

**CODE AUTHORSHIP ATTRIBUTION USING CONTENT-BASED
AND NON-CONTENT-BASED FEATURES**

PARINAZ BAYRAMI
Master of E-commerce, Kntu University, 2018

A thesis submitted
in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

© Parinaz Bayrami, 2021

CODE AUTHORSHIP ATTRIBUTION USING CONTENT-BASED AND
NON-CONTENT-BASED FEATURES

PARINAZ BAYRAMI

Date of Defence: June 14, 2021

Dr. J. Rice Thesis Supervisor	Professor	Ph.D.
----------------------------------	-----------	-------

Dr. F. Li Thesis Examination Committee Member	Associate Professor	Ph.D.
---	---------------------	-------

Dr. R. Benkoczi Thesis Examination Committee Member	Professor	Ph.D.
---	-----------	-------

Dr. Y. Chali Chair, Thesis Examination Com- mittee	Professor	Ph.D.
--	-----------	-------

Dedication

I would like to dedicate my thesis to the remarkable woman who always assured me
that everything is going to be alright, My mother.

Abstract

Machine learning approaches are widely used in natural language analysis. Previous research has shown that similar techniques can be applied in the analysis of computer programming (artificial) languages. In this thesis, we focus on identifying the authors of computer programs by using machine learning techniques. We extend these techniques to determine which features capture the writing style of authors in the classification of a computer program according to the author's identity. We then propose a novel approach for computer program author identification. In this method, program features from the text documents are combined with authors' sociological features (gender and region) to develop the classification model. Several experiments have been conducted on two datasets composed of computer programs written in C++, and the results are encouraging. According to the experimental results, the author's identity can be predicted with a 75% accuracy rate.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, professor Jacqueline E. Rice who always supported me throughout my graduate journey. I am grateful for the patient guidance and advice she has provided during my graduate study. I have been extremely lucky to have a supervisor who cared so much about my research.

I am truly grateful to my family and fiance for their unconditional support and love.

Contents

Contents	vi
List of Figures	viii
List of Tables	xi
1 Introduction	1
1.1 Motivation and Contribution	4
1.2 Organization of Thesis	6
2 Background and Literature Review	7
2.1 Machine Learning	7
2.2 Text Mining	9
2.2.1 Data Collection	10
2.2.2 Data Preprocessing	11
2.2.3 Metric/Feature Extraction	14
2.2.4 Classification	23
2.2.5 Model Evaluation	29
2.2.6 Related Work	33
3 Methodology	39
3.1 Data Collection	39
3.2 Document Representation and Preparation	41
3.3 Metrics/Features Extraction	42
3.3.1 Software Metrics	42
3.3.2 N-grams	44
3.3.3 Similarity Measurement	46
3.3.4 Lexical Diversity	48
3.3.5 Sociolinguistic Characteristic	48
3.4 Programming Environment	49
3.5 Summary	50
4 Experiments and Results	51
4.1 Experiments	51
4.1.1 Experiment 1	53
4.1.2 Experiment 2	53
4.1.3 Experiment 3	55
4.1.4 Experiment 4	56

4.1.5	Experiment 5	56
4.1.6	Experiment 6	56
4.2	Results	57
4.2.1	Experiment 1 Results	59
4.2.2	Experiment 2 Results	60
4.2.3	Experiment 3 Results	61
4.2.4	Experiment 4 Results	62
4.2.5	Experiment 5 Results	63
4.2.6	Experiment 6 Results	64
5	Discussion	66
5.1	Performance Discussion of Experiments 1, 2, and 3 on the Github Dataset.	66
5.1.1	Accuracy distribution	70
5.1.2	Gender-based Analysis	70
5.1.3	Region-based Analysis	72
5.2	Performance Discussion of Experiments 4, 5, and 6 on the Codeforce Dataset.	73
5.2.1	Accuracy distribution	77
5.2.2	Gender-based Analysis	77
5.2.3	Region-based Analysis	78
5.3	Feature Analysis	79
5.4	Author Identification using Top-ranked Features	81
5.5	Word Frequencies	83
6	Conclusion	87
6.1	Future work	89
	Bibliography	92

List of Figures

2.1	The red rectangle is a row of data, known as instance, which consists of an array of observations from the domain, such as height and width. The blue rectangle indicating a single column of data is called a feature. In this example the labels are the classes of the animals that should be predicted.	10
2.2	Data structure for the author identification problem. Text samples are stored as instances in rows, while columns represent features. The labels (output) are the author’s identity.	12
2.3	Document representation using bi-gram approach. The bi-gram variables extracted from ‘One Cent, Old Cent, New Cent’.	16
2.4	A is an $m * n$ matrix. m is the number of input documents (files) and n is the number of features. To see the full example of SVD decomposition of this matrix see [62].	17
2.5	Illustration of singular value decomposition (SVD) and reduced singular value decomposition (RSVD) approaches.	18
2.6	Geometrical representation of two documents, Doc1 and Doc2, where $Y = \text{new}$ and $X = \text{old}$.	22
2.7	RF bootstrap sampling.	25
2.8	Concept of RF classification. In training set, the algorithm creates T multiple bootstrapped samples, and then builds a classification trees from each bootstrapped sample set. For classification, each tree gives a unit vote for the most popular class at each input data. The final label is determined by a majority vote of all trees.	26
2.9	OOB data are separated for each bootstrapped sample. The accuracy performance is evaluated by testing the trees with the separated OOB data, and then the performance of the RF is calculated by taking the average of these performance rates.	27
2.10	Separating hyper-planes, or possible decision boundaries, which can separate two possible classes. Circles indicate documents with “negative” labels and squares represent documents with “positive” labels.	30
2.11	Choosing the optimal decision boundary using SVM: the distance to the closest negative (circle) point should be equal to the distance to the closest positive (square) point.	30
2.12	Confusion matrix for multiple classes. This example represents a three-class problem with the classes A, B, and C.	32
3.1	Information about data distribution in our datasets.	40
3.2	An example dataset of four texts written by two authors.	41

