text.

One of examples of authorship attribution problems could be as follows: Given three texts below from 19^{th} century authors Edgar Allan Poe, Mary Shelley, and Howard Phillips Lovecraft, we want to identify the author of each sentences.

(a) "But the Raven, sitting lonely on the placid bust, spoke only that one word, as if his soul in that one word he did outpour."

(b) "Again there is a sound as of a human voice, but hoarser; it comes from the cabin where the remains of Frankenstein still lie."

(c) "Great Cthulhu is their cousin, yet can he spy them only dimly. "

Having a closer look at the sentences would help us capture a few useful words that would pinpoint the authors of each text. As we know Edgar Allan Poe is famous with his poem "The Raven", Mary Shelley is famous with her book "The Modern Prometheus" in which she tells a story about Frankenstein, and HP Lovecraft has an imaginary cosmic creature Cthulhu in his book "The Call of Cthulhu". The task of naming the author for each text is not a challenging one in this case. It will, however, become more challenging when an author is describing a daily activity with most commonly used English words, in which case it will require more sophisticated approaches to complete the task.

1.1 Forensic Linguistic

From a broader contextual perspective authorship attribution is also a part of Forensic Linguistics Science and the history of Forensic Linguistic Science goes back to 1968 when a professor of linguistics analyzed police statements of falsely accused felon for the death of his wife and his infant daughter. The felon later was convicted of these two murders and hanged. After three years from his trial, his downstairs neighbor was found to be a serial killer who also had murdered six other women [1]. Back in those days, the authenticity of police statements was questioned due to the specific format of statements rather than the suspect's own words leaving out the important details.

One of the key advantages of Forensic Linguistics in crime solving cases is that it provides a list of suspects. A good example would be the *Unabomber case*, which is an acronym derived from the combination of University and Airlines Bomber [2]. The suspect of the investigation was sending packages using USPS system targeting the academic and technology leaders. After or before the bombings, the suspect was sending his manifesto stating that technological improvement should be removed from our life. It was FBI's longest and costliest investigation in the late 20th century [3]. The suspect was active from 1978 to 1995, sending 16 bombs one of which was to an airplane. All the evidence and writings of *Unabomber* collected in this time period were used to create his profile yet it was not enough to identify the suspect. In his last acquainted event he requested that his manifesto, "Industrial Society and Its Future", to be published in well-known newspapers or he would send another bomb to a plane at the Los Angeles Airport [3]. By the Attorney Generals' order his manifesto was granted to be published, which later led to his brother sharing some of the suspect's writings with the law enforcement officers. The writings provided were from before the suspect's active bombings time and Linguistics analysis showed that the author of the essay papers and the Unabomber's manifesto was almost the same.

Forensic linguistics has a diverse range of topics and its application varies vastly. The authenticity of emergency calls, suicide letters, ransom demands and social media statements analysis are a few examples of forensic linguistics. The question that is tackled in Forensic Linguistic is mainly about "who wrote a specific document". The concept of linguistic fingerprinting put forward by some scholars also led to new applications of AA in law enforcement.

1.2 Related Work

There are different types of authorship attribution studies in the literature such as predicting the date of authorship of historical texts or text genre detection [4], [5]. Vast majority of previous works focuses on authorship identification by taking into consideration the stylistic features of authors such as use of grammar, function words, frequent word allocations [4], [5], [6], [7], [8], [9]. Some of the well-known problems in authorship attribution are disputed Federalist Papers classification, Shakespearean Authorship Dispute, Author of New Testament, and Author of The Dark Tower.

The Federalist Papers are a collection of 85 articles and essays written by Alexander Hamilton, James Madison, and John Jay to persuade the citizens of New York to ratify the U.S. Constitution. Authorship of twelve of these papers has been in dispute. To address this problem, using linear support vector machines as classifier and relative frequencies of words as features a study identified these papers to be written by James Madison [10].

Another dispute in authorship attribution among scholars across the world is whether William Shakespeare wrote the works attributed to him or not. It was argued that Shakespeare wasn't even educated and more than 80 authors were suggested to be the author of the writings that were under the name of Shakespeare. Christopher Marlowe is considered the most likely candidate to write these works under the name of Shakespeare when he was in jail. In order to analyze the stylistic fingerprint of Shakespeare and Marlowe and non-Shakespearean authors, namely Chapman, Jonson,

Middleton, a corpus has been put together. By taking into account "stop words, Part-of Speech (POS) tags, bigram probabilities" the following two questions were analyzed:

- With these features can non-Shakespeare-authors be classified accurately?
- If these features are useful and Shakespeare is truly "Late Marlowe", then classifying other texts by Shakespeare and Marlowe should not be easy using the same features.

The classification results for non-Shakespearean author candidates turned out to be highly accurate (Johnson % 100, Chapman % 92.9 and Middleton %88.9). The results supported the hypothesis that writing styles of Marlowe and Shakespeare were as distinguishable as other authors unless Marlowe did not show a linear change in style over time. Meaning, Marlowe has found not to be the authors of Shakespearean writings [11].

One major application of authorship attribution has also been identifying the author of newly found texts and there has been substantial work done in related areas. One of the earliest papers in this area is about newly found poem called "Taylor Poem" and whether it is actually written by Shakespeare or not. Thisted and Efron have suggested three tests for authorship: based upon the number of new words observed, number of rare words observed and a slope test that uses Poisson regression to combine data. The result of their test is that Taylor poem is found to fit previous Shakespeare writings [12]. Thompson and Rasp have also suggested adding "uniformity test of various p-values" that is computed directly from Poisson probability distribution in addition to Thisted and Efron's approaches, and then applied to The Dark Tower which was attributed to C.S. Lewis when it was published. Their conclusion is also consistent with the claim that Lewis is not the author of The Dark Tower unless he had written the book in a bad day or his writing style was changed drastically [6]. Another interesting study on the unknown texts is also done based on word-level features, vocabulary richness and syntactic features by using

Liblinear SVM for classification purposes [13]. Even though the classification accuracy results are not as high as other related works features like "number of unique words" should be noted for use in any attribution problem.

Authorship of biblical texts has also been a part of previous studies. The New Testament of the Bible that consists of 27 books, 13 of which are contributed by St. Paul is computationally studied by using topic modeling and affinity propagation clustering. All Pauline letters are found to be divided into six groups which also match with church tradition. An anonymous letter in the New Testament, the letters to Hebrews, is also compared with Pauline letters and it was found that the book of Hebrews was not authored by Paul [14]. More detailed analysis of "the Letters to Hebrews" is done by considering the word recurrence interval technique, trigram Markov method, stylometric measures such as the frequency of function words, and multiple discriminant analysis of frequency of function words. Mahalanobis distance is used for the comparison of text centroids. It is concluded that Hebrews is not likely to be written by Paul, Matthew, Mark, Luke, or John yet it's stylistically very similar to texts written by Barnabas [15].

The performances of these various features and classification methods are also analyzed and experiments are performed on newspaper articles. Stylometric analysis of texts such as number of sentences, words, commas, colons, semicolons, incomplete sentences, periods in an article, vocabulary diversity or richness measure, bag of words representation and frequency of function words are extracted as features. During these experiments histogram method, K-nearest method and parzen windows, Bayes Classifier, k-means clustering, support vector machines with $tf - idf$ approach and combination of classifiers are applied as classifiers. Best results are observed with Gaussian classifiers by using stylometric features and function words. Support vector machine classifiers also perform well with bag of words feature set [8].

Usefulness of function words in authorship attribution is introduced by Mosteller and Wallace in their work on Federalist papers [16]. Argamon and Levitan has compared the characteristic features of frequent words, pairs and collocations using the

SMO algorithm, and implemented it for two class (American or British) author nationality classification problem. Their results conclude that function words are useful as stylistic text attribution and frequent words are the best features among others. The reason behind it is that a given same size frequent collocations has less different words comparing to frequent words so it carries less discriminatory features [7].

In summary, there has been substantial work done in authorship attribution and mainly people in forensic linguistic or computer scientists aim to build "stylistic fingerprint of author" by using several features of a given text such as function words, stylometry. It is a classification problem and several classifiers are used such as Naïve-Bayes, SVM. Among them SVM is observed to fit best for these kinds of problems.

2. ADVANCES IN NATURAL LANGUAGE PROCESSING

Natural language processing (NLP) is the machine activity of analyze, understand, alter, or generate natural language data. One of the earliest studies was the Georgetown experiment in which Russian sentences were translated into English by machine interpretations [17].

Authorship attribution problems are also a part of NLP and the advances in NLP proportionally affect the developments of new algorithms. In this chapter, recently developed NLP techniques, publicly available dataset will be introduced to get the attention of researchers in different domains.

2.1 NLP Tasks

In order to extract features from a given text and use it in computations, the preprocessing of the text must be done. The outline of the work can be application specific. On this regard, the most recent developed techniques will be introduced.

Stemming

Because of the grammatical reasons, in text documents words coming from same root utilized differently based on their usage as adjectives, adverbs, nouns, tenses of the verbs. For example, “am, is, are” come from the same root as “be”, “democracy, democratic, democratization” are different deviation of same word “democracy”, or “go, went, gone” are different inflicted forms of “go”.The objective of stemming is to reduce the variation of related words to same stem. Using Porter’s algorithm in Python [18], stemming can be accomplished as follows:

```
#Use "pip install stemming" to install the package
from stemming.porter2 import stem
stem("democratization")
>>Out: 'democrat'
```

Lemmatization

In computational linguistics, Lemmatization is closely related to stemming. The major difference is that stemming operates without the knowledge of the context of the word whilst in lemmatization task it depends on identifying the meaning of word by utilizing "part of speech". Python's NLTK library can be deployed for this task. The "pos" variable in the function can be changed and same task can also be achieved by using different libraries such as "spacy", or "Standford NLP".

```
from nltk.stem import WordNetLemmatizer
my_lemmatizer = WordNetLemmatizer()
my_list = ['good', 'better', 'best']
for word in my_list:
    print(my_lemmatizer.lemmatize(word, pos="n"))

>>Out: good
better
best
```

Part of Speech Tagging

It is process of analyzing sequence of words and attaching a category for each word in the sequence. It is an important task during text-to-speech processing since the pronunciation of the word needs to be known before the speech task.

```
import nltk
nltk.download('averaged_perceptron_tagger')
from nltk import word_tokenize
my_text = word_tokenize(
"I am writing my thesis to graduate from my master degree")

nltk.pos_tag(my_text)
>>Out:
[('I', 'PRP'),
 ('am', 'VBP'),
 ('writing', 'VBG'),
 ('my', 'PRP$'),
 ('thesis', 'NN'),
 ('to', 'TO'),
 ('graduate', 'NN'),
 ('from', 'IN'),
 ('my', 'PRP$'),
 ('master', 'NN'),
 ('degree', 'NN')]
```

Some of the abbreviations in the result are present tense verb (VBP), noun (NN), preposition (IN) that give information regarding the grammatical structure of the sentence. "NLTK", or "spacy" can both be used to perform POS tagging task.

Named Entity Recognition

It is the task of identifying entities in a given text and categorizing each of these entities as person, organization, date, location, time, etc. It is a crucial task since the word "apple" in "I want to eat apple" refers to the fruit but in "I want to work for Apple" refers to the company.

```
from nltk.tag import StanfordPOSTagger
from nltk import word_tokenize
jar = 'path/stanford-postagger.jar'
model = 'path/english-left3words-distsim.tagger'
pos_tagger = StanfordPOSTagger(model, jar, encoding='utf8')
sentence1 = "I want to eat apple now and "
sentence2 = "I want to work at Apple in the future."
sentence = sentence1 + sentence2
my_text = pos_tagger.tag(word_tokenize(sentence))
print(my_text)

#Or Second method
nltk.download('maxent_ne_chunker')
nltk.download('words')
from nltk import word_tokenize, pos_tag, ne_chunk
print (ne_chunk(pos_tag(word_tokenize(sentence))))
>>Out:
(S I/PRP, want/VBP, to/TO, eat/VB, apple/NN, now/RB,
and/CC, I/PRP, want/VBP, to/TO, work/VB, at/IN,
(ORGANIZATION Apple/NNP), in/IN, the/DT, future/NN, ./.)
```

Sentiment Analysis

It is a vast range of applications in NLP where the positivity, negativity or neutrality of a sentence is being investigated. The applications could be movie reviews of “rotten tomatoes”, customer product review or determining the mood of a speaker via voice analysis.In order to showcase an example, using Naive-Bayes Classifier a simple model is being built. In the equation 2.1, the important task is identifying the

number of words whether they're positive, negative, or neutral. To achieve this goal, we can create a simple database of our own choosing to identify the words.

$$Positivity = \frac{\text{Number of Positive Words}}{\text{Number of All Words}} \tag{2.1}$$

$$Negativity = \frac{\text{Number of Negative Words}}{\text{Number of All Words}} \tag{2.2}$$

There is an example case of two sentence and their sentiment analysis measurement result using the algorithm developed here [1].

Table 2.1.: Positivity and Negativity Score of Sentences

Sentences	**Positivity**	**Negativity**
"Awesome movie, great actors, I liked it"	0.71	0.14
"The sound effects were bad, terrible movie"	0.27	0.43

Semantic Text Similarity

It is the task of measuring the degree of equivalence in the underlying semantics of given text pieces. Words can also be similar in two ways: lexically and semantically. The categorization of text similarity approaches can be put into three as: String-based, Corpus-based and Knowledge-based similarities [19]. The work of Wael and Aly is a good survey on introducing the available text similarity approaches [19]. You can look up to words or compare sentence similarities using both NLTK or Gensim. The following word snippet is simply showing you the words that are close to "book".

```
from nltk.corpus import wordnet as wn
print(wn.synsets('book', 'n'))
>>Out:
```

[1] https://github.com/agungor2/Authorship_Attribution/blob/master/Sentiment_Analysis.ipynb

```
[Synset('book.n.01'), Synset('book.n.02'), Synset('record.n.05'),
Synset('script.n.01'), Synset('ledger.n.01'), Synset('book.n.06'),
Synset('book.n.07'), Synset('koran.n.01'), Synset('bible.n.01'),
Synset('book.n.10'), Synset('book.n.11')]
```

Text Summarization

It is the process of shortening a given long text while keeping the informative part. The main goal of the task is to keep as much information as possible from the long text and represent it in as short format as possible. The repository represented by Google Brain team is an extensive work in text summarization using sequence to sequence model and rouge score is being utilized to measure the model performance [20]. It is also implemented in Gensim module and easy use case is being provided below.

```
from gensim.summarization import summarize
sentence = "User choice" #Replace it with your choice
print(summarize(sentence))
```

Word Embeddings

The area of word embeddings has been vastly growing since its first introduction by Mikolov [21]. He introduced Skip-gram model which was an efficient method for learning high quality vector representations of words from large amounts of text data. One of the main advantages of Skip-Gram model is that it can capture different semantics of same words such as Apple as a company or as a fruit. Secondly and most importantly, it allows words to be represented in a vector form which would then allow some fascinating data manipulation to be discovered. For example, the result of a vector calculation vec(“Madrid”) - vec(“Spain”) + vec(“France”) can be calculated with word embeddings and the resulting vector can also be searched within the vector space to find the nearest word to it. In Mikolov’s work [21], the result has been found

that the resulting vector is closer to vec("Paris") than to any other word. Some of the fundamental data handling techniques with word embeddings are given below as a sample.

```
#1. Load pre-trained models: It allows dealing with
#previously trained text data and using their word vectors.
#The vectors are more accurately represented because
#they're being trained on large corpus of text data.
#Some of the available pre-trained sets:
#Google's Word2Vec
#Glove Word2Vec(dimension of Word2Vec is from 50 to 300)
#fastText
#LexVec
#Meta-Embeddings
from gensim.models import Word2Vec
model = gensim.models.KeyedVectors.load_word2vec_format(
'GoogleNews-vectors-negative300.bin', binary=True)

#2. Extracting word vectors from the imported model
print(model['NLP'])

#3. We can do vector operation and find the closest vector
#vec(\Madrid") - vec(\Spain") + vec(\France")
#The numbers following the words show the similarity measure
print(model.most_similar(positive=['Madrid', 'France'],
negative=['Spain']))

>>Out: [('Paris', 0.75), ('Marseille', 0.61),
('French', 0.60), ('Colombes', 0.59),
('Hopital_Europeen_Georges_Pompidou', 0.58),
```

```
('Toulouse', 0.57), ('Parisian', 0.57),
('Cergy_Pontoise', 0.56), ('Marseilles', 0.55),
('Strasbourg', 0.54)]

#4. Cluster analysis
print(model.doesnt_match("breakfast lunch dinner milk".split()))
>>Out:milk

print(model.doesnt_match("China Turkey France Italy".split()))
>>Out:China

#5. Train your own model
import pandas as pd
my_text = pd.read_csv("my_text.txt", header=["text"])
model = gensim.models.Word2Vec(my_text, min_count=1,
size=300,workers=4)
```

The techniques introduced here are some of the fundamental ones that can help beginners kick start their analysis. Using these strategies one can build application specific more sophisticated NLP toolbox of his own.

2.2 Data Availability

In every data analytics task, one of the main necessities to get a perfect result is to have a large enough and suitable dataset that can be worked on. Thanks to the advances in social media, optical character reader machines, and several projects that aim to make documents available online. We are filled with large corpuses of text data sets publicly available. In order to inspire and inform the readers the existence of such big corpuses, they are categorized and stated as follows.

- **Text Classification:** It refers to labeling texts documents such as filtering emails as spam or movie reviews sentiment analysis.
  - Apache Software Foundation Public Mail Archives
  - Stanford Collection of Amazon Reviews
  - Reddit Comments
  - Wesbury Lab Usenet Corpus
  - Reuters Newswire Topic Classification Data
  - Yelp Reviews
  - IMDB Movie Review
  - Stackoverflow and Twitter Sentiment analysis data
- **Language Modeling:** There are several application specific texts that are used to generate models.
  - Common Crawl
  - Yahoo! N-Grams
  - Google Books Ngram Corpus
  - American National Corpus
  - Wikipedia XML Data or Dbpedia
  - ClueWeb09
  - Personae Corpus
  - Turkish National Corpus
  - NTU-Multilingual Corpus
  - Open Multilingual Wordnet
- **Authorship Attribution:** It is, as described before, the task of identifying the author of a given text.

  - Project Gutenberg
  - GDELT
  - C50 (subset of RCV1) and PAN dataset

- **Text Summarization:** It is the task of creating short and meaningful text after analyzing a large document.
  - TAC Dataset
  - Australian Legal Cases
  - RTLTDS Dataset
  - 17-timelines Summarization Dataset
  - ArXiv Summarization Dataset

2.3 Neural Networks

A neural network is a computational nonlinear model that is inspired from the neural structure of the human brain. It consists of three interconnected layers as input, hidden and output layers. Every neuron has weighted inputs, an activation function which could be a linear, step, sigmoid, or rectified linear function, and one output.

One of earliest studies of language models using neural network techniques to calculate the joint probability function of sequence of words was introduced more than a decade ago by Bengio [22]. In his work, he states that one of the difficult problems of language modeling is the curse of dimensionality when one wants to find out the joint distribution of consecutive words because of the fact that there are enormous amount of free parameters to consider. The cure to this problem lies within the structure of sentences. A language model can be represented by the conditional probability of the next word given all the previous ones in the sequence [22]. His idea has then led to the development of several different NLP techniques that are categorized as follows.

Multi Layer Perceptron

It has three or more layers and helps classify data that is not linearly separable. It is mainly utilized in speech recognition and machine translation tasks. Using MLPs for phonetic event detection system is introduced by researchers from Microsoft in [23] and the performance improvement of speech recognition systems on the Broadcast News task has been achieved using Segmental Conditional Random Fields.

In order to show case the usage of multi layer perceptron on a text classification task, a sample tutorial workload has been created [2]. The main idea is to create a sentence vector representation of a given text by utilizing Google's pre-trained datasets and "Gensim" to semantically analyze comments. In order to build the model, hidden layer size has been set to 5. To train the model the following sentences are being used as training set "I hate this movie. It is horrible.", and "Great movie and amazing actors.". The negative sentence is classified as "0" and positive one as "1". After training the model, to test the model "Terrible and useless movie.", and "Fantastic. I love it." are used to predict. Their results were found to be matching the expected outcome.

Convolutional Neural Network

It contains one or more convolutional layers and uses a variation of multilayer perceptrons. The layers in the network use a convolution operation to the input passing the result to the next layer which then makes the network to be deeper with less parameters. In [24], the author presents a model built on top of Word2Vec, tests the model with several other benchmarks such as movie reviews, or Standford Sentiment Treebank. With little tuning of hyperparameters, a simple CNN structure with one layer of convolution improve the performance [24]. Building CNN model

[2] https://github.com/agungor2/Authorship_Attribution/blob/master/Classification_MLP.ipynb

for a text classification task can be easily possible by Mxnet, Tensorflow, or Keras libraries.

Recurrent Neural Network

Unlike a feed-forward neural network, connections between neurons make a directed cycle which results in the output to be affected both by present inputs and the previous step's neuron state. In the earliest work of Mikolov, he introduced his language model based on the RNN structure [25]. He later described his Skip-gram and CBOW language model which outperformed better than RNN model [21]. RNN structure can also be used to perform text classification task as in [26]. Their model outperformed traditional methods such as SVM, LDA, or CNN on the experimented 4 datasets that are 20Newsgroups, Fudan set, ACL Anthology Network, and Stanford Sentiment Treebak.

Long Short-Term Memory

It is also a RNN structure but it is designed to model temporal sequences and their long-range dependencies more accurately than normal RNN. It does not have an activation function nor the gradient vanishes during the training. It is implemented in the units of "blocks". It is a highly appreciated method by the researchers in different domains and in [27] authors show the achievement of the state of-the-art performance for large scale acoustic modeling using LSTM. Using Keras library and IMDB movie review dataset a simple LSTM structure has been built to test out the model. More details of the model can be found on this tutorial [3]

[3] https://github.com/agungor2/Authorship_Attribution/blob/master/LSTM_classification.py

Sequence to Sequence Models

It can be imagined as an encoder-decoder architecture system where both of encoder and decoder is a RNN structure. It is mainly used in chat-bots and machine translation systems. Utilizing LSTM on both side of the structure with limited data is being implemented in [28] and the performance of the system has outperformed traditional systems with large-scale training data. This suggests that it should do well on many other sequence learning problems. Building a sequence to sequence model is simple using Tensorflow. The example below shows how to create a simple encoder and decoder structure in Tensorflow.

```
import tensorflow as tf
# Build RNN cell for encoder
encoder_cell = tf.nn.rnn_cell.BasicLSTMCell(num_units)
# Build RNN cell for decoder
decoder_cell = tf.nn.rnn_cell.BasicLSTMCell(num_units)
# Build Helper
helper = tf.contrib.seq2seq.TrainingHelper(
    decoder_emb_inp, decoder_lengths, time_major=True)
# Build Decoder
decoder = tf.contrib.seq2seq.BasicDecoder(
    decoder_cell, helper, encoder_state,
    output_layer=projection_layer)
#For the optimizer, you can use AdamOptimizer as a start
```

2.4 Cloud Computing

Some of the applications in text mining or NLP might require more storage than an individual computer can handle or it might need more powerful tools. To ease the pain of overcoming such large scale operation issues, cloud services and APIS provide solutions to researchers. Top providers of such services are Microsoft Azure, IBM

Bluemix, Amazon Elastic Compute Cloud (AWS), Verizon Cloud and Google Cloud platform.

There are multiple different ways one can make effective use of such services in text mining tasks. For example, using Amazon's Natural Language Processing service, also called Comprehend, one can start analyzing the datasets such as support emails, social media posts, online comments, telephone transcriptions to understand the positivity and negativity factors or it can be used to build a semantic search engines rather than basic keyword one. AWS also provides Machine learning services that anybody can build their classifiers or regression models based on the desired task. IBM Bluemix can also be used to analyze text data to extract meta-data from content such as concepts, entities, keywords, categories, relations and semantic roles. Another practical use case of Bluemix is that it could be used to develop data-driven chatbot systems. Dialogflow is also another great tool to start building a data-driven chatbot system.

AWS, Bluemix, Dialogflow are tools that are meant for entry level NLP analysts to build their own models. In order to further make use of Cloud computing systems one can make use of Kaggle website or Google Apis. Kaggle has servers that can help you analyze gigabytes of data in fast pace and build your model on the language of your choosing. This would enable you not to worry about operating system library installation issues, gives you large storage availability and help you run your program faster than your own machine. Google has also provided several Apis such as text to speech, Google Maps Api, Natural Language processing Api, Youtube Analytics Api. In order to show case a simple use of Google Api with Python an example case is shown below. It takes input parameter as the longitude and latitude of one person's starting point as well as the final destination point, then returns the time spent on the overall trip. The workload below has been used by the author to create a parking meter finder by the author and more can be found in the following link [4].

[4] https://youtu.be/H6A4cQFhHWo

```
import httplib2
import json

def getdistance(x1,y1,x2,y2,mode=""):
    command = "https://maps.googleapis.com/maps/api/distancematrix"
    +"/json?units=imperial&origins="+ str(y1) +","+ str(x1)
    +"&destinations="+ str(y2) +","+ str(x2) +"&mode=" + mode +
    "&key=put_your_key_here"

    #Parse the web page
    http = httplib2.Http()
    status, response = http.request(command)

    #print response
    el = json.loads(response).get("rows")[0].get("elements")[0];

    if ("duration_in_traffic" in el.keys()):
        return int(el.get("duration_in_traffic").get("value"))
    else:
        return int(el.get("duration").get("value"))
```

Google NLP Api can also be used to do sentiment analysis in Python. A simple example model has been built in this tutorial to help readers take the initial step in building Api supported tools [5].

[5] https://github.com/agungor2/Authorship_Attribution/blob/master/Google_NLP_api.py

3. DATASET PREPARATION

From a machine learning perspective, AA is a multi-class, single-label text classification task. However, it is a bit different from other text classification problems since the style of writing and text content are important, too. One of the challenges here is to identify and create a dataset without introducing any bias to the problem to be studied. Writing style of an author can vary due to following top reasons:

- The century that the author is living and writing his works: General writing styles may change significantly even throughout a century due to changes in literature or word usages.
- The idea movement that the author has been inspired from: Authors are usually being affected by movements such as romanticism, modernism, or symbolist movement which then also affect their writing style.
- The language in which the author is writing: Every language has its own grammatical structures. The writing style of a multi-lingual author is different than the mono-lingual writer. If a book is also being translated from its original language to another one, this translation also creates linguistic differences.
- The topics that the author likes to write about such as science fiction, love, drama: Usage of words changes across different topics, which could make it easier to identify authors based on word usage.

In this context, there are different problems that can be studied such as the classification of a given text based on the century it was written, classification of a given text as translated or original, or simply choosing authors from the same century, same idea movement, same topics, and classify text according to their authors.

In the beginning of this thesis workload, we aimed at achieving three goals. Firstly, we wanted to create practical features using traditional and new methods that could help us with author classification task. Secondly, we wanted to create a translated book identification problem by collecting books that are translated from non-English language text domains to English. Lastly, we plan to do an exploratory analysis on the style differences of authors from 19th century to 20th century. However, due to time limitations we have focused only one side of the problem that is the identification of the author of a given text. If the researchers are interested in conducting an analysis of English translated texts from French, German, Russian, or mixed language, they can find more details here [1] and can contact the author of this thesis for the whole raw dataset. The list provided consists of book names translated from other languages and the number of words each book has in it. There are totally 200 French, 200 German, 448 Russian, and 89 mixed languages translated book in the whole text corpus.

3.1 50-Author Dataset

The GDELT Project is one of the largest publicly available digitized book database which has more than 3.5 million books published from 1800-2015. The GDELT Project is an open platform for research and analysis of global society and thus all datasets released by the GDELT Project are available for unlimited and unrestricted use for any academic, commercial, or governmental use of any kind without any fee [29]. The whole digitized dataset is publicly available and interested researchers can freely perform SQL queries using the Google big query platform. For example; the book names, publication year, quotations, themes, the original text of the book of "Mark Twain" which were written between 1890 to 1900 can be found as follows using the Big query platform of Google.

[1] https://github.com/agungor2/Authorship_Attribution/blob/master/Translated_list.txt

```
SELECT Themes, V2Themes, Quotations, AllNames, TranslationInfo,
BookMeta_Identifier, BookMeta_Title, BookMeta_Creator,
BookMeta_Subjects, BookMeta_Year,

FROM (TABLE_QUERY([gdelt-bq:internetarchivebooks],
'REGEXP_EXTRACT(table_id, r"(\d{4})") BETWEEN "1890" AND "1900"'))

WHERE
  BookMeta_Creator CONTAINS "Mark Twain"
LIMIT 50
```

To decrease the bias and create a reliable dataset the following criteria have been chosen to filter out authors: English language writing authors, authors that have enough books available (at least 5), 19th century authors. With these criteria 50 authors have been selected and their books were queried through Big Query Gdelt database. The next task has been cleaning the dataset due to OCR reading problems in the original raw form. To achieve that, firstly all books have been scanned through to get the overall number of unique words and each words frequencies. While scanning the texts, the first 500 words and the last 500 words have been removed to take out specific features such as the name of the author, the name of the book and other word specific features that could make the classification task easier. After this step, we have chosen top $10,000$ words that occurred in the whole 50 authors text data corpus. The words that are not in top $10,000$ words were removed while keeping the rest of the sentence structure intact[2]. Afterwards, the words are represented with numbers from 1 to $10,000$ reverse ordered according to their frequencies. The entire book is split into text fragments with 1000 words each. We separately maintained author and book identification number for each one of them in different arrays. Text segments with less than 1000 words were filled with zeros to keep them in the dataset as well. 1000 words make approximately 2 pages of writing, which is long enough to extract

[2] https://github.com/agungor2/Authorship_Attribution/blob/master/Clean_data.py

a variety of features from the document. The reason why we have represented top $10,000$ words with numbers is to keep the anonymity of texts and allow researchers to run feature extraction techniques faster. Dealing with large amounts of text data can be more challenging than numerical data for some feature extraction techniques.

Table 3.1.: Author Book Number Distribution

Author ID	**Author Names**	**Book Numbers**	**Total Words**
1	Arthur Conan Doyle	16	1394980
2	Charles Darwin	10	513085
3	Charles Dickens	7	368314
4	Edith Wharton	27	2487978
5	George Eliot	22	1064661
6	Horace Greeley	9	624680
7	Jack London	16	2216225
8	James Baldwin	76	10220446
9	Jane Austen	12	1617346
10	John Muir	20	1162637
11	Joseph Conrad	8	484102
12	Mark Twain	12	988713
13	Nathaniel Hawthorne	15	815377
14	Ralph Emerson	28	3936896
15	Robert Louis Stevenson	21	1976693
16	Rudyard Kipling	10	402301
17	Sinclair Lewis	14	1051828
18	Theodore Dreiser	16	1839202
19	Thomas Hardy	22	2078753
20	Walt Whitman	10	839168
21	Washington Irving	62	3426646

Continued on next page

Table 3.1 – continued from previous page

Author ID	Author Names	Book Numbers	Total Words
22	William Carleton	16	696567
23	Albert Ross	16	844634
24	Anne Manning	9	827145
25	Arlo Bates	18	1531925
26	Bret Harte	64	6406984
27	Catharine Maria Sedgwick	18	453158
28	Charles Reade	20	1170720
29	Edward Eggleston	6	806316
30	Fergus Hume	20	1291603
31	Frances Hodgson Burnett	43	3464120
32	George Moore	9	1174176
33	George William Curtis	19	2180926
34	Helen Mathers	17	806600
35	Henry Rider Haggard	13	923255
36	Isabella Lucy Bird	20	1170895
37	Jacob Abbott	32	3316171
38	James Grant	30	1620580
39	James Payn	54	3667172
40	John Kendrick Bangs	12	665969
41	John Pendleton Kennedy	14	1251338
42	John Strange Winter	13	1358015
43	Lucas Malet	12	1677565
44	Marie Corelli	16	797761
45	Oliver Optic	36	3484099
46	Sarah Orne Jewett	16	1105534
47	Sarah Stickney Ellis	74	5296691

Continued on next page

Table 3.1 – continued from previous page

Author ID	Author Names	Book Numbers	Total Words
48	Thomas Anstey Guthrie	30	2740952
49	Thomas Nelson Page	15	1217070
50	William Black	18	1597066

In Table 3.1, author id number in the dataset, author name, total number of books, total number of wordings after filtering out less occurred words have been listed out. The distribution of author text pieces in the training data has been provided in Figure 3.1. In the training data, it has found that James Baldwin has the most text pieces with 6914 whilst Rudyard Kipling has the least with 183 pieces.