3.3	Tokenization process for a dataset with four documents written by two authors. The tokenization technique divides each program line into identified tokens. Each row represents a text document, each column represents a distinct token, and each cell represents a count of the tokens in the document.	42
3.4	Example showing how 5 keywords in TF-IDF format represent the dataset of four texts with software metrics.	44
3.5	3-gram extraction process for our example dataset with four documents belonging to two authors. Each row represents a text document, each column represents a 3-gram (3 sequential tokens), and each cell represents a count of the sequence (terms) in the document.	45
3.6	Data frame in the example dataset showing 3-gram frequencies after applying TF-IDF.	45
3.7	Example dataset represented by 3-gram frequencies after applying TF-IDF and reducing the feature set using SVD.	45
3.8	The similarity and lexical diversity features of our example dataset of four texts written by two authors. We compared the similarity value between text1 and text2 written by author A and then calculated the value of similarity between text3 and text4 written by author B. As we were considering cosine distance between two texts, the similarity entries were repeated. A value of 0 indicates that the documents do not have any similarity and a value of 1 indicates that they are identical.	47
3.9	A visualization of five categories of features we extracted using machine learning techniques for a dataset that contained four texts from two authors. Each category was coloured a different shade.	49
4.1	An overview of the six experiments in this thesis.	52
4.2	Steps for experiments 1 and 4.	53
4.3	Steps for experiments 2 and 5.	55
4.4	Steps for experiments 3 and 6.	57
4.5	Confusion matrix for multi-class machine learning model in our work.	58
5.1	Experiment 2 using $n \in \{3, 6, 10\}$ to establish optimal settings for the choice of n-gram length. Three classification models were employed, and their performance was demonstrated with each feature set.	67
5.2	Comparative results of experiments 1,2, and 3. The results of experiment 1, experiment 2, and experiment 3 respectively appear in EX1, EX2, and EX3.	68
5.3	Each of the three experiments evaluated the accuracy performance of the RF model by averaging individual accuracy scores of 60 classes.	69
5.4	Accuracy distribution by class for the GitHub dataset.	70
5.5	Label distribution of individual accuracy over 80% when grouped by gender and region. For experiments 1, 2, and 3 (respectively) results are labeled EX1, EX2, and EX3.	72
5.6	Each of the three experiments 4,5 and 6 evaluated the accuracy performance of the RF model by averaging individual accuracy scores of 60 classes.	74

5.7	Experiment 5 using $n \in \{3, 6, 10\}$ to establish optimal settings for the choice of n-gram length. Three classification models were employed, and their performance was demonstrated with each feature set.	75
5.8	Comparative results of experiment 4 (EX4), experiment 5 (EX5), and experiment 6 (EX6).	76
5.9	Accuracy distribution by class for the Codeforces dataset.	76
5.10	Label distribution of individual accuracy more than 80% concerning gender and region in experiment 4 (EX4), experiment 5 (EX5), and experiment 6 (EX6).	78
5.11	Top 15 highest-ranked features in Experiments 1,3,4 and 6.	80
5.12	Top 15 frequently occurring words from datasets Coll-G and Coll-F.	84
5.13	Top-ranked features of experiment 4. The red line indicate the keywords.	85
5.14	Top-ranked features of experiment 1. The red line indicate the keywords.	85

List of Tables

2.1	Tokenization of the phrase ‘One Cent, Old Cent, New Cent’	12
2.2	The number of word types (V) and the number of tokens (N) in a text phrase ‘One Cent, Old Cent, New Cent’.	19
2.4	Matrix with two documents and two terms.	22
3.1	List of 18 software metrics.	43
4.1	Comparison of previous contributions [54].	59
4.2	Experiment 1 results.	59
4.3	Results from experiment 2 with feature set A.	60
4.4	Results from experiment 2 with feature set B.	61
4.5	Results from experiment 2 with feature set C.	61
4.6	Experiment 3 results.	62
4.7	Experiment 4 results.	62
4.8	Results from experiment 5 with feature set A (3-grams).	63
4.9	Results from experiment 5 with feature set B (6-grams).	63
4.10	Results from experiment 5 with feature set C (10-grams).	64
4.11	Experiment 6 results.	64
5.1	The RF label distribution of individual accuracy from experiment 1, 2, 3 grouped by gender.	70
5.2	The RF label distribution of individual accuracy from experiment 1, 2, 3 grouped by region.	72
5.3	The RF label distribution of individual accuracy from experiment 4, 5, 6 grouped by gender.	77
5.4	The RF label distribution of individual accuracy from experiment 4, 5, 6 grouped by region.	78
5.5	Comparing the results of experiment 3 using the 15 most important features and the 32 initial features.	82
5.6	Comparing the results of experiment 6 using the 15 most important features and the 32 initial features.	82

Chapter 1

Introduction

Stylometry, or authorship attribution, is the study of style in language, based on a set of techniques that attempt to identify a person's writing style through determining the unique characteristics that can be derived from the author's books or writing samples [19]. For instance, if a professor writes some notes, they may write them in a certain way and use certain phrases. Those phrases create a pattern that is characteristic of their writing and talking style. Language is divided into the following two groups: natural language (such as English or Arabic) and artificial language (which is created for controlling the behavior of a machine) [37]. Stylometry looks at both groups of languages.

Authorship attribution (stylometry) in natural language is analyzing texts or speeches to determine their authorship [19]. For instance, some scholars examined English written texts to identify the authorship. JK Rowling is known for her Harry Potter series, but she is also a bestselling author of crime novels. In 2012, a poem named *The Cuckoo's Calling*, written by Robert Galbraith, was published. It was Galbraith's first poem, and surprisingly the poem became a bestseller overnight. A journalist said he had recalled a tip that the writer was actually JK Rowling. A group of British researchers in Pennsylvania tested this by applying authorship attribution. Their tests confirmed that the poem was written by JK Rowling. She then admitted to the press that it was indeed her work. This is an example of how stylometry can be used on natural language [3].

Authorship profiling is one use of stylometry in natural language. Profiling is used to detect an author’s sociolinguistic characteristics—such as age, gender, location, educational background, and personality traits—by examining writing patterns in written text [19]. Studies have shown that the use of natural language varies by the social context of a situation, the speaker, and the audience. These variations reveal aspects of our social identity [92]. The differences that influence natural language use mostly occur due to sociolinguistic characteristics.

The second broad category of language is artificial language. A programming language is an artificial language [74]. In some studies of programming languages, researchers have investigated whether coding style is unique for each programmer [68]. They have found that just as each speaker of a natural language will reveal their identity through their language choices [5], so each programmer can demonstrate their unique fingerprint and individuality in their code. It might be thought that the standard guidelines for aspects of programming style—such as layout, naming conventions, and source file organization—are too strict for individual style to be demonstrated in a program language. Nonetheless, there is much freedom and choice left up to programmers, including the use of keywords, operators, statements, the choice of standard library functions, and the use of white space. Thus, we can use these choices to investigate author style [22].

Code stylometry is the application of stylometry to attribute anonymous program code. Since an individual learns to code independently, it should be possible for them to develop a unique coding style. Additionally, the use of programming language among programmers may differ depending on sociolinguistic factors such as gender and region, just as the compelling impact of sociolinguistic characteristics on natural language use (authorship profiling) [83, 74]. Code stylometry (authorship attribution) can be used for software forensics [19], detecting plagiarism [88], and Canadian copyright investigation [43]. However, at the same time, code stylometry is often also

privacy infringing depending upon how it is used. The following is a real example of using stylometry to identify the software programmer.