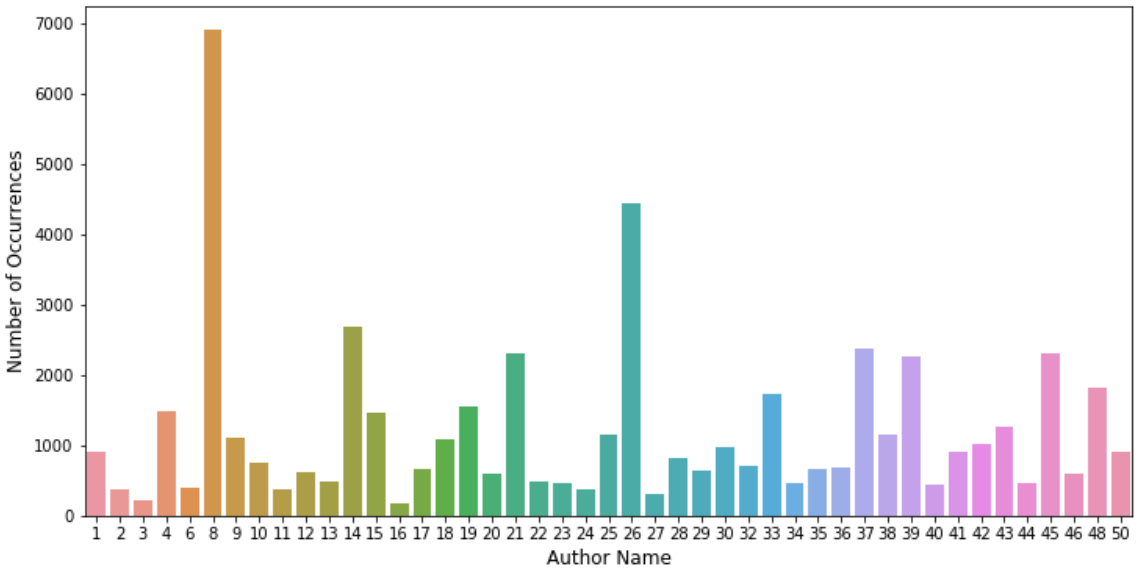

Figure 3.1.: Distribution of Authors in Training Set

In order to make the AA problem more realistic, when splitting the training and testing dataset the writings of George Eliot (5), Jack London (7), Frances Hodgson Burnett (31), Sarah Stickney Ellis (47), Thomas Nelson Page (49) have been removed

from the training and added to testing set. This creates a non-exhaustive training set with 45 authors while the test set contains 50 authors. In total this leads to 53678 training data instances, 39922 testing data instances. Each of these instances consist of 1000 words text fragment. Another important aspect we have considered while splitting training and testing data is to keep all fragments of the same book either in the training or testing dataset. This way we do not end up training and testing on the same books. Without this restriction the classification task would be much simpler and simple bag of words representation with SVM would give much higher F-1 scores. To study the sentence structure and writing style from a different perspective, the same steps also have been applied to create another dataset in which stop words have been removed while keeping the word order same for the remaining words.

3.2 3-Author Dataset

The dataset we have provided consists of 50 authors, 93600 instances where each instance is a 1000 word of text document. Due to the scale of our data, some analysis take long time to train and test it. To address this problem, we were able to find a smaller author identification data on Kaggle server [30].

Using CoreNLP's MaxEnt sentence tokenizer, Kaggle has chunked the works of Edgar Allan Poe (EAP), HP Lovecraft (HPL) and Mary Shelley (MWS) into sentences. Some sentences are 2-3 words and some are up to 100 words, so every instance in this dataset is very small comparing to our 50-author dataset. There are 19579 of instances for training and 8392 for testing. In the 50-author data, one of the challenges is the missing author problem but for this 3-author dataset both training and test sets have the same three authors. Another setting that matches to our earlier criteria regarding author selection is that the works of these authors are categorized as fiction writing and they all lived in 19th century.

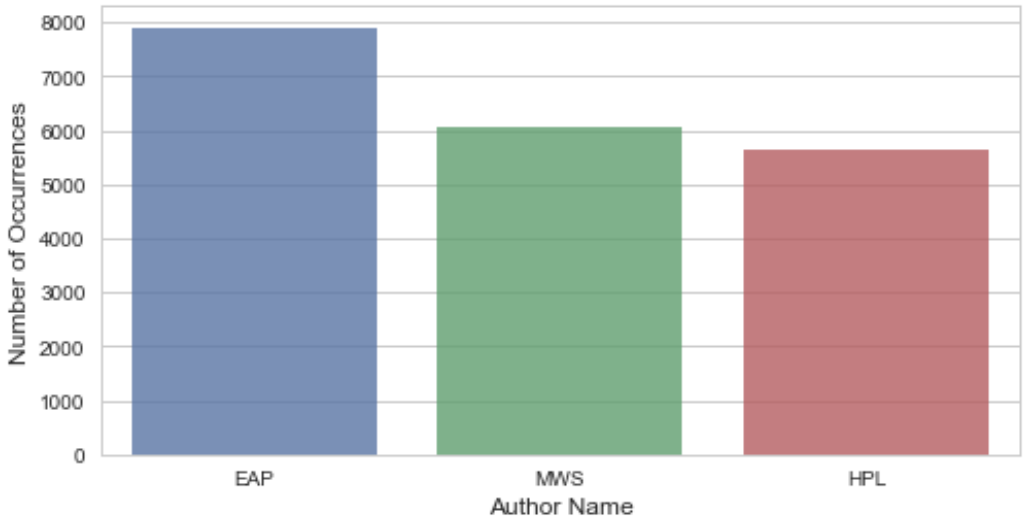

Figure 3.2.: Distribution of Authors in Training Set

The distribution of authors in the text data is shown in Figure 3.2 using seaborn and matplotlib [3]. There is a slight difference between HPL and MWS whereas EAP is dominating the other two authors.

[3] https://github.com/agungor2/Authorship_Attribution/blob/master/Author_Distribution.py

4. PRACTICAL FEATURE EXTRACTION

In order to identify the authorship of an unknown text document using machine learning the document needs to be quantified first. The simple and natural way to characterize a documents is to consider it as a sequence of tokens grouped into sentences where each token can be one of the three: word, number, punctuation mark. To quantify the overall writing style of an author, stylometric features are defined and studied in different domains. Mainly, computations of stylometric features can be categorized into five groups as lexical, character, semantic, syntactic, and application specific features. Lexical and character features mainly considers a text document as a sequence of word tokens or characters, respectively. This makes it easier to do computations comparing to other features. On the other hand, syntactic and semantic features require deeper linguistic analysis and more computation time. Application specific features are defined based on the text domains or languages. These five features are studied and the methods to extract them are also provided for interested readers.

4.1 Lexical Features

Lexical features relate to the words or vocabulary of a language. It is the very plain way of representing a sentence structure that consists of words, numbers, punctuation marks. These features are very first attempts to attribute authorship in earlier studies [7], [11], [13], [14], [31]. The main advantage of Lexical features is that it is universal and can be applied to any language easily. These features consist of bag of words representation, word N-grams, vocabulary richness, number of punctuation marks, average number of words in a sentence, and many more. Even though the number of lexical features can vary a lot, not all of them are good for every authorship attribution

problem. That is why, it is important to know how to extract these features and try out different combinations on different classifiers.

Bag of Words

It is the representation of a sentence with frequency of words. It is a simple and efficient solution but it disregards word-order information. In order to apply it on 50-author dataset, a model has been built here [1]. In the approach, for each text fragment the number of instances of each unique word is found to create a vector representation of word counts. Since there are uniquely $10,000$ words in the whole dataset you can choose the range of the words you want to use in your feature vector. We have also shown how to extract top words usage author-wise in every author's text corpus. As expected top words are determiners that every writer use while constructing an English sentence. For example, for Arthur Conan Doyle top 20 words are "the, to, of, he, a, and, i, that, in, you, was, it, she, his, her, had, as, not, with, for" but for Charles Dickens they are "the, to, of, i, and, a, in, that, it, is, he, she, be, her, you, was, as, not, with, for" in decreasing order. Even though the two sets are mostly the same the orders are different for most authors.

The main assumption with authorship attribution problems is that every authors word usage and content differs and based on these differences the work of one author can be differentiated from the other. In order to illustrate this assumption, the content and word usages for every book can be seen as a picture of some forms as illustrated for two sample books in the 50-author dataset in Figure 4.1 & 4.2 for Charles Dickens's famous book *Oliver Twist* and Mark Twain's book *Horse Tale*. Afterwards, stop words (common words) are removed and using the frequency of remaining words we have plotted it on a boy's figure and a horse's figure, respectively in Figure 4.1 & 4.2 using word-cloud library here[2]. In both figures, most frequent words are written in bigger fonts and these words are recognized firstly when looking at the figures.

[1] https://github.com/agungor2/Authorship_Attribution/blob/master/BOW_50author.html
[2] https://github.com/agungor2/Authorship_Attribution/blob/master/oliver_horse.py

Figure 4.1.: Oliver Twist, Charles Dickens

The same methodology can be applied to 3-author dataset as well. Using Vectorizer[3] a simple model is being shown on the tutorial. It simply takes the whole corpus and vectorize it based on the frequency of each word. It is found that in the training set, there are 25068 unique words whereas in the testing set the number of unique words is 17546. It is also shown how to extract total number of appearance in the whole corpus. As an example a search has been done on "Frankenstein" and it is found to appear "6301" times. In order to apply the same methodology to 50-author dataset a conversion algorithm has been provided for readers [4].

[3] https://github.com/agungor2/Authorship_Attribution/blob/master/bow_3authors.py
[4] https://github.com/agungor2/Authorship_Attribution/blob/master/convert_data.py

Figure 4.2.: Horse Tale, Mark Twain

One important aspect of the authors in our dataset is that they are English language writers. However, some of these authors like Charles Dickens or William Black are originally from United Kingdom and some like Thomas Nelson Page are from United States. There are slightly differences between British English and American English. These differences also affect the usage of some of the words in the author's work. Words in American English such as "humor, color, honor, endeavor, theater" are used as "humour, colour, honour, endeavour, theatre" in British English. Using these features could improve the accuracy of the model and a simple algorithm has been provided to use it with Word2Vec here [5].

[5] https://github.com/agungor2/Authorship_Attribution/blob/master/word2vec_extract_data.py

Word N-grams

It is a type of probabilistic language model for predicting the next item in the form of a (n-1) order. Considering n-grams are useful since Bag of words miss out the word order when considering a text. For example, a verb "take on" can be missed out by Bag of words representation which considers "take" and "on" as two separate words. N-gram also establishes the approach of "Skip-gram" language model. An N-gram is a consecutive subsequence of length N of some sequence of sentence while a Skip-gram is a N-length subsequence where the components occur at a distance of at most k from each other [21].

In order to extract N-grams from a given text data a model has been built and tested on 50-author and 3-author dataset[6]. Stop-words have not been considered while constructing the N-grams. In 3-author dataset, for Edgar Allan Poe "upon, let us, general john b, general john b c" are the most occurrent uni-gram, bi-gram, three-gram, and four-gram respectively whilst in Mary Shelley they are " one, lord raymond, let us go, nearest town took post" and in HP Lovecraft they are "one, old man, heh heh heh, oonai city lutes dancing". Features such as "lord raymond" or "heh heh heh" could be a good identifiers as one mentions about the specific character in the book whilst the other one shows a preference of the author when it comes to phrasing an emotion.

Table 4.1.: N-Grams Distribution

Author id	**1-gram**	**2-gram**	**3-gram**
A. Doyle	would	young man	love gone astray
A. Doyle	said	sugar princess	sugar princess chapter
A. Doyle	one	marriage bond	drew long breath
W. Carleton	said	mr george	said mr george
Continued on next page			

[6] https://github.com/agungor2/Authorship_Attribution/blob/master/n-grams.py

Table 4.1 – continued from previous page

Author id	1-gram	2-gram	3-gram
W. Carleton	would	yes said	said yes said
W. Carleton	one	said mr	yes sir said
A. Manning	mr	sir john	original hymns poems
A. Manning	one	years ago	two three years
A. Manning	would	mr hill	sir walter scott

Table 4.1 contains uni-grams, bi-grams, three-grams from authors Arthur Conan Doyle, William Carleton, Anne Manning. It also provides characteristic features about the persona in their books such as "sugar princess, mr george, sir john".

Same task can be also be achieved with considering the stop-words and can be better visualized as in Figure 4.3. In this task we have simply scanned through all consecutive word pairs of two and saved it in a dictionary. After sorting the dictionary based on the frequency of each bi-grams we have plotted it on a network diagram[7].

Vocabulary Richness

It is also referred as vocabulary diversity. It attempts to quantify the diversity of the vocabulary text. It is simply the ratio of V/N where V refers to the total number of unique tokens and N refers to the total number of tokens in the considered texts [31]. In order to apply this feature to both 50-author dataset and 3-author dataset a model has been built here [8]. For 3-author dataset due to the large number of texts of Edgar Allan Poe, it was dominating the overall richness number but when normalized with the overall text number for each author the value became around 0.9. As for 50-author dataset the vocabulary richness has been found the lowest

[7] https://github.com/agungor2/Authorship_Attribution/blob/master/n-grams2.py
[8] https://github.com/agungor2/Authorship_Attribution/blob/master/vocab_diversity.py

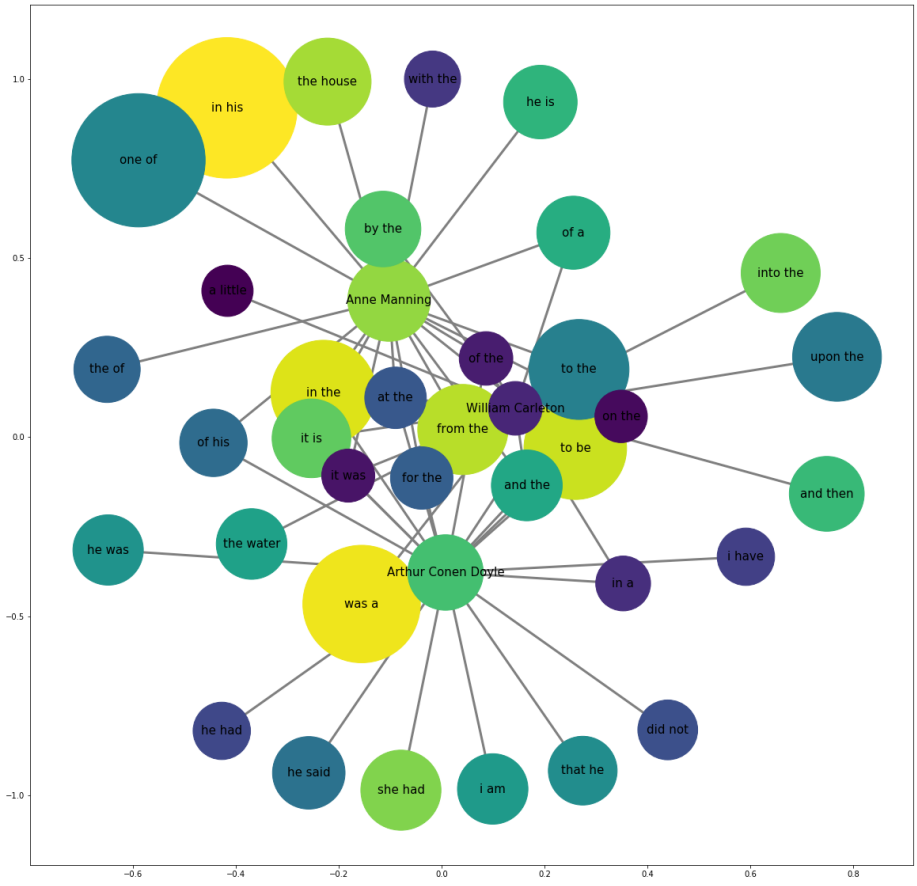

Figure 4.3.: Bigram Network Diagram for Author Id:1, 22, 24

for William Carleton and the highest for Henry Haggard. Overall distribution of vocabulary richness is plotted in Figure 4.4.

Stylometric features

These are features such as number of sentences in a text piece, number of words in a text piece, average number of words in a sentence, average word length in a text piece, number of periods, number of exclamation marks, number of commas, number of colons, number of semicolons, number of incomplete sentences, number of uppercase, title case, camel case, lower case letters. During the preprocessing of 50-author dataset we have excluded punctuations from the dataset. Hence, calculating stylometric

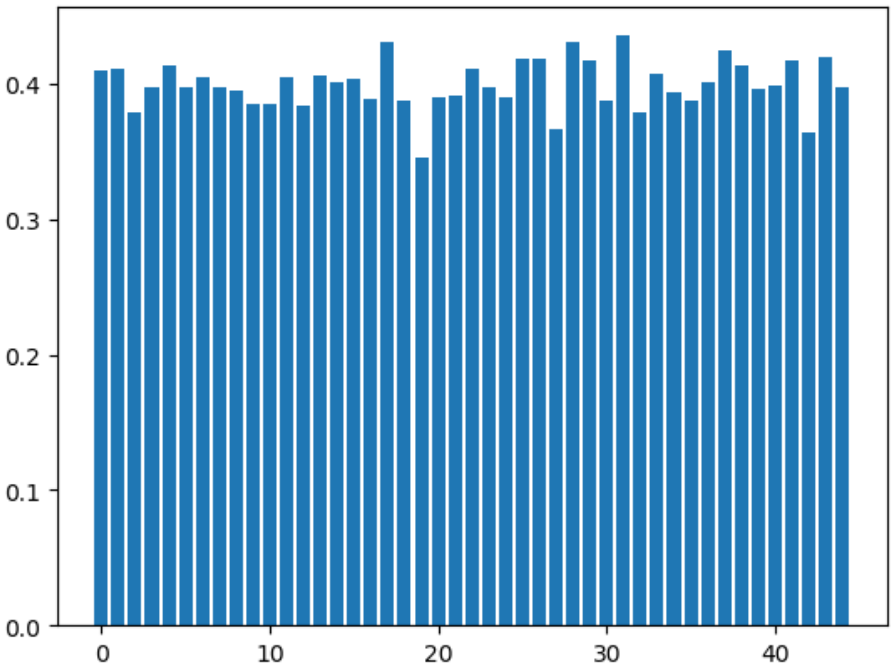

Figure 4.4.: Vocabulary Diversity for 50-Author Dataset

features are not available at the current stage for 50-author dataset. However, we can compute these features for 3-author dataset as in here [9]. Overall distribution of some of the features introduced here are applied and the resulting density measures are calculated for each author and shown in Table 4.2. Among these five features introduced, number of punctuations and number of stop words usage varies the most among the authors and hence they can be better distinguisher comparing to other feature sets.

Power Words

These are the words that affect the readers emotionally. It could be shown under the category of "semantic features", too. These words give emotional excitements or

[9] https://github.com/agungor2/Authorship_Attribution/blob/master/stylometric_features.py

Table 4.2.: Stylometric Feature Density Distributions

Features	**Authors**		
	EAP	**HPL**	**MWS**
Number of Punctuations	4.10	3.21	3.83
Number of Title Case	2.10	2.33	2.12
Upper Case Words	0.55	0.50	0.75
Average Words Length	4.64	4.63	4.60
Number of Stopwords	12.62	12.94	13.74

fears and serve as an emotional roller coaster. A great example could be given from the Winston Churchill.

"We have before us an ordeal of the most grievous kind. We have before us many, many long months of struggle and of suffering. You ask, what is our policy? I can say: It is to wage war, by sea, land and air, with all our might and with all the strength that God can give us; to wage war against a monstrous tyranny, never surpassed in the dark, lamentable catalog of human crime. That is our policy. You ask, what is our aim? I can answer in one word: It is victory, victory at all costs, victory in spite of all terror, victory, however long and hard the road may be; for without victory, there is no survival."

These frequency of such words in the database can also be studied to see how the authors are affectively using such words [10]. The power words that are in 50-author dataset is also provided and their frequency for every author has been calculated. Then their density value which is the total number of power words divided by the text fragments for every author has been recorded. Figure 4.5 shows the distribution of these recordings for every author. The writings of Jane Austen has the most power words density whilst William Carleton has the lowest usage of such words.

[10] https://github.com/agungor2/Authorship_Attribution/blob/master/power_words.py

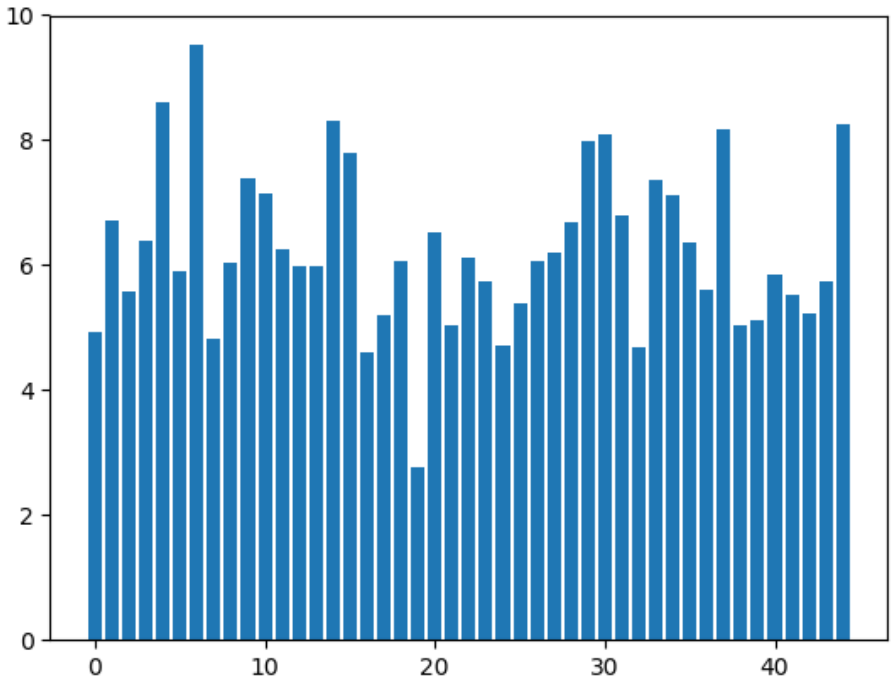

Figure 4.5.: Power Word Density for 50-Author Dataset

Function Words

Function words are the words that have little meaning on their own but they're necessary to construct a sentence in English language. They express grammatical relationships among other words within a sentence, or specify the attitude or mood of the speaker. Some of the examples of function words might be prepositions, pronouns, auxiliary verbs, conjunctions, grammatical articles. Words that are not functions words are called as content words and they can also be studied to further analysis the use case in the authorship attribution problems. In order to implement the use case of function words in our dataset a model is built [11]. The list of commonly used function words are chosen from Robert Layton's book on data mining [32] and the overall function word density measure is plotted on Figure 4.6. Among the authors

[11] https://github.com/agungor2/Authorship_Attribution/blob/master/function_word.py

in training set, George Moore has the lowest usage of function words whereas Bret Harte has the highest function words density.

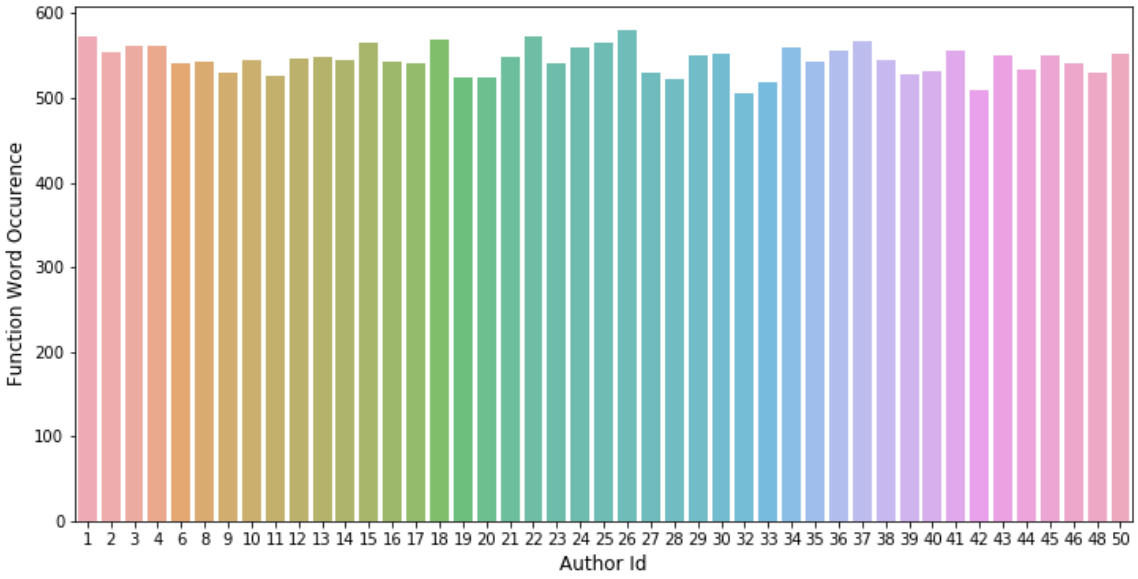

Figure 4.6.: Function Words Density for 50-Author Dataset

Tf-Idf

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

a high tf-idf score are assumed to provide the most information about that specific text [31].

$$
\begin{aligned}
TF(t) &= \frac{\text{Number of times term t appears in a document}}{\text{Total number of terms in the document}} \\
IDF(t) &= log_e(\frac{\text{Total number of documents}}{\text{Number of documents with term t in it}}) \\
Tf - Idf &= TF(t) * IDF(t)
\end{aligned}
\tag{4.1}
$$

By considering every authors texts alone within the text corpus a Tf-Idf model has been built both for 3 and 50-author datasets. [12]. In the model, not only the single forms of word tokens but their n-grams are considered as well. As for the 3-author dataset, "afforded means, aware fact, dungeon make, perfectly uniform wall" are found to have highest Tf-Idf scores for Edgar Allan Poe, "fumbling mere mistake, occurred fumbling, mistake, fumbling" for HP Lovecraft, and "looked, beneath speckled, cheering fair, cottages wealthier, counties spread, happy cottages wealthier, lovely spring" for Mary Shelley. Table 4.3 provides the top 10 words and n-grams with highest Tf-Idf scores for the 50-author dataset. Comparing between Table 4.1 & 4.3 new meaningful words have appeared that could serve as a new feature for each author such as "row, canal, passenger" for A. Doyle, or "writer, virtue, tale" for H. Greeley.

Table 4.3.: Highest Tf-Idf Pairs

A. Doyle	**C. Darwin**	**C. Dickens**	**E. Wharton**	**H. Greeley**
rowed	mr	priest	george	writer
canal	mrs	women	said george	virtue
time listen	sir john	half	girl	tale
boat	try nut	path	stray	neighbours
effect	fine lady	forest	hotel	expensive

Continued on next page

[12] https://github.com/agungor2/Authorship_Attribution/blob/master/tfidf_example.py

Table 4.3 – continued from previous page

A. Doyle	C. Darwin	C. Dickens	E. Wharton	H. Greeley
craft	old mr	dead	miss said george	tale humble
city	mr says	old square	pump room	obligation
row	don know	old square church	drive	preface
passenger	lady	settlement	pump	habits
noise	dr	square church	coach	pecuniary

4.2 Character Features

Based on these features a sentence consists of a sequence of characters. Some of the character-level features are alphabetic characters count, digit characters count, uppercase and lowercase character counts, letter frequencies, character n-gram frequencies. This type of feature extraction techniques has been found quite useful to quantify the writing style [33].

A more practical approach in character-level features are the extraction of n-gram characters. This procedure of extracting such features are language independent and requires less complex toolboxes. On the other hand, comparing to word n-grams approach the dimensionality of these approaches are vastly increased and it has a curse of dimensionality problem. A simple way of explaining what a character n-grams could be with the following example: assume that a word “student” is going to be represented by 2 character grams. So, the resulting sets of points will be “st, tu, ud, de, en, nt”.

Table 4.4.: Highest Character N-gram

A. Doyle	**C. Darwin**	**C. Dickens**	**E. Wharton**	**H. Greeley**
the	the	the	the	the
and	and	and	and	and
her	ing	ing	ing	her
ing	her	her	her	ing
hat	you	hat	hat	hat
that	ther	that	that	that
ould	with	with	ther	ther
ther	that	ould	with	with
with	ould	ther	ould	tion
thin	said	thin	here	ould

In order to apply the character n-gram models an algorithm has been developed and provided here [13]. Table 4.4 also shows the top five three and four character grams. Since most common stop words have 3 or 4 letters when constructing character level 3 or 4-grams these words also appear in Table 4.4.

Another useful feature extraction method to consider is the usage of "n't, or not" and "is, or 's". Since there is no punctuation in 50-author dataset we only apply this to the 3-author dataset. Using the *moses tokenizer* a model has been built for every author and their respected usage of "n't, not, is, 's" [14].

Figure 4.7 also shows the distribution of the usage in the training data across authors. It is found that Mary Shelley does not use "n't" but Edgar Allan Poe prefers to use "is" more often than "'s".

[13] https://github.com/agungor2/Authorship_Attribution/blob/master/char_ngram.py
[14] https://github.com/agungor2/Authorship_Attribution/blob/master/moses_tokinezer.py

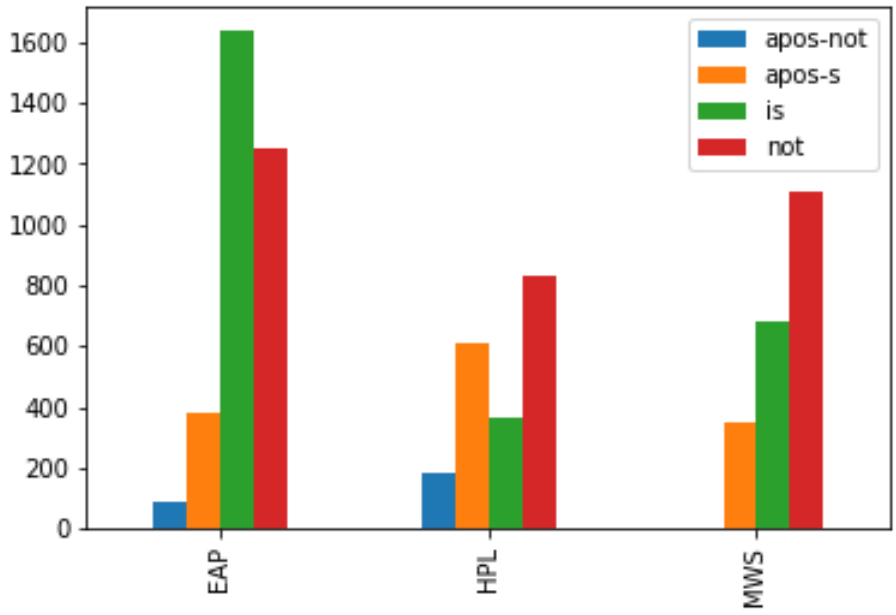

Figure 4.7.: Apostrophe Usage for 3-Author Dataset

4.3 Syntactic Features

For certain text grammatical and syntactic features could be more useful compared to lexical or character level features. However, this kind of feature extraction techniques requires specific usage of Part of Speech taggers. Some of these features consider the frequency of nouns, adjectives, verbs, adverbs, prepositions, and tense information (past tense,etc). The motivation for extracting these features is that authors tend to use similar syntactic patterns unconsciously [31].

Some researchers are also interested in exploring different dialects of the same language and building classifiers based on features derived from syntactic characteristic of the text. One great example is the work that aims to discriminate between texts written in either the Netherlandic or the Flemish variant of the Dutch language [34]. The feature set in this case consists of lexical, syntactic and word-n grams build on different classifiers and F1-score has been recorded for each cases. Employed syntactic features are function words ratio, descriptive words to nominal words ra-

tio, personal pronouns ratio, question words ratio, question mark ratio, exclamation mark ratio [34]. Some of these features can also be implemented by using 3-author or 50-author dataset.

By making use of the part of speech tagging, a model has been built to analyze the usage of adjectives, nouns, and verbs in every author's training text corpus [15]. The main steps of the model consists of tokenizing text pieces author-wise and searching through the adjective, noun, verb list among the tokens. The measure of density is being defined as number of searched element divided by the total number of tokens for every author. In 3-author dataset, it is found that Edgar Allan Poe's preferred sets of top 3 adjectives, verbs, nouns list "little, other, more, was, is, had, time man day", HP Lovecraft uses "old, great, many, was, had, were, man, night, time", and Mary Shelley's set is "own, other, many, was, had, be, life, heart, Raymond".

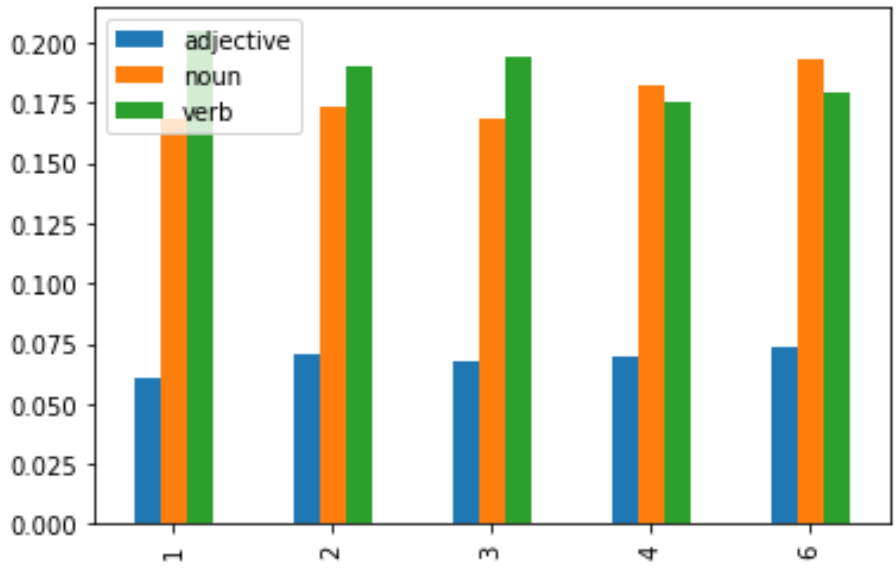

Figure 4.8.: Adjective, Verb, Noun Density for 50-Author Dataset

Same methodology has also been implemented on the 50-author dataset. Figure 4.8 shows the density measure comparison across authors who are A. Doyle, C. Dar-

[15] https://github.com/agungor2/Authorship_Attribution/blob/master/syntactic_features.py

win, C. Dickens, E. Wharton, H. Greeley. List of top five adjectives, verbs, and nouns used by every author has also been recorded in Table 4.5. Noun variations across different authors is much significant than adjective and verb variations as expected.

Table 4.5.: Top 5 Adjectives, Nouns, Verbs Usage

A. Doyle	**C. Darwin**	**C. Dickens**	**E. Wharton**	**H. Greeley**
young	good	little	little	little
little	little	good	other	poor
other	old	more	old	new
good	young	much	great	good
more	much	old	good	old
was	was	is	was	was
had	said	be	had	is
have	is	was	is	had
is	had	have	have	have
be	be	had	be	be
time	time	time	man	jane
mr	man	literature	time	mrs
man	mr	way	face	mother
way	day	mrs	eyes	man
nothing	mother	man	men	mr

4.4 Semantic Features

Features that we discussed so far aim at analyzing the structural concept of a text such. Semantic feature extraction from text data is a bit challenging. That might explain why there is limited work in this area. One example is the work of Yang who has proposed combination of lexical and semantic features for short text classification [35]. Their approach consists of choosing a broader domain related to target categories and then applying topic models such as Latent Dirichlet Allocation to learn a certain number of topics from longer documents. The most discriminative feature words of short text are then mapped to corresponding topics in longer documents [35]. Their experimental results show significant improvements compared to other related techniques studying short text classification.