Saeed Malekpour is an Iranian programmer who developed photo uploading software. Stylometry identified him as a web programmer for a pornography website. When an Iranian court sentenced him for his work on this software, he confessed that he did not know that this website was using his program. If he had known, he would never have allowed it, as it is illegal in Iran [23]. Code stylometry is a double-edged sword, and it could be a concern for programmers who want to remain anonymous.

Code authorship attribution can be formed as a machine-learning problem/task. Machine learning is a process with the ability to automatically learn and improve performance or make accurate predictions from experience [32]. In authorship attribution there are a set of documents of known authorship, and the goal is to use them to classify a document of unknown authorship. To perform the authorship attribution analysis, a traditional workflow of a supervised learning algorithm for classification is required, where each document (or text entry) is partitioned into a set of features and those features are associated with the user-defined output (labels). Next, labels or known class and extracted features are then passed to the training phase where machine learning algorithms are used to identify a good model which map inputs to desired outputs. Once a model is constructed, it is used to fit new (unlabeled) data [8].

In this thesis, we explore the possibility of determining the author of an unattributed piece of work/program based on the features of the candidate's writing style and their sociolinguistics characteristics [74, 83]. In this research, two data sets with different contexts are used: Codeforces (contest-based programming website) and the GitHub repository (non-contest-based programming website). We also examine statistical differences in the importance of the features in different programming contexts. Considering the programming website's context leads us to find an appropriate feature set and to identify the author of written computer programs more accurately. In

this study, we use machine learning tools to investigate the programming style of programmers using several popular machine learning techniques, including random forest, support vector machine and naïve Bayes.

1.1 Motivation and Contribution

The stylometric analysis of textual data can numerically represent style expressed in natural language and programming language. The existence of stylometry techniques can provide key information in several basic areas [57]:

- In industry, some companies may wish to determine which employee wrote harmful code; or, whenever a particular program needs to be rewritten, the author may need to be located. It would be convenient to be able to determine the name of the programmer who wrote a particular piece of code from a set of several hundred programmers, so they can be located to assist in the upgrade process.
- In the academic community, it is considered unethical to copy programming assignments. A professor wants to determine if students are plagiarizing assignments. While plagiarism detection can show that two programs are equivalent, authorship analysis can show that some code fragment was indeed written by the person who claims authorship of it.
- The legal community wants to determine authorship in disputes and to identify cybercriminals. Governments need practical methodologies that can be used to provide empirical evidence to show that the same person wrote two or more programs.

There have been many works in authorship attribution in computer programs, starting in 1993 with Eugene Howard Spafford, a professor of computer science at

Purdue University. Spafford’s research [87] suggests that measuring style in programming language could aid in program code authorship attribution. Our research builds on prior works in program stylometry [22, 74, 83].

University of Lethbridge M.Sc. graduates Naz and Rafee implemented machine-learning techniques to analyze programmer coding style based on the programmer’s sociolinguistic characteristics [74, 83]. Naz used vocabularies of programming language (e.g., keywords, operators, loops, and comments) as features to classify the programming language based on gender. Rafee focused on a different group of features consisting of 13 IEEE standard software metrics, such as the numbers of code lines to categorize the programmer based on regions. Burrows [22] also studied authorship attribution in computer programs. In his experiment, he used stylometry to identify anonymous programmers in large programming datasets using n successive words (N-gram) as features. However, the use of these features or programming elements often depends on the problem. Burrows extended his work to examine the combinations of features, using three categories of features: N-gram, vocabularies of programming language, and software metric features. The hybrid features have been shown to be more effective for code stylometry. That is why, in this research, we try to discover and compare the different combinations of feature categories of the program code, rather than use a particular feature category.

Argamon’s research in natural language analysis has explored the sociolinguistic factors of written documents [11]. Argamon suggested that it is possible to find a link between an individual’s linguistic expression and their sociolinguistic characteristics, such as region and gender [11]. Naz worked to assess the impact of gender on the programming language [74]. In Naz’s research, the program classification showed that gender does appear to affect the use of programming language. Following this, Rafee also investigated the gender and region difference in the use of programming languages [83].

Our work is novel because we seek to evaluate the effectiveness of adding gender and region to programming content-based features, such as n-gram and keywords. We use real-world data from the public GitHub and the competitive programming website Codeforces. GitHub repository is a non-contest-based programming website, while Codeforces is a contest-based programming website. These data sources provide a real-world space to examine stylometry techniques to be applied in two different dataset concepts. We used R programming to apply machine-learning techniques. R is an open-source programming language and is widely used in data-mining problems. R supports various machine-learning functions such as data processing, feature extraction, supervised and unsupervised learning, and model evaluation of different kinds of data.

1.2 Organization of Thesis

This chapter has given an overview of the research work, followed by a discussion of the motivation and contributions of this work and the organization of the thesis. Chapter 2 provides background material important for understanding the remainder of the thesis, including definitions, fundamentals of machine learning, and text-mining methods. Chapter 2 ends with a review of the related research in these fields. Chapter 3 presents the methodology we used to classify written computer programs based on author identity, and discuss the programming environment in this chapter. We also explain all the steps that we implement in our approaches. Chapter 4 describes six experiments and numerical results. Chapter 5 provides some analysis of features to identify the author of the programs. In chapter 6, we conclude this research with the discussion of possible research directions and future work.

Chapter 2

Background and Literature Review

2.1 Machine Learning

Machine learning can be defined as a process with the ability to automatically learn from experience to make accurate predictions or improve performance [32]. Experience is the information available related to the past period of time. In 1997, Mitchell explained machine learning as follows: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ” [73]. At a high level, machine learning algorithms are classified into two groups: supervised and unsupervised, according to the way they “learn” about data [53].

- Unsupervised machine learning, or clustering, is concerned with finding any hidden pattern within a dataset. The data provided to the unsupervised algorithm are not labelled, which means only the input variables are given with no corresponding output variables. A clustering analysis uses the similarity and dissimilarity between data features to group them into clusters. For example, businesses or marketing campaigns can use clustering to segment customers by personal attributes and identify key differentiators that divide customers. As a result, different marketing campaigns targeting various types of customers can be designed [14].
- Supervised machine learning algorithms, or classifiers, are used when training data consists of input variables (X), paired with the correct output variable (Y),

and an algorithm is used to learn the mapping function from the input(s) to the output (see Equation 1). The algorithm will search for patterns in the data that fit the relationship between the input attributes and the correct outputs. For example, a classification algorithm will learn to distinguish animals after being trained through repeated experience with a dataset of images that are properly labelled with the species of the animal and some identifying characteristics [14].

$$Y = f(X) \tag{1}$$

The goal is to approximate the mapping function so that when the algorithm is provided with new unlabeled input(s), it can determine which label the new inputs should be classified as based on prior training data. This is called supervised learning because the process can be thought of as having a teacher supervise the learning process.

The first step in solving a problem with machine learning is to determine how to represent the learning problem in terms of something the computer can understand. Next, users must choose a learning model, typically the classifier users want the system to use.

Data are raw facts, values, text, sound, or pictures that refer to, or represent, conditions, person, ideas, or objects. Since the collected data may not be in a format that is easily used by the machine learning algorithm, further modification may be needed to represent the data in the desired format for the learning algorithms. For example, an object can be represented by formulating the description of the object. Each input object, which we often call a sample/instance, is converted into a set of features that describes the object.

In the field of machine learning, a common structure for data representation consists of rows and columns, such as a database table or an Excel spreadsheet as shown

in Figure 2.1. A row of data is known as instance, which consists of an observation from the domain (a red rectangle in Figure 2.1). A single column of data is called a feature or an attribute shown as a blue rectangle in Figure 2.1. A feature or attribute is a component of an observation. These features describe each of the instances. The label or class is the attribute, or factor, that should be predicted, or the goal of prediction. A dataset is a collection of instances used to train the system. If users wish to predict an animal's type, each instance will be labeled with the name or type of the animal. Each instance, in turn, is composed of features that describe it, such as the animal's weight and mass. In machine learning, a vector that stores the feature values of an instance is known as feature vector.

Normally in machine learning applications, the main dataset is divided into two subsets: a training set and a test set. The training set consists a set of data used as input to a supervised machine learning algorithm to train the machine learning model. The test set is a collection of data used to evaluate the model performance after it has been trained. The test data may not be provided to the machine learning algorithm model during training.

2.2 Text Mining

Machine learning has a broad set of practical applications, including text mining: the process of extracting useful information and identifying concealed patterns in large repositories of text data [4]. One area of text mining is linguistic stylometry, where analyzing an author's writing style can potentially identify who authored a specific text. Computer program code is a special form of text. Combinations of words, and different coding expressions are used to form the program, or a set of instructions for computers [30]. Style expressed in computer code also can be quantified and characterized. The general assumption is that each programmer automatically tends to use the same linguistic pattern, which is unique to them [68]. In this way, it may

Label	Features			
Label	width	height	mass	color
Bird	7.5	7.1	162	Red
Snake	3.5	20.1	800	Yellow

Figure 2.1: The red rectangle is a row of data, known as instance, which consists of an array of observations from the domain, such as height and width. The blue rectangle indicating a single column of data is called a feature. In this example the labels are the classes of the animals that should be predicted.

be possible to differentiate programmers' identities by mining their code.

Author identification, also known as author attribution or stylometry, deals with attributing authorship of an unknown program based on similarities and features common to an author's known works. Author identification consists of various steps, namely data collection, data processing, feature extraction, classification, and model evaluation (evaluating the output to determine whether useful knowledge was discovered) [32].

2.2.1 Data Collection

The first step in any authorship attribution study is the gathering of data. Researchers for authorship attribution should give thought to the following considerations:

1. Samples must be of single authorship for authorship attribution. Obtaining single-authored samples for constructing source code collections is challenging since most of the time, software efforts tend to be collaborative in nature, which

limits the availability of meaningful single-author samples [19].

2. Having a high number of programmers makes the problem of authorship attribution complicated and more difficult to solve [19]. The greater the number of outputs the higher complexity that is added to the model [17]. A system is defined as complex according to the number of inputs and outputs it has. For example, labelling a sample as dog, cat, or human requires several features, such as weight, height and number of legs. However, to classify cats and humans, the number of legs might be the only feature needed [9]. Also take, for example, a typical authorship attribution problem, where an unattributed piece of text is assigned to one of four possible authors. With four candidate authors, this problem is described as four-class. In this example there is a 25 percent chance of identifying the author by random chance. However, if there were a hundred possible authors, the chance to identify the authors by random chance is reduced to 1 percent [19].
3. Having a higher number of samples per author is preferable. This increases the amount of training data available to model the writer's style [19].

2.2.2 Data Preprocessing

Preprocessing can have a significant influence on the success of text mining [91]. Preprocessing is a text mining technique used to transform the text data into an understandable, predictable, and analyzable form before it can be imported to a machine-learning algorithm. Machine-learning algorithms operate on a numeric feature space. To perform machine learning on text, we must transform our documents into vectors or numerical representation, generally as a two-dimensional array, where rows are the instances and columns are the features.

The first step in representing text documents as vectors is decomposing the documents into distinct pieces or tokens. This process is called tokenization and involves

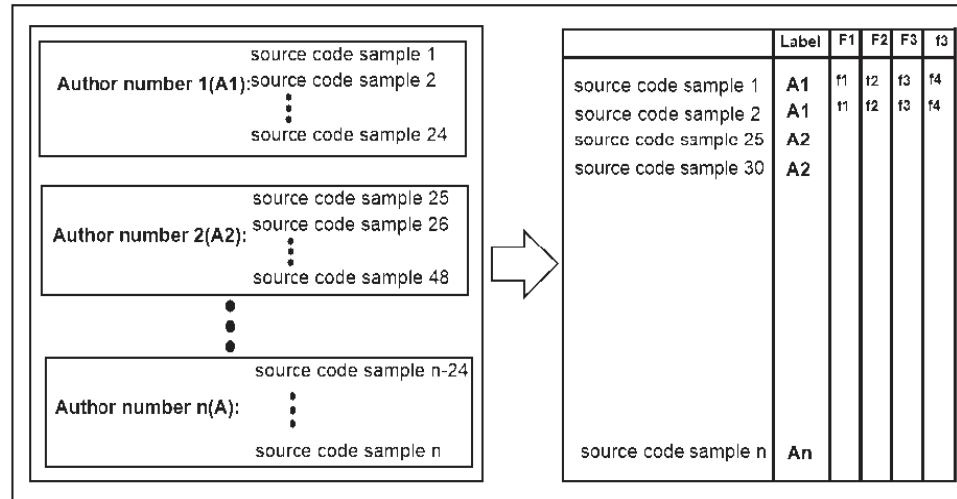


Figure 2.2: Data structure for the author identification problem. Text samples are stored as instances in rows, while columns represent features. The labels (output) are the author's identity.

splitting longer strings of text, code or character sequences into tokens [8]. Typically, tokens include words, symbols, and numbers in natural language. In programs (code), tokens may include functions, punctuation, operators, and keywords.