Positivity, neutrality, and negativity index, and synonym usage preference are good examples of semantic features. Distributed representation of words, Word2Vec, is also an attempt to extract and represent the semantic features of a word, sentence, and paragraph [25]. The usage of Word2Vec in authorship attribution tasks has not yet been studied explicitly. Due to the application domain dependency of Word2Vec features their usage will be introduced when discussing application specific feature sets.

Positivity and Negativity Index

In order to understand the general mood and the preference of positive and negative sentence structure in each author's work, a positivity and negativity score model has been built [16]. In the algorithm, the sentences that have positive polarity score have been labeled as positive and the negative polarity scored ones are labeled as negative. In 3-author dataset, the most negative person has been found to be HP Lovecraft whereas the most positive one is Mary Shelley.

[16] https://github.com/agungor2/Authorship_Attribution/blob/master/P_N_index.py

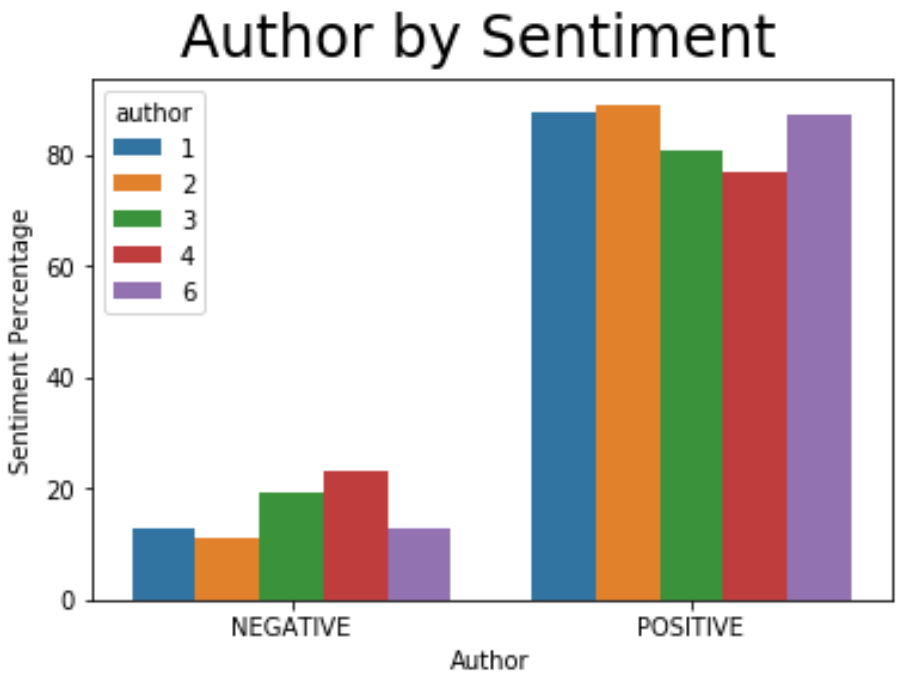

Figure 4.9.: Positivity and Negativity Comparison

In the 50-author training set, again the works of the authors A. Doyle, C. Darwin, C. Dickens, E. Wharton, H. Greeley have been chosen and their polarity scores have been calculated. Figure 4.9 shows the calculated positivity and negativity index for the chosen authors. Among these five authors the most negative one is found to be E. Wharton and the most positive one is C. Darwin.

Synonym Usage

The preference to use synonyms and antonyms in different text structure could be an identifiable feature in different tasks as well. However, extracting such features and modeling it could not be an easy task. The simple approach could be creating a domain knowledge where the pairs of synonyms and antonyms are paired. Then, a simple brute force approach can be used to find such words within a specific window size of sentences. Another approach is to employ the word vectors and represent

a sentence with the average of all word vectors in the sentence. Using these two approaches an example model has been created [17]. In the example model, given a synonym. its antonym set can be retrieved using NLTK wordnet library. Also, a similarity score can be calculated comparing the average vector forms of two sentences. Synonym and antonym word usage in 3 and 50-author dataset has not yet been studied and applied. Interested researchers can use the example model to further explore their affect in the authorship attribution problems.

4.5 Application Specific Features

When the application domain of the authorship attribution problems are different such as email messages or online forum messages, author style can be better characterized using structural, content specific, and language specific features. In such domains, the use of greetings and farewells, types of signatures, use of indentation, paragraph lengths, font color, font size could be good features [31]. Word2Vec can still be implemented in such domains as well as in 3 and 50-author dataset, but the way to use such vector forms depend on the creativity of one's approach.

Vector embeddings of words (Word2Vec)

It gives the ability to represent a word in a vector dimension of your choosing. The ways to make use of Word2Vec in 3-author and 50-author dataset is various. For example, a Word2Vec model can either be built by considering every authors text data separately, or can be imported using previously trained word vectors on other large text corpus. It can, then, be plotted into two dimensional vector space by using dimensionality reduction techniques. We built a model based on profile based Word2Vec training and using TSNE to decrease it to two dimensions[18]. In the

[17] https://github.com/agungor2/Authorship_Attribution/blob/master/synonym_example.py

[18] https://github.com/agungor2/Authorship_Attribution/blob/master/word2vec_tsne.py

model, our baseline approach is to extract A. Doyle and E. Wharton's text data and train Word2Vec on both of these authors text sets separately. Then, we have checked the word closeness for "listen" in both of these authors using 300 dimensional word vectors. Figure 4.10 shows the closest words in 2 dimensions for E. Wharton. The same comparison can also be done between pre-trained word vectors of Google or Glove to see the difference of usages in such words between an author and a pre-trained word vector.

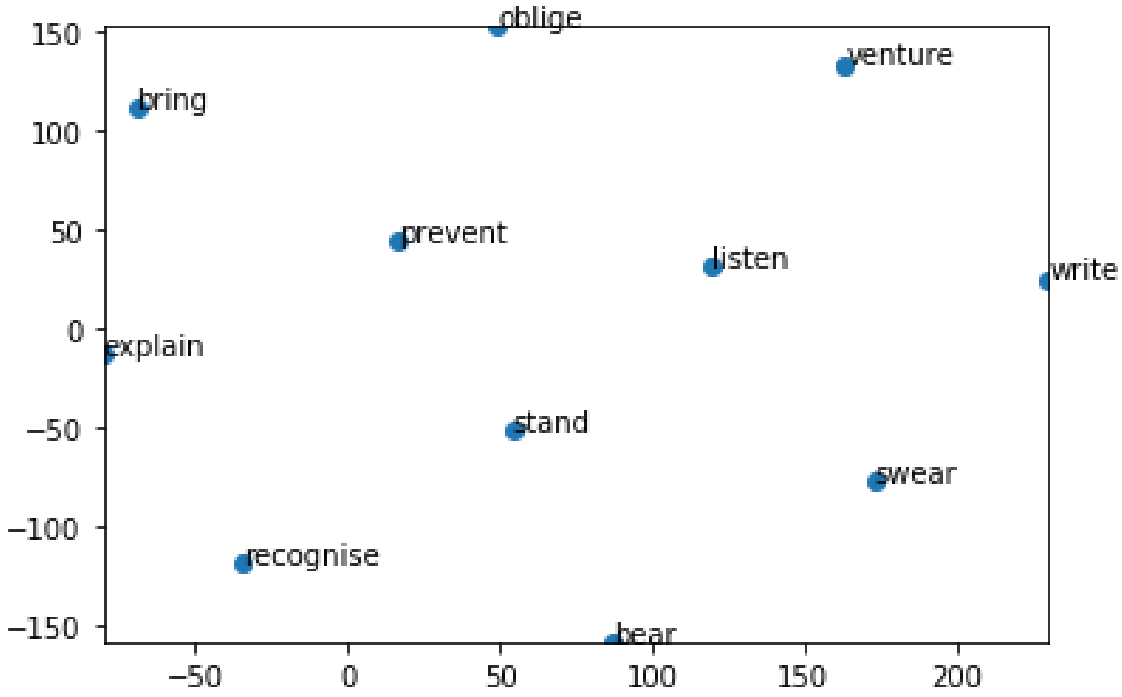

Figure 4.10.: Word2Vec 2-D Closest Words for *'listen'*

Moving with the idea of training Word2Vec per author, one can also do a cosine distance measure for the same word or same sentence. The measured cosine distance for A. Doyle and E. Wharton regarding the usage of "listen" is 0.094. In order to apply this strategy on sentence level, we can have a few ways to do so. One way is by simply taking the scaled average of Word2Vec vectors in the sentence. Another one is to employ Tf-Idf score of each word as a gain when calculating a sentence vector. We

then take the scaled average of all word vectors in the text piece. For the simplicity, we only consider taking the average vector without Tf-Idf gain for now. In this case, "her lips were parted" has been compared for both A. Doyle and E. Wharton. The cosine distance has been recorded as 0.258 which is much larger than the distance for the word "listen". The reason is that "her lips were parted" is an exact phrase that is extracted from A. Doyle whereas "listen" is a common verb for both authors. The same comparison can also be done by considering the Google's pretrained Word2Vec. The cosine distance for "listen" between A. Doyle and pretrained set is found as 0.015 whereas for E. Wharton, it is 0.012. As for "her lips were parted", the cosine difference for A. Doyle and pretrained set is -0.009, and for E. Wharton, it is 0.012. This implies that E. Wharton uses "listen" close to the pretrained set which was trained on large corpus of text data. As for sentence comparison, the sentence average vector is closer to A. Doyle stating that this sentence is more likely to be written by A. Doyle then E. Wharton. The way to achieve this comparison criteria has been provided here for readers to move forward with this methodology[19].

[19] https://github.com/agungor2/Authorship_Attribution/blob/master/word2vec_cos_distance.py

5. CLASSIFICATION METHODS IN AUTHORSHIP ATTRIBUTION

In authorship attribution problems, there is a set of candidate authors and a set of text samples in the training set covering some of the works of the authors. In the test dataset, there are sample of texts and each of them needs to be attributed to a candidate author. Considering 50-author dataset, we make the task of attributing the author for a given test text sample by placing only 45 authors in the training set, and by distributing the different books across training and testing set. However, for the simplicity purposes 3-author dataset does not have unknown author in the testing set.

In our approaches, we will distinguish the authorship attribution techniques based on treating each training text individually or cumulatively (per author). By treating every text pieces individually we mean by extracting features by not considering the other available text samples in the training. In author-wise or profile based approaches, we will keep all available texts per author in one file and try to extract relevant features.

In order to compare results, one can follow different measures as shown in the equation 5.1. In order to test out our training model, we can either create a validation set and compare the performance of each model or since we have the correct labels for both 3 and 50-author test dataset, we can simply try the model on the test data.

There will be a difference in both cases. That is why, we will only consider the performance of the model in the test data with F_1 score.

$$\begin{aligned} Logloss &= -\frac{1}{N}\sum_{i=1}^{N}[y_i \log\, p_i + (1-y_i)\log\,(1-p_i)] \\ F_1 &= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \\ Mean_{Accuracy} &= \frac{\sum_{i=1}^{N} \text{Accuracy for each class}}{\text{Total Class Number}} \end{aligned} \tag{5.1}$$

5.1 Working on Dataset Without Stop Words

At the early stage of our work, we have considered taking out all the stop words from the raw text data and keep the order of the rest of words. By doing so, we have eliminated %76 of the whole word corpus and the size of training set has become 17153 and as for test data it has 12728 instances. Each text pieces again consist of 1000 words.

Feature Extraction

There are several features that are deployed to analyze their usefulness for this dataset. In this application not only the content is important but also stylometry and other features are useful. The features that are extracted to perform classification are as follows:

- Vocabulary diversity
- Bag of words
- N-gram models: 2,3,4 gram models are built. However, it's found that for 3-gram model there are nearly 3500 texts and for the 4-gram model there are 700 texts that contain these features. When introduced three grams and four grams into our SVM model, they are observed to serve as a noise in our model.

- Word2Vec: Build Word2Vec author wise (profile-based) or use pretrained Word2Vec of Google directly.

- Paragrapvec: Use 8 as the window baseline and divide the text into 8 different segments. Average the Word2Vec of each word with their Tf-Idf score multiplication in the 1000 words text fragment.

- Nonoccurence words list: We can define a set of union features that contains the union of all nonoccurence words for each author.

While running Word2Vec one observation is that there are some words that can't be found at Google's pretrained Word2Vec model. Some of these are "theatre", or "centre". The reason is because of the minor difference of writings between American and British English. This also gave me an idea about the existence of English and American authors in the dataset and how I can make use of these features.

To compare the usage of every unique words for each author we created a window size of 10 and evaluated the weights by adding $\frac{1}{|n|+1}$ where n ranges from -4 to 4. Next, we added 1 to word itself. With this method, we can create a similarity matrix for every word and compare their usage in different authors. It can be useful while building one versus all classifier to identify unknown authors in our list.

Building Classifiers

Various classifiers can be applied to our specific task. The followings are used for comparison reason to find a best fitting classifier for our dataset.

- Naïve Bayes Classifier: It's the first classifier that was applied to our dataset. Due to better performance with SVM, we didn't continue to further explore this classifier. However, a better performance comparing to what we have tried could be achieved by defining prior probabilities as the class percentage for each author in the training set.

- Support Vector Machines: They are always advised to be useful in text classification and authorship attribution problems [1], [6], [7], [8], [9]. One very important characteristic of SVM is the training datasets normalization techniques before feeding it into SVM model. The boundaries of SVM are strictly affected by the feature dataset. After finding the proper normalization technique (min-max normalization) we also need to tune "C" parameters which would also fit best to our feature sets.

- Ensemble Methods: RUSBoost, random forest, Adaboost are used to build our classifiers. In order to decrease computation time, PCA and data whitening technique is applied but due to low cross validation scores we didn't continue on building our analyses based on them. The reason why we choose RUSBoost is that it's suggested to perform better for imbalanced dataset.

- K-means clustering is used to find out the nearest words by taking their cosine differences. It's also made useful in unknown author classification task.

- Image Classification: We also think of paragraph vectors or averaged sum of Tf-Idf multiplied Word2Vec as pixels of an image and try to apply spatial descriptor which makes use of estimation and spectral and coarsely localized relationship of pixels [36].

- Convolutional Neural Networks: Using Google's pretrained Word2Vec with one convolution layer Kim Yoon's CNN-nonstatic model is modified to multi class classification problem and CV score of a 0.49 is achieved with considering the book id distribution in classes. It's not a bad result comparing to other classifiers. However, we still proceed with SVM classifiers due to their faster and higher performance results [24].

Experimentation

We have identified two significant problems for our dataset. One of them is the imbalanced data and another one is the unknown authors in our test set. In order to address imbalanced class distribution, we have followed two different approaches. Firstly, the training sets are provided with book ids that are identical for each text and the order of these texts are not being modified. This means that we can combine the texts that share the same book id and reproduce training texts that are from these books again. Alternatively, we can randomly increase the number of minority classes by using randsample of Matlab or using SMOTE which is observed not to perform well on this dataset. To identify the weight of each class in overall f1 score performance we have dropped out each class one by one and used 5 folds cross validation to capture mean F1 scores with bag of words representation. Maximum captured mean F1 score is 0.81 with Albert Ross's work and the minimum one is 0.75 for Lucas Malet. Second problem is to identify right cross validation technique [1]. One way to solve is that while taking the instances for cross validation we can take the same book id as a reference for the books of all authors which would make it closer to final test score.

- Stylometric Features with SVM: Bag of words representation, diversity of a text which is defined as the unique number of words over total number of words, 2,3,4 grams, British & American writing differences of some specific words are extracted. To find out best classifier we've used only bag of words representation with the top 3000 most occurred words. Naive Bayes, Random Forest and Ensemble techniques perform 0.56, 0.52 and 0.49 with unknown book id cross validation technique. SVM outperforms them all with a score of 0.61.

  To further improve the performance of SVM we have tried different normalization technique as log of nonzero elements, row and column wise normalization with different number of bag of word selection. The best normalization technique is min max normalization for our dataset. When taking min-max normal-

[1] https://github.com/agungor2/Authorship_Attribution/blob/master/check_oversampling.m

ization we have considered training and testing together. After normalization, we split them again. We also have chosen the words that are labeled from 7000 to 10000 and compared with the all bag of words representation feature set. All words representation feature set has proven a better result with 0.84 on known book id cross validation calculation whilst the other one performs 0.78.

Table 5.1.: Mean F_1 Scores for Known and Unknown Book Id

Features	**Known**	**Unknown**
Bow 3000	0.78	0.68
Bow all	0.84	0.71
Bow all, & bigrams	0.87	0.73
Bow all, 2&3 grams	0.86	0.72
Bow all, &2,3,4 grams	0.83	0.69
All useful features	0.87	0.73

As summarized in Table 5.1 the best scores are achieved with concatenating all bag of words representation, bigrams, 74 different used words between English and American author, and vocabulary diversity measure. During the experimentation C parameters are also tuned and the best score is achieved with C=3. During the experimentations, the cross-validation indexes are kept same for each experiment since different configuration of cross validation value may vary minor differences. Since these results are cross validation results, F_1 scores on the test data will be closer to unknown book id settings but it will be lower than them.