The next step is to build a dataset where

- each row represents a document,
- each column represents a distinct token, and
- each cell is a count of how often that token occurs in the document.

Let us take the simple sentence 'One Cent, Old Cent, New Cent' as an example. The tokenization result is shown in Table 2.1.

Table 2.1: Tokenization of the phrase 'One Cent, Old Cent, New Cent'.

Tokens	One	Cent	Old	New
Doc	1	3	1	1

To evaluate how relevant a token is to a document in a collection of documents, TF-IDF can be used. TF-IDF is a vector-space model, which creates a numerical rep-

resentation of textual documents in terms of features [74]. TF-IDF uses two metrics: term frequency (TF) and inverse document frequency (IDF).

Term frequency (TF):

- Let $freq(t, d)$ be the count of occurrence of the term t in document d ,
- $TF(t, d)$ be a proportion of the count of term t in document d , and
- n be the number of distinct terms in document d and $i \in \mathbb{Z}^+$ is a positive integer. Then

$$TF(t, d) = \frac{freq(t, d)}{\sum_i^n freq(t_i, d)} = \frac{\text{Frequency of term } t \text{ in document } d}{\text{Total numbers of terms in a document } d} \quad (2)$$

Inverse document frequency (IDF):

- Let N be the count of distinct documents in the dataset, and
- $count(t)$ be the count of documents in the dataset in which the term t is present. Then

$$IDF(t) = \log\left(\frac{N}{count(t)}\right) \quad (3)$$

TF-IDF:

- TF-IDF values are calculated by multiplying TF and IDF to enhance document representation.

$$TF-IDF(t, d) = TF(t, d) * IDF(t) \quad (4)$$

2.2.3 Metric/Feature Extraction

Stylometry (explained in Section 2.2) has been widely applied to various areas. For example, it has been studied in fine art for attributing works to particular artists. Style in art can include shapes, designs, textures, and colors, which may present in a way that looks unique [66]. Stylometry has been applied in music to study harmony and melody [13]. However, the most common study of style has been applied to text and source code [23]. The theory in program (code) stylometry is that everyone learns coding on an individual basis and unconsciously tends to write in relatively consistent, recognizable, and unique ways [23]. Differences in coding style make the identification of the author possible since the coding style expressed in computer code can be quantified. The underlying notion of style is that works by different programmers are strongly distinguished by quantifiable features of the programs, known as style markers [54, 68]. These style markers may include various software metrics, as well as language metrics such as n-grams and lexical diversity. In this section we discuss the methods of extracting these features. We begin by reviewing software metrics definitions. We then review n-grams, lexical diversity, and the cosine similarity feature, which are categorized as content-based features. Content-based features are contained in the program code. Following that, we explain the authors' sociolinguistic characteristic features, which we categorize as practical non-content based features.

Software Metrics

Krsul et al. [57] collected metrics for authorship identification from a wide variety of sources [78] and grouped them in a package named software metrics. Software metrics is a broad category of features which can be further categorized according to their relevance to programming layout, programming style, and programming structure [68]. Programming layout specifies the form and pattern of the source code, such as the number of code lines, blank lines, and spaces. Programming style deals

with characteristics that are difficult to change automatically by code formatters (for example, mean variable length and mean comment length). Programming structure metrics are connected to the programmer's skills and experience, such as keywords and usage of data structures.

N-grams

N-gram modeling is a popular feature identification technique used in natural language processing [5]. An n-gram is a contiguous sequence of n words extracted from a text or program. Each text is considered to be a sequence of words. To generate the n-grams from the text, all the overlapping sequences of n consecutive words are extracted [5]. N-grams allow us to capture information about tokens occurring near one another, which the software metrics cannot do [19]. For example, the bi-grams (n=2) and tri-grams (n=3) corresponding to the sentence 'One Cent, Old Cent, New Cent' are:

Bi-grams: (One Cent) (Cent ,) (, Old) (Old Cent) (Cent ,) (, New) (New Cent) (Cent .).

Tri-grams: (One Cent ,) (Cent , Old) (, Old Cent) (Old Cent ,) (Cent , New) (, New Cent) (New Cent .).

Bi-gram variables keep track of which token pairs occur and how often two consecutive tokens occur in each text shown in Figure 2.3. Tri-gram variables keep track of which three consecutive tokens occur and how often three consecutive tokens occur in each text. The n-gram technique is used to identify groups of two or three tokens that repeat throughout the sample.

N-gram language models have proven effective in text categorization problems when applied to any language or even non-language scripts such as music or DNA [67]. Adding n-grams to the feature set can increase the model's efficiency because the machine-learning algorithms will learn more signals in the dataset [76]. These n-

	Bi-gram variables			
sentences (text documents)	One Cent	Cent ,	. . .	Cent .
'One Cent, Old Cent, New Cent'	1	2	. . .	1
⋮				

Figure 2.3: Document representation using bi-gram approach. The bi-gram variables extracted from ‘One Cent, Old Cent, New Cent’.

grams capture grammatical and orthographic preferences that include capitalization, punctuation and white spaces [76].

However, the n-gram approach considerably increases the dimensionality of the problem in comparison to other approaches due to large numbers of n-gram features. The dimensionality of the feature vectors also may be intractable in terms of memory and time requirements. The other problem is sparsity: n-grams are a sparse representation of text data. This is because we build the features based on the sequence of n words co-occurring. The approach will give a zero value to all sequences of n keywords that are not present in the other programs. For example, the sequence “include math” which occurs in one computer program, may never occur in the other programs in the dataset. This is called the curse of dimensionality [41].

Therefore when n-grams are used, dimensionality reduction is of crucial importance. Singular value decomposition (SVD) is used to reduce dimensionality. The major purpose of SVD is to reduce a dataset involving a large number of values to a dataset involving significantly fewer values, but which still contains a large fraction of the variability present in the original data [47]. In our work, SVD reduced the overall dimensionality of the input matrix to a matrix of much smaller size with fewer variables. The input is a matrix, with documents designated by rows and n-gram terms by columns. The elements of the matrix are the n-gram counts. In other studies [16], researchers applied SVD to the input matrix and the resulting reduced matrix still

		n features (n-gram)				
		First 2gram sequence	Second 2gram sequence	Third 2gram sequence	Fourth 2gram sequence	Fifth 2gram sequence
Author 1	document 1	1	1	1	0	0
	document 2	3	3	3	0	0
	document 3	4	4	4	0	0
	document 4	5	5	5	0	0
Author 2	document 5	0	2	0	4	4
	document 6	0	0	0	5	5
	document 7	0	1	0	2	2

Figure 2.4: A is an $m * n$ matrix. m is the number of input documents (files) and n is the number of features. To see the full example of SVD decomposition of this matrix see [62].

proved sufficient to capture the language context.