- **Word2Vec:** Using pretrained words, taking their averages for all 1000 words in a text segment, and averaging with Tf-Idf for each text gives a cross validation

score of 0.57 with known book ids. The same experiment is also done with Tf-Idf divided feature set and performed 0.54. Using a window size of 8 with the suggestion of Mikolov's work [25], we have defined paragraph vectors of 125 words of text piece. However, we didn't further proceed with this idea due to low results in the first Word2Vec approaches.

Even though Word2Vec or CNN's are well defined techniques in semantic feature selection they are not as useful as other simple techniques such as bag of words or n-grams in our dataset that is removed stop words. The reason is that their fundamental working principle bases on sequential order of words in each text whereas in our dataset we have lost this information. However, dealing with them have helped us found out about the existence of American and British authors in our dataset. Simple bag of words with n-grams and vocabulary diversity is good features for this classification problem. The way to improve this technique is done by trying out different normalization as SVM boundaries depend on them. The increase was observed with 0.05 on mean F_1 score with just pursuing a different normalization technique which wasn't observed even with Word2Vec or CNN.

5.2 Feature Engineering with Different Classifiers

In order to work on the semantic and syntactic features of text we have put back the stop words in the dataset. In this setting, again there are 1000 word pieces of texts and the total number of unique words in the whole corpus is set to be 10,000. Total number of training set 53678 whilst total number of test set is 38809. In the training set, there are 45 authors and in the test set there are 50 authors. G. Eliot, J. London, F. H. Burnett, S. Ellis, T. Page are the missing authors in the training set which consists of %34 of all testing data.

Table 5.2.: Mean F_1 Scores for Different Experimentation Settings

Dataset	**Features**	**Classifier**	F_1 **Score**
3-author	Tf-Idf train, test separate	Logistic R.	0.806
3-author	Tf-Idf train, test together	Logistic R.	0.808
3-author	Countvectorizer train, test separate	Logistic R.	0.801
3-author	Countvectorizer train, test together	Logistic R.	0.801
50-author	Tf-Idf train, test separate	Logistic R.	0.46
50-author	Tf-Idf train, test together	Logistic R.	0.47
50-author	Tf-Idf train, test together svd 120	SVM	0.52
50-author	Tf-Idf train, test together svd 120	Xgboost	0.496
50-author	Tf-Idf train, test together scaled svd 120	Xgboost	0.495
50-author	BOW words 9501:10000	SVM	0.5416
50-author	BOW words 7000:10000	SVM	0.5958
50-author	BOW words 5000:10000	SVM	0.6110
50-author	BOW words 3000:10000	SVM	0.6172
50-author	BOW words 7000:10000	Xgboost	0.516
50-author	combined features only	SVM	0.017
50-author	Combined, BOW words 7000:10000	SVM	0.6104
50-author	Combined, BOW words 3000:10000	SVM	0.6361
50-author	BOW words 1:10000	SVM	0.6410
50-author	Combined, BOW words 1:10000	SVM	0.6425

In most of NLP tasks, Tf-Idf and bag of words representations are the first features to implement into a classifier of choice. To test out the performance of these two modules, we have implemented on 3-author dataset using Tf-Idf vectorizer and Count vectorizer. In both of these two settings, we consider the number of unique words,

bi-grams and tree grams. To test out their performance we can do two adjustments. Before feeding the text into Tf-Idf or Count vectorizer, it is possible to combine all training and test set together to feed in or get these vectors by separately considering training and test texts. To take the measure of F_1 scores for both of these two settings, we have implemented it on Logistic Regression classifier by keeping the C=1. The result for Tf-Idf vectors which are considered separately for the training and test set is 0.806 whilst when considered together it is 0.808. When we tested the same experiment on Count vectors, F_1 scores are 0.801 in both settings.

We observed not much of a difference for using Tf-Idf or Count vectors as a feature, so we picked one of them, Tf-Idf, and implemented for 50-author dataset using Logistic Regression Classifier. We had two settings as before when building Tf-Idf vectors. The advantage of combining training and testing set for feature selection is that it serves as a semi-supervised learning. We saw better performance for semi-supervised learning case with 0.47 mean F_1 score comparing to separated training and testing set. By considering the performance of semi-supervised learning setting, we have applied it on SVM and Xgboost Classifiers as well. Since the dimension of Tf-Idf vectorizer is huge and it is sparse matrix, we have implemented Singular value decomposition (svd) and selected 120 components for both training and testing set. From our experimentation we record that a good range of svd component choosing for this problem is between 120 to 200. For SVM we choose C=1 and for Xgboost max depth is 7, number of estimators are 200, and learning rate is chosen as 0.1. With svd 120 components, SVM and Xgboost performed 0.52 and 0.496 respectively. We scaled and standardized 120 svd features by removing the mean and scaling to unit variance. The scaled dataset has then been implemented to Xgboost with same settings and has not been observed a performance improvement. We chose 120 components from svd but it could go up to 200 components for better results. The models that have been implemented so far are provided here [2].

[2] https://github.com/agungor2/Authorship_Attribution/blob/master/feature_engineering.py

Comparing the result of SVM and Xgboost classifier, Xgboost did not perform good enough. There is a way to increase the performance by hyper-parameter tuning or so called Grid search. The parameters we choose can be optimized by testing out different settings one by one with each other and creating a stop condition to see if there is a change up to 50 rounds of iteration. Hashing vectorizer can also be used instead of count vectorizer or Tf-Idf vectorizer. The difference is that with hashing vectorizer it uses the hashing trick to find the token string name to feature integer index mapping. It is very low memory scalable to large datasets as there is no need to store a vocabulary dictionary in memory. We can also choose the number of components we want to store with this technique which allows smaller size of sparse matrix. Another major task to improve the score is to create probability of the text per author and feed this as a feature to classifier of choice.

As mentioned, using count vectorizer is fast for 3-author dataset but is challenging for 50-author dataset due to size of 50-author dataset. In order to implement feature extraction techniques simply, we have built the dataset by representing each word with a unique number and storing it in array. To apply the bag of words representation a simple model is built here [3]. In this model we can select the word of our choice as a feature and measure the frequency of these words per text. Since the boundaries of SVM heavily relies on the feature data, we have also implemented min-max normalization before feeding the data into SVM. Even by only checking the most occurring 500 words, SVM performs a lot more than previous settings with a score of 0.5416. We see a performance improvement as we increase the number of features to 3000, 5000, and 7000 and the measured scores follow as 0.5958, 0.6110, and 0.6172 respectively.

We also checked the performance of top 3000 frequencies with Xgboost by considering the same parameters as in previous cases. We measured 0.516 F_1 score and Figure 5.1 shows the importance of the top 20 words on the F_1 score measure. Us-

[3] https://github.com/agungor2/Authorship_Attribution/blob/master/stylometric_feature_svm.html

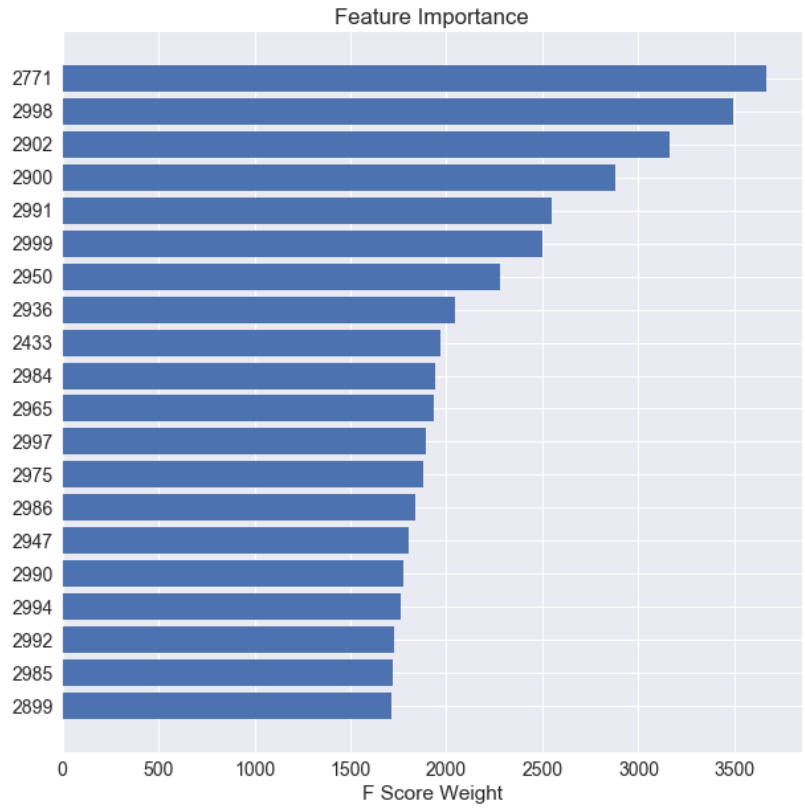

Figure 5.1.: Xgboost Feature Distribution

ing the word ids provided here one can decrease the number of features and retrain the Xgboost. Another comparison was also done by choosing the top 500 important words and implementing it in both of Xgboost and SVM. After experimentation, we have not observed any difference.

To increase the F_1 score we can understand that we can not keep increasing the number of bag of words feature as it contributes less when increased. One way to address this issue is to go through the model weight of each word features and take out the ones that has less weight. Another way is to introduce more features as a distinguisher. We have listed out several lexical, syntactic, and character level difference. In order to implement some of them to 50-author dataset a model has

been built [4]. These features are number of unique words, characters, stop words, adjectives, nouns, verbs, and polarity & subjectivity measures (sentiment scoring). We have implemented only these features to see the affect in overall mean F_1 score with SVM. It gives a score of 0.017. When implemented these feature with top 3000 bag of words we see an increase in the score from 0.5958 to 0.6104. As you can see the difference between the results are almost same as unique contribution of the stylometric features. We have also implemented it with top 7000 bag of words representation and have recorded better measure of 0.6361. Figure 5.2 shows the confusion matrix of this experiment and we can see the misclassified areas near by the authors G. Eliot, J. London, F. H. Burnett, S. Ellis, T. Page because they are the missing classes in the test data. We also observe some misclassification for R. Kipling and A. Manning. There are 13173 texts misclassified that are written by G. Eliot, J. London, F. H. Burnett, S. Ellis, T. Page in the test data. 23464 are labeled to true classes and 2172 are misclassified that are member of 45 authors. The misclassified cases that are part of 45 authors consist of %5.6 of the test data. This is the room to improve the performance of our approach for further studies. It could be possible by hyper parameter tuning with Grid search approach and adding additional discriminative features. In the final stage, we also have implemented using the frequency of all words in data set and we have achieved 0.6410 F_1 score. All bag of words features are also combined with the other useful features and have recorded a performance of 0.6425.

We have not implemented syntactic n-grams (sn-grams) that are defined in this paper [37]. Sn-grams are combination of Part of Speech tagging and n-grams in which the advantage of sn-grams is that they are based on syntactic relations of words and, thus, each word is bound to its real neighbors. This allows ignoring the arbitrariness that is introduced by the surface structure [37]. Sn-grams are expected to bring

[4] https://github.com/agungor2/Authorship_Attribution/blob/master/feature_engineering4.py

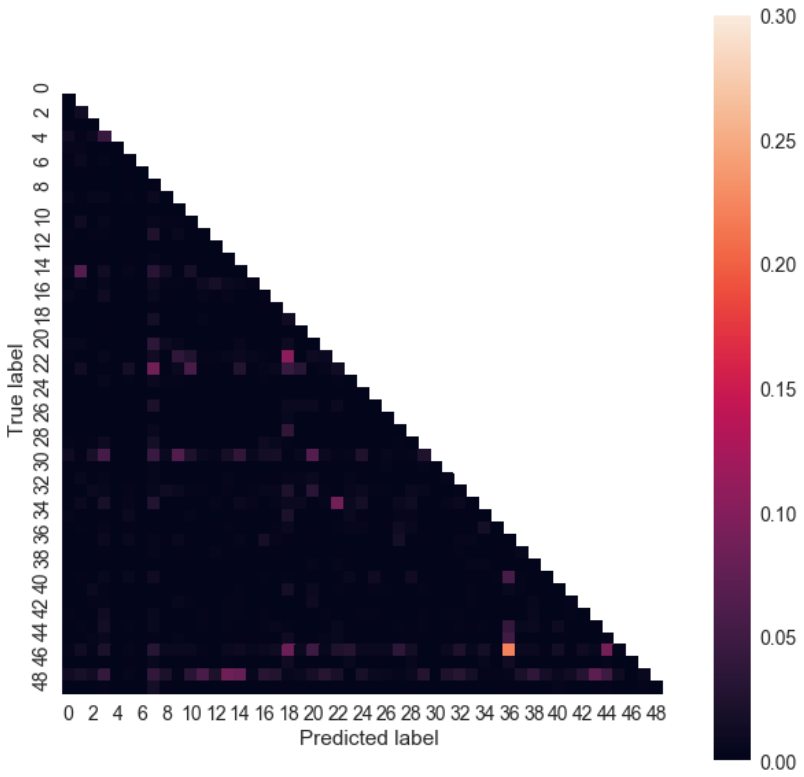

Figure 5.2.: Confusion Matrix for SVM Combined Features

grammatical perspective of the text that could serve as a good discriminative feature set.

5.3 Sentence and Paragraph Generating Model

The methods we have introduced so far have been made use of frequency of words, characters and other stylometric distinguisher as a feature to stack them together and implement with different classifiers. Another possible usage of such information can

be to create character or sentence based language models using very simple conditional probability distributions.

$$\begin{aligned} P(Word) &= P(c_1, c_2, c_3, ..., c_n) \\ P(c_1, c_2, c_3, ..., c_n) &= \prod_{i=1}^{n} P(c_i) \\ P(Paragraph) &= P(w_1, w_2, w_3, ..., w_n) \\ P(w_1, w_2, w_3, ..., w_n) &= \prod_{i=1}^{n} P(w_i) \end{aligned} \tag{5.2}$$

The probability of a word can be simply written as the multiplication of all characters probability in the word and same description can be defined for a sentence or a paragraph, too. As for paragraph or sentence we can also think of combination of words and their probability can also be redefined.

$$\begin{aligned} P(word) &= P(c_1, c_2, c_3, ..., c_n) \\ P(word|Author) &= \frac{P(Author|word) * P(Author)}{P(word)} \\ P(Paragraph) &= P(w_1, w_2, w_3, ..., w_n) \\ P(Paragraph|Author) &= \frac{P(Author|Paragraph) * P(Author)}{P(Paragraph)} \end{aligned} \tag{5.3}$$

We can then simply define the probability of a word being written by one of the authors as the conditional probability. The same settings can be applied for paragraph level creation considering from the word base. Here we can think of $P(Author|word)$ as the normalized frequency of each word in training set and $P(Author|Paragraph)$ can be considered as the normalized distribution of the text pieces per author in the training set.

$$\begin{aligned} \textbf{Predicted Author} &= argmax\ \{P(c_1, c_2, c_3, ..., c_n|Author)P(Author)\} \\ \textbf{Predicted Author} &= argmax\ \{P(w_1, w_2, w_3, ..., w_n|Author)P(Author)\} \end{aligned} \tag{5.4}$$

We can then simply choose the author by picking the maximum probability for each word or character cases as shown in the equation 5.4. It is also possible to define

a Markov Chain model by storing the probabilities of characters or words in different Markov Chain steps as described in equation 5.5 and 5.6 respectively.

$$\begin{aligned} P(c_1, c_2, c_3, ..., c_n) &= P(c_1) \prod_{i=2}^{n} P(c_i | c_{i-1}, c_{i-2}) \\ P(c_1, c_2, c_3, ..., c_n) &= P(c_1) P(c_2|c_1) \prod_{i=3}^{n} P(c_i | c_{i-1}, c_{i-2}) \end{aligned} \tag{5.5}$$

$$\begin{aligned} P(w_1, w_2, w_3, ..., w_n) &= P(w_1) \prod_{i=2}^{n} P(w_i | w_{i-1}, w_{i-2}) \\ P(w_1, w_2, w_3, ..., w_n) &= P(w_1) P(w_2|w_1) \prod_{i=3}^{n} P(w_i | w_{i-1}, w_{i-2}) \end{aligned} \tag{5.6}$$

In equation 5.5 and 5.6 steps we need to store the probability so that when defining conditional probability we can use them again. However, when considering the size of the corpus the character level Markov Model is computationally less expensive to implement comparing to word level. Hence it is the reason why, for our computations we have not considered storing prior probabilities. For the simplicity, we will consider 4 settings. In the first two settings, we have defined a text piece as a combination of words and the conditional probability of that text piece being written by one of the authors is defined as in equation 5.3. In order to consider to the probability of authors we have taken two options. First one is to take them as equal, and second one is to consider them as their normalized text pieces number in the training set [5]. In the second setting, we have considered the 1000 words text piece as a combination of characters and calculated their conditional probabilities [6].