As shown in Figure 2.4, A is an $m * n$ matrix where m represents the number of input documents (files) and n represents the number of features. The SVD of the document-term matrix A_{m*n} splits the matrix into three sub-matrices or “factors” [2, 72] as described in Equation 5.

$$A = U\Sigma V^T \quad (5)$$

where U is $m * n$ with orthonormal columns, V is $n * n$ with orthonormal columns, and Σ is diagonal with the main diagonal entries sorted in decreasing order. A matrix U is orthogonal if $U^T U = I$, or the inverse of U is its transpose.

To reduce dimensionality, a truncated SVD of the term-document matrix was used, where A is approximated by

$$A \approx U_k \Sigma_k V_k^T \quad (6)$$

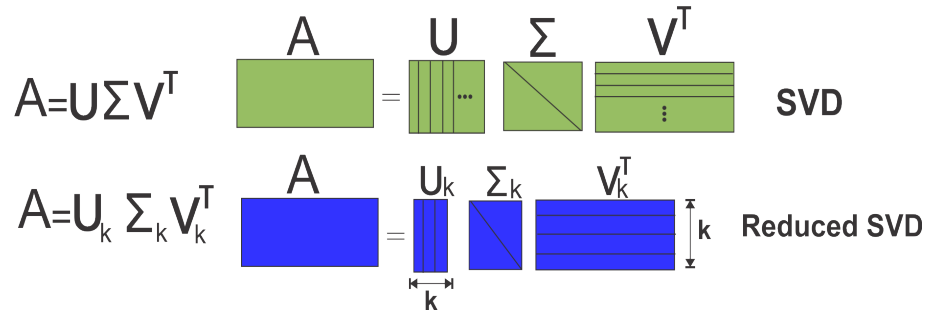


Figure 2.5: Illustration of singular value decomposition (SVD) and reduced singular value decomposition (RSVD) approaches.

where U_k is the first k columns of U , V_k is the first k columns of V , and Σ_k is the upper left k by k part of Σ). This gives the best rank k approximation to the original matrix. Because a full SVD is not required the truncated SVD is usually computed by an iterative technique [79] (see Figure 2.5).

Lexical Diversity Metrics (LDMs):

Lexical diversity is “the proportion of words in a language sample that are not repetitions of words already encountered” [51]. LDMs refer to the range of different unique words used in a text, with a greater range indicating a higher diversity [71]. Lexical diversity is useful for analyzing speakers’ or writers’ linguistic skills or the complexity of ideas expressed in documents [38]. Gregori [38] claimed that it is possible to obtain a reliable measure of lexical diversity that is stable across two pieces of writing produced by the same writer. Some studies [51, 70, 65] dealing with the assessment of lexical diversity show that the scores given by metrics are highly variable and some may work better than others, depending on the data under scrutiny. In this research, we use 9 metrics based on Lissón’s research [65], namely Type–token ratio (TTR), R Guiraud’s Root TTR, Carroll’s Corrected TTR (CTTR), Herdan’s C (LogTTR), Uber Index (U), Yule’s K (K), Yule’s I (I), Simpson’s D , Maas, and V_m .

- Type–token ratio (TTR) and several of its transformations, such as Guiraud’s Root TTR (R) and Carroll’s Corrected TTR (CTTR), is probably the most

Table 2.2: The number of word types (V) and the number of tokens (N) in a text phrase ‘One Cent, Old Cent, New Cent’.

Word	Frequency
One	1
Cent	3
Old	1
New	1
4 word types (V)	6 tokens (N)

well-known measure of lexical diversity. TTR is the ratio of the number of word types (V) to the number of words (N) in a text [28] (see Equation 7). “Tokens” refers to the sum of all words in the text (total number of words), where “types” refers to each individual word (different words). For example we expressed the number of word types (V) and the number of tokens (N) in a text phrase ‘One Cent, Old Cent, New Cent’ in Table 2.2. In the following formulas, N refers to the total number of tokens, and V to the number of types.

$$TTR = \frac{V}{N} \quad (7)$$

- R Guiraud’s Root TTR is the ratio of the number of types divided by a square root of the number of tokens in a given text [90].

$$R = \frac{V}{\sqrt{N}} \quad (8)$$

- Carroll’s Corrected TTR (CTTR) is the ratio of the number of types divided by a square root of twice a number of tokens in a given text [70].

$$CTTR = \frac{V}{\sqrt{2N}} \quad (9)$$

- However, many studies have shown that TTR highly depends on text length [90].

As a consequence, some interesting attempts to improve the TTR index have been proposed in the literature. In the following, a TTR variant proposed by Herdan (1960) and usually addressed as Herdan’s C or LogTTR is reported here [65]. In Equation 10, N refers to the total number of tokens and V to the number of types.

$$C = \frac{\log V}{\log N} \quad (10)$$

- Uber Index reflects lexical diversity by relating the total number of lexical words used (N) in a text sample with the number of unique lexical words used (V). A high Uber Index value would indicate that complex sentence structures are being used in an individual’s communication [70, 90].

$$U = \frac{\log^2 N}{\log N - \log V} \quad (11)$$

- Yule’s K measure is based on lexical repetitions. The formula for calculating Yule’s K is shown in Equation 12. If we obtain a low value, the text analyzed is rich in vocabulary. On the contrary, if we obtain a greater value, the text analyzed contains less vocabulary richness. Therefore, the larger the result of Yule’s K , the greater the number of repeated words and the vocabulary appears to be simpler in the text [65]. In the following formulas, N refers to the total number of tokens, V to the number of types, and $f_v(i, N)$ to the numbers of types occurring i times in a sample of length N [65].

$$K = 10^4 \times \left[-\frac{1}{N} + \sum_{i=1}^V f_v(i, N) \left(\frac{i}{N}\right)^2 \right] \quad (12)$$

- Yule’s I is based on the reciprocal of Yule’s K : the larger Yule’s I , the greater

the diversity of the vocabulary [65].

$$I = \frac{V^2}{M_2 - V} \quad (13)$$

$$M_2 = \sum_{i=1}^V i^2 * f_v(i, N)$$

- The next measure is V_M . This indicator can be considered another modification of Yule's K. Herdan defined V_m as follows [44]:

$$V_m = \sqrt{\sum_{i=1}^V f_v(i, N)(i/N)^2 - \frac{i}{V}} \quad (14)$$

- Indices from outside the field of linguistics have sometimes been used to measure the LDMs [90]. For example, Simpson (1949) derived the following formula as a measure to capture the diversity of a population [86]. Simpson's D also has been used to measure LDMs:

$$D = \sum_{i=1}^V f_v(i, N) \frac{i}{N} \frac{i-1}{N-1} \quad (15)$$

Cosine Similarity Feature

A typical way to discriminate between objects is by representing those objects via their similarities or dissimilarities. The cosine similarity feature is a measure of the degree of similarity between two documents [60]. Concepts and issues specific to the generalizability of using similarities as features were reported by varying researchers [81, 19]. Cosine similarity within the text-mining domain calculates the cosine between two documents, for example Doc1 and Doc2. Documents must be represented by vectors of features. As the text documents are represented in a vector space, it allows us to work with documents geometrically (see Figure 2.6). Calculating similarities between two documents can be expressed as:

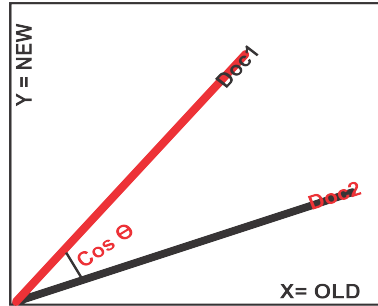


Figure 2.6: Geometrical representation of two documents, Doc1 and Doc2, where Y = new and X = old.

$$Sim(Doc1, Doc2) = \cos\Theta = \frac{Doc1 \cdot Doc2}{\|Doc1\| \|Doc2\|} = \frac{\sum_{i=1}^n A_i \cdot B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}} \quad (16)$$

Table 2.4: Matrix with two documents and two terms.

Terms:	Old	New
Doc1:	6	10
Doc2:	10	3

For example, take a hypothetical matrix with two documents and two terms. In Table 2.4, Doc1 consists of 6 ‘old’ terms and 10 ‘new’ terms, and Doc2 consists of 10 ‘old’ terms and 3 ‘new’ terms. The similarity rate between Doc1 and Doc2 can be measured as:

$$Sim(Doc1, Doc2) = \cos\Theta = \frac{(6 \times 10) + (10 \times 3)}{\sqrt{6^2 + 10^2} \times \sqrt{10^2 + 3^2}} = \frac{90}{\sqrt{136} \times \sqrt{109}} = 0.73$$

The cosine similarity is widely used in text mining for document classification [34]. It restricts values to the range of (0, 1), where a value of 0 indicates that there is no similarity between the documents and a value of 1 indicates that the documents are identical.

Sociolinguistics Characteristics

Talking styles are influenced by the social context of a situation, the speaker, and the audience, such as who can hear us and where we are talking. Styles reveal aspects of our social identity involving who we are, where we come from, and perhaps what

kind of social experience we have [45]. Sociolinguistics focuses on how people use language in their social interactions and studies the relationships between language and society. Sociolinguistics investigates how social structure influences how people talk and how language varieties and patterns of use correlate with social attributes, such as class, sex, and age [45].

Computer code is a system of symbols and rules used to represent instructions to a computer. Coding involves writing a certain line of code to send a message to the computer. It is how humans communicate with machines and computers. Some variations may be found in computer programs, such as spacing (e.g., spaces vs tabs), naming styles (e.g., CamelCase vs. snake_case), commenting, and how a programmer implements certain types of functionality or uses keywords. Some research has found that social variables may influence a programmer's style just as society influences a person's verbal or written communication [74]. When we observe how varied computer programming style is, we can search for the causes of that variability.

2.2.4 Classification

Using the features presented in section 2.2.3, computer programs can be expressed as numerical vectors, making them applicable to classification algorithms. Text classification aims to assign predefined classes to text documents [24]. The problem of text classification is as follows [8]:

- Given a training set of text documents,

$$D = \{d_1, d_2, \dots, d_n\}$$

- each document d_i is labeled with a label l_j from the set.

$$L = \{l_1, l_2, \dots, l_k\}$$

- The task is to find a classification model f where

$$f : D \rightarrow L \quad \text{and} \quad f(d_i) = l_j \tag{17}$$

which can assign the correct class label (author) to unlabeled document d (test instance).

Next, this section summarizes the classification algorithms that have been applied most frequently in previous authorship attribution studies [54, 23]. The classifiers covered are random forest (RF), naïve Bayes (NB) and support vector machine (SVM).

Random Forest

To understand the random forest (RF) model, we must first learn about the decision tree: a graphical depiction of a decision and every potential outcome or result of making that decision. In a decision tree a divide and conquer technique is used as the basic learning strategy (flowchart-like tree structure). Sharma [85] stated that “a decision tree is a structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. The topmost node in the tree is the root node”.

Random forest is a supervised random decision-tree classifier. RF takes the decision-tree concept further by producing a large number of trees. Random forest randomly creates and merges multiple decision trees into one “forest”. The goal is to rely not on a single learning model but rather on a collection of decision models to improve model performance. Random forests differ in how randomness is introduced in the tree-building process. Here, we consider the random forest version as used by Breiman [18]. To classify a new object, each decision tree provides a classification for input data; random forest collects the classifications and chooses the most voted prediction as the result. In Figure 2.8 the input vector (d) is pushed through each

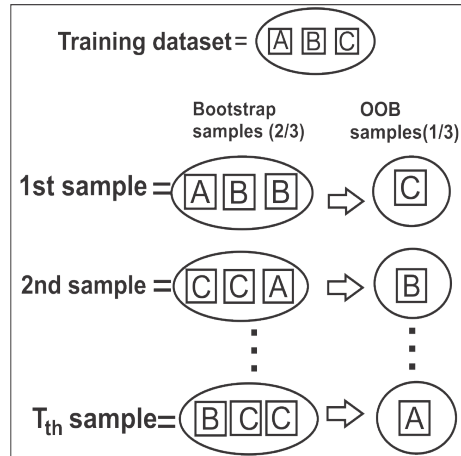


Figure 2.7: RF bootstrap sampling.

of the trees in the forest. Each tree classifies, and each tree votes for the class. The forest chooses the classification with the most votes (over all the trees in the forest).

Each random forest's tree is grown in the following way: if the number of records in the training set is N , then N records are sampled at random but with replacement from the original data as shown in Figure 2.7. This is called a bootstrap sample [59]. Bootstrapping is a type of resampling where large numbers of samples of the same size are repeatedly drawn, with replacement, from a single original sample [49]. In RF bootstrap sampling, drawing a sample of size n with replacement from an original sample of the same size tends to include roughly two-thirds of the original sample, with duplication in the remaining third [48] (see Figure 2.7).

This set will form the training set for growing the tree. If there are M input features, a number $m \ll M$ is selected such that at each node, m features are selected at random out of the M , and the best split on these m is used to split the node. The value of m is held constant during forest growing. Each tree is grown to the largest extent possible until all the leaves belong to a single class [59] such as red squares in Figure 2.8.