Figure 5.3 shows the difference of each characters in the entire training corpus for Arthur Doyle, Charles Dickens, and James Baldwin. We can see the dissimilarities among them for different characters. The same steps can also be done when considering the usage of punctuation to extract more features.

[5] https://github.com/agungor2/Authorship_Attribution/blob/master/word_freq_guess.py

[6] https://github.com/agungor2/Authorship_Attribution/blob/master/Character_level_distribution.py

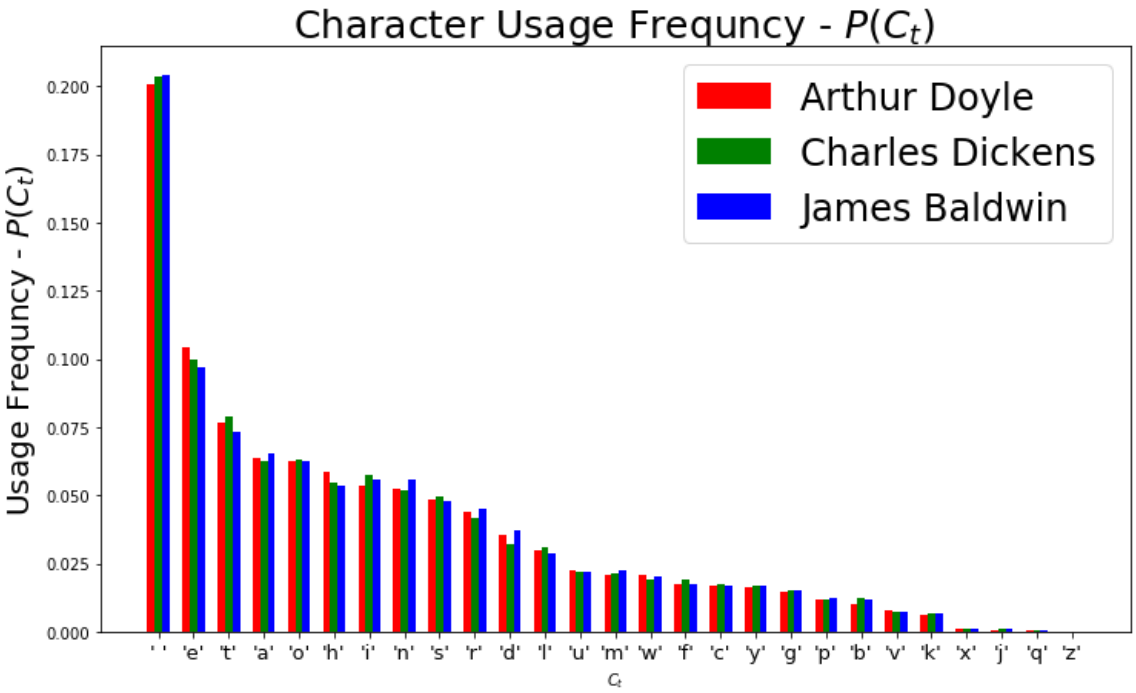

Figure 5.3.: Character Frequency for A. Doyle, C. Dickens, J. Baldwin

Another method using the Character distribution knowledge is to create a character level Bag of characters considering the 27 unique characters in the dataset and their bi-grams, tri-grams, and four-grams to feed it into any classifier as we have shown in the previous experiments. We have not implemented this methodology but interested readers could pursue this part or deploying the next steps of Markov Chain Model as described before.

When doing the experimentation of (a), (b), (c), (d) we have firstly calculated the character level and word level distribution per author and then used this knowledge to calculate the conditional probability per author for each text. The scored measures are comparably a lot lower than simple bag of words approach. The reason why the measured scores low is because of large number of words in a given text piece. However, intended goal with this methodology is to lay the foundation of the model and help researchers implement such models. In order to improve scores, Markov Model steps are needed to get correlated relationship between characters and words.

Another method to improve scores is to use Markov Models and the conditional probabilities we use as a feature to train classifiers with other useful features to follow the steps of feature stacking methodologies.

(a) In the first experiment, we have considered the character level distribution to create the whole text pieces and the prior probability of each author is taken equal.

(b) In the second one, we have considered the same setting as in (a) but changed the prior probabilities with author normalized distribution in the training set.

(c) Instead of considering characters, we took the word distribution as the probability per author and experimented when considering same probability for each author.

(d) In the last experiment, we look at the performance of (c) when considered the normalized distribution of authors in the training set as prior probabilities.

5.4 Ensemble Methods

As shown in Table 5.2, every model has a different performance on the same set of feature. We have introduced grid search strategy to improve the overall performance for each classifier. We have also introduced feature engineering strategies in which adding and concatenating extra discriminative features could help build better classifiers (feature ensemble). In the feature ensemble models, we have not introduced weight per feature which could also be a way to improve the performance.

There is also an aspect of ensemble methods in which multiple different models can be built and calibrated to get better performance. By using multinomial Naive Bayes, SVM, Multivariate Bernoulli model, and Logistic Regression, an ensemble example case is built [7]. We have also used classifier calibration which makes use of

[7] https://github.com/agungor2/Authorship_Attribution/blob/master/ensembling_methods.py

a cross-validation generator and estimates for each split the model parameter on the train samples and the calibration of the test samples. The probabilities predicted for the folds are then averaged. In this setting, we can also define gains per model to try out different experimentation. The experimentation of 50-author dataset has decreased on Tf-Idf feature set to 0.432. However, it is expected to improve score when choosing with the well fitted weights and best parameters for each classifiers.The same methodology can be implemented on the ensembled feature set to see the performance change in further studies.

5.5 Defining Sentence Vectors

We have introduced five possible usage cases of word embeddings in different NLP tasks. Despite the vast implementation growth in word embeddings, it is still a research topic on how these word vectors can be used to represent a sentence or a paragraph [38]. For constructing sentence embeddings, naively using averaged word vectors was recently shown to outperform LSTMs [39], and using CBOW Word2Vec training objective to train sentence instead of word embeddings have also recently being studied [38]. In our approaches, we will introduce three ways to represent a sentence vector.

Paragraph Vector

The idea of paragraph vectors are firstly introduced by Lee and Mikolov [40] as a continuation of their model of CBOW on Word2Vec. The idea is straightforward: act as if a paragraph (or document) is just another vector like a word vector, but we will call it a paragraph vector. This idea has then been implemented with gensim and sent2vec that is built on top of fasttext [38].

When it comes to applying this methodology to 50-author dataset, due to the size of our dataset it is computational expensive unless the C library implementation of Word2Vec has been built and CPU core computation has been allowed. In order to

create paragraph vectors we can train the training and testing part of our dataset separately or train them together. Since every text piece has 1000 words defining paragraph can be possible by dividing the text into small paragraph pieces. For simplicity in our implementation we have defined very 1000 text piece as a paragraph and trained testing and training separately [8]. During the training process of paragraph vectors it is also possible to manually set the learning rate at each iteration and train a better model.

Average of Word Embeddings

A sentence vector can also be simply defined as the average of all word vectors in the sentence. There are also two methods to define a word vector here. One way is training a Word2Vec model on every 1000 text pieces and recording the word vectors, then taking the average. This method is a bit controversial since the order of the words in different sentences vary and it is questionable that the trained word vectors are good representation of them in vector space. This problem could be addressed by feeding the pretrained word vectors of Google's or Glove's as an embedding to the neural network. Second method is simply using a pretrained Word2Vec model to extract each word vectors and use them to define a sentence vector. In our approach we deployed Glove's 300 dimensional pretrained model and scaled the Word2Vec when defining the sentence vectors by dividing them with the squared root sum of all vectors square [9].

Average of Word2Vec with Tf-Idf

The main difference with the previous case is that we have deployed every Tf-Idf words measure as a weight before defining the paragraph vectors. Each Word2Vec is multiplied with their Tf-Idf scores and then scaled the sum of all word vectors. In this

[8] https://github.com/agungor2/Authorship_Attribution/blob/master/doc2vec_example.py
[9] https://github.com/agungor2/Authorship_Attribution/blob/master/word2vec_xgb.py

approach it gives more gains to the words that are more meaningful in the sentence vector calculation.

Table 5.3.: Mean F_1 Scores for Sentence Vectors

Dataset	**Setting**	**Classifier**	F_1 **Score**
50-author	Doc2Vec	Logistic R.	0.085
50-author	Doc2Vec	Simple XGB	0.140
50-author	Doc2Vec	XGB Paramter Tuning	0.150
50-author	Doc2Vec	SVM	0.090
50-author	Average Word2Vec	Logistic R.	0.284
50-author	Average Word2Vec	Simple XGB	0.367
50-author	Average Word2Vec	XGB Paramter Tuning	0.434
50-author	Average Word2Vec	SVM	0.371
50-author	Average Word2Vec with Tf-Idf	Logistic R.	0.302
50-author	Average Word2Vec with Tf-Idf	Simple XGB	0.384
50-author	Average Word2Vec with Tf-Idf	XGB Parameter Tuning	0.445
50-author	Average Word2Vec with Tf-Idf	SVM	0.397

During the experimentation of defined sentence vectors, we have used 300 dimensional representation of Word2Vec to represent a feature of a given text. The experimented results are summarized in Table 5.3. Doc2vec has not performed well despite of the different adjustments. One of the reasons is that every text piece has 1000 words which makes it a challenging job to represent as a vector. One way to address this issue could be by splitting every 1000 text piece by 10 fragments and each fragment is a combination of 100 words which could be a good number to represent a paragraph. It is possible to take the scaled average of 10 fragments to represent for each vector. Average Word2Vec has performed better when representing a text piece.

We can also see a performance boot with parameter tuning for Xgboost from 0.367 to 0.434. By introducing Tf-Idf gains when defining a sentence vector outperformed the other approaches and it increased mean F_1 scores slightly for each case. The way to improve the scores for defining a sentence vector in average Word2Vec approach could again be done by dividing the text pieces into small text fragments. Each fragment can then be concatenated to create a feature set. Another approach to check is to try out point-wise multiplication or convolution technique with Word2Vec to see if it improves the overall scoring when defining a paragraph vector.

5.6 Specific Word Usage Score

The common use case of word vectors have been so far aimed at defining a sentence vector from the pre-trained or newly trained word vectors. In this approach, we would like to see the usage of some specific words for every author. As described before, authors prefer to use some specific words such as power words, stop words, flagging sentiment words like surprise, fear, jovial feelings at a different frequency. By considering these unique word usage we aim at defining a specific word score. It is also possible to select random words to analyze. However, for experimentation we have used power words and stop words.

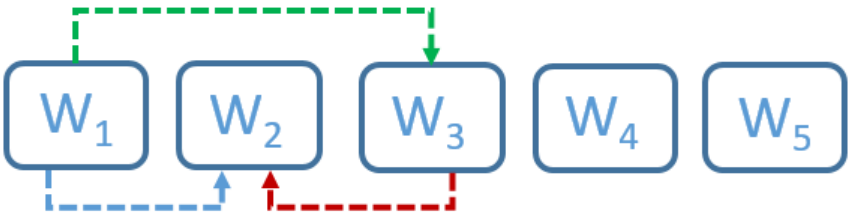

Figure 5.4.: Word Scoring Model

$$W_{2\text{ score}} = \frac{|d(w_1, w_3)|}{|d(w_1, w_2)| + |d(w_2, w_3)|} \tag{5.7}$$

Let's take a sentence of five words shown in Figure 5.4 and assume w_2 is the word we are trying to give a score measure. We have defined the scoring for window of 3 words by taking the ratio of the distance of words that are surrounding w_2 with distance of w_2 to w_1 and w_3 as in equation 5.7. It is also possible to extend the word numbers within this range to your choosing. For example, with 5 words window scoring measure can be defined for w_3 as in equation 5.8. The distance measure between vectors here is cosine distance.

$$W_{3\text{ score}} = \frac{|d(w_1, w_3)| + |d(w_3, w_5)|}{|d(w_1, w_2)| + |d(w_2, w_3)| + |d(w_3, w_4)| + |d(w_4, w_5)|} \tag{5.8}$$

After defining the word score, we can check distribution of this scoring across all the text documents in the training set for every author. The t test would tell us how significant the differences are in these distributions. T test can be done by considering two settings: one vs one, and one vs all fashion. In one vs one setting, we can compare the t test result by taking one author from the training set and comparing it with the rest of the authors one by one. In the second method, we can compare each author with the remaining all author's distribution. In each t test we record the p values for every experimentation. In order to see if this approach is promising we need to do t test to see how authors use these words different than the others. Figure 5.5 shows the p values that are smaller than 0.05 for top 20 words. The words that are shown in white squares are used differently comparing the other authors. These t test have been implemented on the most common words from word id 9981 to 10000 choosing 20 of them, but one can choose to it on power words or flagging words to see the usage difference [10].

Now that we know there is a difference of usage in choosing some of the top common words for each author, we also would like to apply Bhattacharyya distance to define the author usage. Bhattacharyya distance calculation is provided in equation 5.9. In the equation, p, q are two different distributions, μ_p is the mean of the p-th distribution, and σ_p^2 is the variance of the p-th distribution.

[10] https://github.com/agungor2/Authorship_Attribution/blob/master/ttest_one_all.m

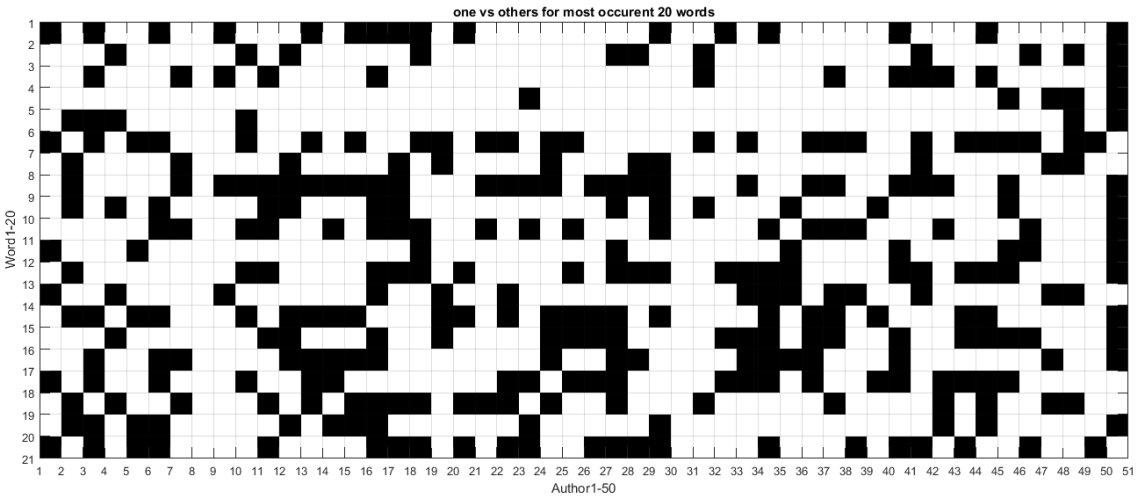

Figure 5.5.: One vs Others Most Common 20 Words

$$D_B(p,q) = \frac{1}{4}\ln\left(\frac{1}{4}\left(\frac{\sigma_p^2}{\sigma_q^2} + \frac{\sigma_q^2}{\sigma_p^2} + 2\right)\right) + \frac{1}{4}\left(\frac{(\mu_p - \mu_q)^2}{\sigma_p^2 + \sigma_q^2}\right) \tag{5.9}$$

The reason why we do these two tests is to identify the different word choosing of authors and how these authors have used them. We choose the words in t test that have p values less than 0.05 or for Bhattacharyya distance we take the indexes that are greater than the mean value of the Bhattacharyya distance distribution for authors.

After identifying the words of choosing we calculate the mean and variance of each words for every author in the training set. When identifying the author for a text piece in the test data, we find the distribution of word scorings measure for the words we identify after analyzing with Bhattacharyya distance and t test measure. In the final stage, we calculate the joint likelihood for every author given the calculated mean and variance values for every word in the training set. The maximum of the calculated joint likelihood is being labeled as the author in the testing set. The testing algorithm for this whole process has been implemented here [11]. Pseudo code

[11] https://github.com/agungor2/Authorship_Attribution/blob/master/Bhat_5word_distance.m

of algorithm steps are also provided below for future developments. In order to test

Algorithm 1 Word Scoring Joint Likelihood Algorithm (WSJL)

```
1: procedure WSJL PART1(words, authors)
2:     for range(length(words)) do
3:         for range(length(authors)) do
4:             Calculate word scoring
5:         Bhattacharyya distance                          ▷ Or do t test
6:     return Bhattacharyya distance
7: Choose words that are worth investigating
8: procedure WSJL PART2(worthy words)                      ▷ In training set
9:     for range(length(authors)) do
10:        for range(length(worthy words)) do
11:            Calculate word scoring
12:        Calculate mean and variance
13:    return mean and variance
14: procedure WSJL PART3(mean and variance)                ▷ In testing set
15:    for range(length(testing set)) do
16:        for range(length(authors)) do
17:            for range(length(worthy words)) do
18:                Calculate word scoring
19:            Calculate likelihood
20:        Label authors with index of maximum joint likelihood
```

out the performance of this algorithm 50 author dataset has been modified and only 5 author are chosen for this task. These authors are W. Irving, F. H. Burnett, J. Abbott, J. Payn, O. Optic. The reason why these authors are chosen is because of their number of text pieces close to each other. In the testing set, there was no class imbalance or unknown authors.

Table 5.4.: Mean F_1 Scores for WSJL Algorithm

id	Text Number	100 words	200 words	1000 words
J. Payn	3694	0.34	0.28	0.41
o. Optic	3504	0.15	0.28	0.26
F. H. Burnett	3487	0.12	0.10	0.17
W. Irving	3455	0.27	0.26	0.21
J. Abbott	3332	0.06	0.29	0.37

Table 5.4 summarizes the experimentation on this algorithm performance. In the settings we have tested out for 100, 200, 1000 words. At each setting these words are chosen from the most common word lists. However, one important note is that the number of word decreases when selecting worthy words. For example, in the 100 words setting number of worthy words is 40 and in the 200 words it is 132. After identifying the worthy words among the selected words, we have implemented joint likelihood. We can see the overall F_1 score increase when increased the number of words that we have selected when defining word scores.

5.7 Unsupervised Feature Learning

There are publicly many algorithms available to learn features from the unlabeled dataset. However, training such algorithms can be tricky and difficult for some datasets. It has been found that K-means clustering can be used as a fast alternative training method [41]. The resulting features performs effective for learning large-scale representations of images.

In this approach, windows and strides are defined. Windows represent the number of words chosen for the calculation and strides are the sliding window that next words are chosen in the given text pieces. Then, we combine all training and testing data

Algorithm 2 Unsupervised Feature Learning (UFL)

```
1: procedure UFL PART1(window, stride)
2:     Combine Training and Testing Set
3:     for range(1000/s*length(combined data))-1 do        ▷ Divide text piece
4:         Calculate sent2vec and create new data          ▷ Could use Tf-Idf
5:     return new data
6: Normalize new data
7: Find K cluster center points
8: Normalize cluster center points
9: procedure UFL PART2(cluster center points)    ▷ Both train & test data
10:    for range(length(train or test)) do
11:        Create new data as in part 1 using w and s
12:        for range(length(new data)) do
13:            for range(length(K Cluster)) do
14:                Calculate distance
15:            Set the distance greater than mean to zero
16:        Create feature set from the distance and normalize it
17:    return feature set
18:    Future Set can be normalized again
19:    Apply classifier
```

together. For every 1000 words text piece, the window is chosen to create a sentence vector by averaging the Word2Vec of all words in the window. During the calculation of sentence vectors one can also implement it with using Tf-Idf average. By sliding through stride numbers a new data is being created. For example, for a 1000 words text piece by choosing window as 10 and stride as 5 we create 201 new sentence vectors. This process is done for all 1000 text pieces. Overall number of new data points with window size 10 and stride size 5 then becomes 18813600. After creating the new data, we take the row wise normalization of it. By using the vlfeat K-means clustering algorithm we find the K number of cluster center points. Identified K cluster center points are then used in training and testing data to find the cluster distance to the chosen window and stride points. For each data point, the distances that are greater than the mean distance set to zero. These distances are then used to train classifiers like SVM.

Using Word2Vec and Google pre-trained word vectors we take w=10 (imagine 10 words make a sentence) and s=5 (strides). Then we create new dataset with all the points, run vl-kmeans and find out 1000 points, see them as our topics. Then we check the likelihood of each sentence to be written by these authors.

We have shown the useful representation of sentence vectors from Word2Vec, especially using with Tf-Idf averaging. In this approach, we can think of dividing the whole data points into small sentence pieces and identifying clusters (think of each cluster as a document topic) among these sentence pieces. To implement this algorithm we have used pre-trained Google vectors to calculate sentence vectors. In order to see the performance of this algorithm, text pieces are chosen from the works of W. Irving, F. H. Burnett, J. Abbott, J. Payn, O. Optic and there is no unknown author in the test data. In this setting, our implementation of Unsupervised feature learning has performed %92 accuracy by choosing window size as 8 and stride size as 4. When we also implement window size as 10, stride size as 5, and concatenating these two features have performed %97 accuracy. In the same experimentation setting, the

bag of words accuracy has been recorded as %99 [12]. One of the main reasons why bag of words and unsupervised feature learning have performed well on the test data is because of the test and train data split. As noted before, when splitting train and test data if the book ids are not uniquely distributed then the classification task becomes an easy job.

Table 5.5.: UFL F_1 Score on 50-Author Dataset

Setting	**K=100**	**K=1000**
W=10, s=5	0.16	0.37
W=20, s=10	0.14	0.31
W=50, s=25	0.12	0.27
Concatenated	0.23	0.38

To further investigate this approach due to its good performance we have implemented on 50-author dataset with keeping the same settings as 45 authors in the training and 50-author in the testing. In our test set, we have experimented on window size of 10, 20, 50 with 5, 10, 25 stride numbers. After extracting the features, we also have concatenated all distances into one feature set and implemented SVM classifier. Table 5.5 shows the performance of each case scenario. We have concluded that the small window sizes are good at identifying the number of good features. Increasing the number of clusters also increase the overall F_1 score. However, computational time complexity and overall performance of unsupervised feature learning is not as good as bag of words.

[12] https://github.com/agungor2/Authorship_Attribution/blob/master/Unsupervised_feature_learning.m

5.8 Inversion with Word Embeddings

Distributed language models that map words to vector space are rich in information about word choice and composition. Using the Bayes rule, a distributed language model can be turned into a probability model. The application of this approach has been studied on Yelp review dataset to do text classification [42]. The dataset consists of reviews that are rated from 1 to 5 and there are over 2 million sentences. Inversion method has been noted to perform as well as or better than complex purpose-built algorithms [42].

The steps of the algorithm consist of training profile based Word2Vec model for every author in the training set and creating a probability score for text pieces in the testing set per author. Then, by choosing the maximum likelihood we identify the author of the given text. To apply Bayes rule to go from $P(text|authors)$ to $P(authors|text)$ we make use of gensim score model. In the implementation of Word2Vec score model, a binary Huffman tree is employed to calculate each word probability [42]. The measurement steps are provided below.

Algorithm 3 Inversion Word2vec (IW)

```
1: procedure IW
2:     for range(Unique(Authors)) do                ▷ Author wise Train
3:         Calculate Word2Vec on training set
4:         for range(test data) do
5:             Measure P(authors|text)
6:     return P(authors|text)
7: P(authors|text) to 0-1 range
8: Pick the highest likelihood
```

There is one thing about the size of our data that might cause a problem when calculating $P(authors|text)$. Since each text piece consists of 1000 words, overall probability of an author having to write that sentence would go down drastically.

One way to address this problem is again by using window and stride methodology. By defining different length of window and strides we can obtain probability for each window and by summing log of all the probabilities in the divided text piece we can redefine a probability measure for all authors.It is also possible to train document vectors for every author and get a probability score for every author, too [40].

We have implemented this methodology on 50-author dataset [13]. In the experimentation, we have defined window=1000, stride=500, and window=500, stride=250. In the first experimentation measured F_1 score is recorded as 0.13 and in the second is 0.12 respectively. The simplicity of the approach makes it easier to apply to our dataset but the scale of our text piece is big enough to introduce noises when doing experimentation hence the reason why it performs low.

Another simple way to test out document vectors for identifying the author of a text in the testing data can be done by employing the document vectors that are being trained in 5.5 and creating cluster center points by simply choosing the average of paragraph vectors in the training data per author and looking up the cosine distance of each instance in the testing set to every author. In the final stage, we can label the text piece to the closest author. This simple approach gives a good starting accuracy measure on 50-author dataset based on our experimentation. In order to further improve this methodology, one way is to create multiple center points in every authors and do the same distance measure among all authors.

[13] https://github.com/agungor2/Authorship_Attribution/blob/master/inversion_word2vec.py

6. SUMMARY

From the traditional NLP methods to sophisticated ones using Word2Vec in authorship attribution problems have been introduced. In the first chapter, an extended literature review with previously studied methodologies are introduced. In the second chapter, to create a knowledge base and inspire new ideas to rise, NLP tasks with available NLP toolboxes are introduced by providing multiple examples. These NLP tasks are crucial steps to start analyzing a text document. In the third chapter, a detailed information regarding the used datasets is given. In the next chapter, feature extractions strategies that are categorized as lexical, character, syntactic, semantic and application specific have been explicitly implemented on both of the datasets and the source code is available for future development. The fifth chapter analyzes these features on different classification experiment settings. Some novel approaches using Word2Vec is also introduced to lay the groundwork on this set of problems.

One of the main objects in this work was to create author signature using the common used words with publicly available pretrained word vectors. The several attempts we have tried by word vector averaging, paragraph vector creating, specific word usage score, unsupervised feature learning from word vectors, word2vec inversion have been around this objective. These new techniques in authorship attribution have not outperformed the usage of traditional methods such as bag of words, n-grams, or other author specific features. The strong power of SVM and simple bag of words with stacked features out performed all other experimentations leading the highest score. However, there is still room for further development on the methodologies introduced. Hence it is the reason why, feature extraction methods and analyzing them have been made publicly available to allow new researchers to pick up where we have left off.

The methodologies introduced in this work have also been around the idea of creating the largest available Authorship Attribution data set for researchers and testing out the various methodologies to create a benchmarking strategies. Feature extraction techniques and author signature models introduced here can be applied in different datasets and text mining domains as well.

7. RECOMMENDATIONS

We have introduced several methods to tackle authorship attribution problems. However, there are several other techniques that still wait to implement on 50-author dataset or other available ones. One of the possible future directions of this workload is the applications of neural network on our dataset. CNN has been simply implemented on dataset without stop words, but due to the computation time we have not further investigated. To proceed with Deep Learning methods, useful features introduced in this work can be easily implemented by stacking them and decreasing the feature dimensionality to give as an input to simple convolutional neural network or an LSTM. The way to implement such neural network algorithms are also introduced. Another take off point of this dataset is to implement Markov Models on word and character level. We have introduced the simple Markov model implementation on our dataset but to further improve the performance of this method, conditional distributions of characters and words need to be stored and redone the experimentation.

One major advantage of our dataset is the large number of books that are available to reproduce. The text pieces given can be used to recombine the books together to apply in different domains such as topic modeling or sentiment analysis per author. It is also possible to create one's own dataset using the provided resources depending on the application. Some of the suggested tasks are twitter author classification, Bitcoin founder identification, and applications of authorship attribution in Forensic science. The problems with Bitcoin founder is that it is still anonymous and there are possible candidates in the web. These are Nick Szabo, Hal Finney, Adam Back, Wei Dai, Craig Wright. These are real personalities and have available large texts on their own forums. By scrapping the writings of these suspects one can create a dataset to follow up the steps we introduced and compare their writing styles with

the unknown founder of Bitcoin. There are also publicly unsolved cases that can be turned into a authorship attribution problems. The robust and fast methodologies can be implemented in suspect detection for law officers.

Last but not least, one problem we introduce in our dataset is the unknown authors that makes % 34 of testing set. Non-exhaustive learning techniques have not yet been studied on this dataset and using the best features we defined one can further investigate into Bayesian non-exhaustive learning methods as in [43].

REFERENCES

REFERENCES

[1] J. Olsson, *Forensic Linguistics: An Introduction to Language, Crime and the Law*, 2nd ed., London, 2008.

[2] *Industrial Society and its Future.* Washington Post, Sept. 22 1995.

[3] M. Haberfeld and A. V. Hassell, *A New Understanding of Terrorism: Case Studies, Trajectories and Lessons Learned.*

[4] A. Tausz, *Predicting the Date of Authorship of Historical Texts*, 2011.

[5] B. Kessler, G. Numberg, and H. Schutze, *Automatic Detection of Text Genre.* ACL '98 Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics Pages 32-38, 1997.

[6] J. R. Thompson and J. Rasp, *Did C.S. Lewis write The Dark Tower?: An Examination of the Small-Sample Properties of the Thisted-Efron Tests of Authorship.* Austrian Journal Statistics, Volume 38 Number 2, 71-82, 2009.

[7] S. Argamon and S. Levitan, *Measuring the Usefulness of Function Words for Authorship Attribution.* Proceedings of ACH/ALLC Conference, 2005.

[8] I. Bozkurt, O. Baglioglu, and E. Uyar, *Authorship Attribution Performance of Various Features and Classification Methods*, 2007.

[9] S. Kim, H. Kim, T. Weninger, and J. Han, *Authorship Classification: Syntactic Tree Mining Approach.* UP'10 Proceedings of the ACM SIGKDD Workshop on Useful Patterns Pages 65-73, 2010.

[10] G. Fung, *Authorship Classification: Syntactic Tree Mining Approach.* The disputed federalist papers: SVM feature selection via concave minimization.

[11] N. Fox, O. Ehmoda, and E. Charniak, *Statistical Stylometrics and the Marlowe-Shakespeare Authorship Debate.* Providence, RI: Brown University M.A Thesis, 2012.

[12] R. Thisted and B. Efron, *Did Shakespeare write a newly discovered poem?*, 1986.

[13] S. Stanko, D. Lu, and I. Hsu, *Whose Book is it Anyway? Using Machine Learning to Identify the Author of Unknown Texts*, 2013.

[14] W. Hu, *Study of Pauline Epistles in the New Testament Using Machine Learning.* Sociology Mind Vol.3, No.2, 193-203, 2013.

[15] T. Putnins, D. Signoriello, M. B. Samant Jain, and D. Abbot, *Advanced Text Authorship Detection Methods and Their Application to Biblical Texts.* he International Society for Optical Engineering (pp. 1-13), 2006.

[16] F. Mosteller, *Inference and Disputed Authorship: The Federalist*, 1964.

[17] J. Hutchins, *History of Machine Translation in a Nutshell*, 2005.

[18] M. Porter, *An algorithm for suffix stripping.* MCB UP Ltd, Program , Vol. 14 Issue: 3, pp.130-137, 1980.

[19] W. H. Gomaa and A. A. Fahmy, *A Survey of Text Similarity Approaches.* International Journal of Computer Applications Volume 68– No.13.

[20] X. Pan and P. Liu, *https://github.com/tensorflow/models/tree/master/research/textsum*, 2016.

[21] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient Estimation of Word Representations in Vector Space.* ICLR Workshop, Jan.

[22] B. Yoshua, D. Rejean, and V. Pascal.

[23] S. Thomas, P. Nguyen, G. Zweig, and H. Hermansky, *MLP Based Phoneme Detectors for Speech Recognition.* ICASSP, 2011.

[24] Y. Kim, *Convolutional Neural Networks for Sentence Classification.* EMNLP, 2014.

[25] T. Mikolov, M. Karafiat, L. Burget, J. H. Cernock, and S. Khudanpur, *Recurrent Neural Network Based Language Model.* Proceedings of Interspeech, 2010.

[26] S. Lai, L. Xu, K. Liu, and J. Zhao, *Recurrent Convolutional Neural Networks for Text Classification.* AAAI, 2015.

[27] H. Sak, A. Senior, and F. Beaufays, *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling.* INTERSPEECH, 2014.

[28] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to Sequence Learning with Neural Networks*, NIPS, 2014.

[29] *The GDELT Project.* https://www.gdeltproject.org/about.html, 2017.

[30] *Kaggle Spooky Author Identification.* https://www.kaggle.com/c/spooky-author-identification/data, 2017.

[31] E. Stamatatos, *A Survey of Modern Authorship Attribution Methods.* Journal of the American Society for Information Science and Technology, 2009.

[32] R. Layton, *Learning Data Mining with Python*, 2015.

[33] J. Grieve, *Quantitative authorship attribution: An Evaluation of Techniques.* Literary and Linguistic Computing, 2007.

[34] C. Lee and A. Bosch, *Exploring Lexical and Syntactic Features for Language Variety Identification.* VarDial, 2007.

[35] L. Yang, C. Li, and Q. D. L. Li, *Combining Lexical and Semantic Features for Short Text Classification*. 17th International Conference in Knowledge Based and Intelligent Information and Engineering Systems, 2013.

[36] A. O. and A. T., *Modeling the shape of the scene: a holistic representation of the spatial envelope*. International Journal of Computer Vision, 2001.

[37] G. Sidorova, F. Velasquez, E. Stamatatos, A. Gelbukh, and L. C. Hernandez, *Syntactic N-grams as Machine Learning Features for Natural Language Processing*. 11th Mexican International Conference on Artificial Intelligence, 2012.

[38] M. Pagliardini, P. Gupta, and M. Jaggi, *Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features*. NAACL, 2018.

[39] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu, *Towards universal paraphrastic sentence embeddings*. International Conference on Learning Representations (ICLR), 2016.

[40] Q. Lee and T. Mikolov, *Distributed Representations of Sentences and Documents*. Proceedings of the 31 st International Conference on Machine Learning, 2014.

[41] A. Coates and A. Y. Ng, *Learning Feature Representations with K-meansNeural Networks: Tricks of the Trade*, 2nd ed. Springer, 2012.

[42] M. Taddy, *Document Classification by Inversion of Distributed Language Representations*. ACL, 2015.

[43] M. Dundar, F. Akova, A. Qi, and B. Rajwa, *Bayesian Nonexhaustive Learning for Online Discovery and Modeling of Emerging Classes*. ICML, 2012.