When the training set for the current tree is drawn by sampling with replacement, about one-third of the records are left out of the sample. These are called OOB (out-

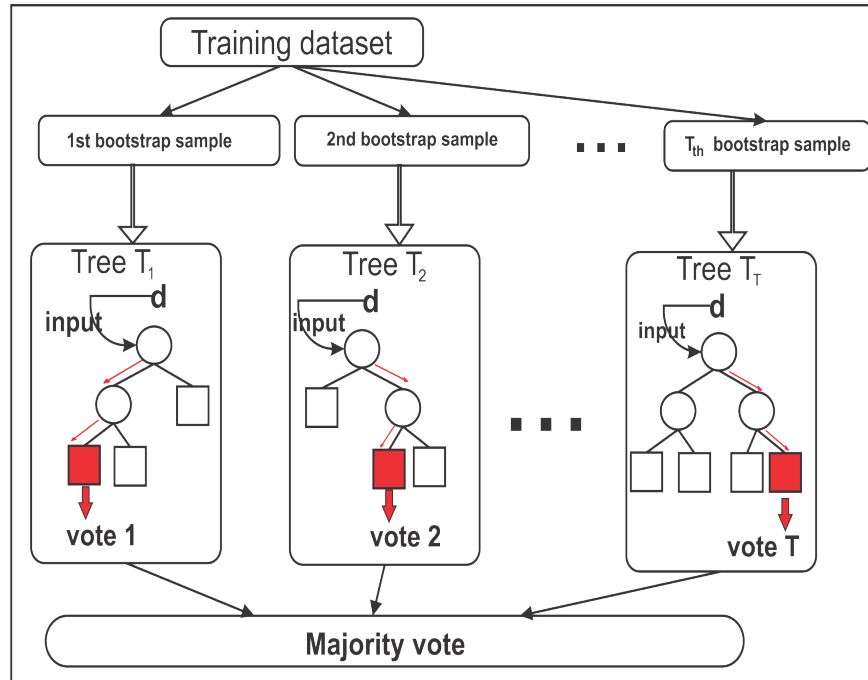


Figure 2.8: Concept of RF classification. In training set, the algorithm creates T multiple bootstrapped samples, and then builds a classification trees from each bootstrapped sample set. For classification, each tree gives a unit vote for the most popular class at each input data. The final label is determined by a majority vote of all trees.

of-bag) samples, and all of these have their labels available. OOB is used to estimate classification accuracy (i.e., for testing purposes) [59] as shown in Figure 2.9.

One useful byproduct of RF is feature importance measures [18]. Liaw [63] implemented an algorithm for calculating feature importance measures in the RF. The algorithm calculates feature importance as the mean decrease in accuracy using the OOB observations, as described below.

First, the accuracy performance on the OOB samples is calculated. Then, the values of the feature in the OOB samples are randomly shuffled, keeping all other features the same. Finally, the change in accuracy on the shuffled samples is measured. The average decrease in accuracy performance across all trees is reported. This importance is calculated explicitly for each attribute in the dataset, allowing attributes to be ranked and compared.

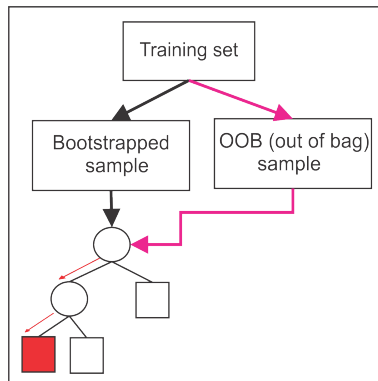


Figure 2.9: OOB data are separated for each bootstrapped sample. The accuracy performance is evaluated by testing the trees with the separated OOB data, and then the performance of the RF is calculated by taking the average of these performance rates.

Naïve Bayes

NB is the most straightforward classifier and is based on Bayes' theorem. NB describes the probability: for document d and class c , our goal is to compute the conditional probability of each class c given a document d , then we use this probability $P(\mathbf{c} \mid \mathbf{d})$ to find the best class. The best class is, out of all classes ($c \in C$), the one that maximizes the probability $P(c \mid d)$. Bayes' theorem describes the probability of an event based on prior knowledge of conditions that might be related to the event.

$$P(\mathbf{c} \mid \mathbf{d}) = \frac{P(d \mid c) P(c)}{P(d)} \quad \text{Bayes' Theorem} \quad (18)$$

- $P(c \mid d)$: Probability of class given a document
- $P(d \mid c)$: Probability of document given a class
- $P(c)$: Probability of class
- $P(d)$: Probability of document

To use Bayes' theorem in a classifier, Equation (a) defines C_M , which is the best class. The goal is to assign document d to the best class.

Using Bayes' rule as shown in Equation (b), the best class C_M can be determined. In (c), we drop the denominator to simplify the equation (whichever class maximizes

Equation (b) will also maximize Equation (c)). In Equation (c), to find the probability of $P(d|c)$, document d should be represented by feature vector (x_1, x_2, \dots, x_n) as shown in Equation (e).

$$C_M = \operatorname{argmax}_{c \in C} P(c|d) \quad (\text{a})$$

$$= \operatorname{argmax}_{c \in C} \frac{P(d|c)P(c)}{P(d)} \quad (\text{b})$$

$$= \operatorname{argmax}_{c \in C} P(d|c)P(c) \quad (\text{c})$$

$$C_M = \operatorname{argmax}_{c \in C} P(d|c)P(c) \quad (\text{c})$$

$$= \operatorname{argmax}_{c \in C} P(x_1, x_2, \dots, x_n|c)P(c) \quad (\text{e})$$

NB assumes that the attributes of any instance of the training-set are conditionally independent of each other as expressed in (f).

$$P(d|c) = P(x_1, x_2, \dots, x_n|c) = P(x_1|c) \cdot P(x_2|c) \cdot P(x_3|c) \cdots P(x_n|c) \quad (\text{f})$$

The final formula to find the best class using NB is presented in Equation 19.

$$C_M = P(x_1|c) \cdot P(x_2|c) \cdot P(x_3|c) \cdots P(x_n|c) \cdot P(c) \quad (19)$$

Note that $P(c)$ is the prior probability of a class. The prior probability of a given target class is the proportion of its occurrence compared with the other target state [75]. For instance, we have 5 documents and there are two possible class labels, class=yes and class=no. 2 of the 5 documents were assigned with a yes class, and 3 were assigned with a no class. The prior probability for class “no” is $P(\text{no}) = \frac{3}{5}$.

Support Vector Machine (SVM)

SVM is a supervised machine learning algorithm [27] based on the separating hyper-plane. This hyper-plane separates the dataset into classes with the maximum margin. Imagine we are given a training set of documents with two known class labels (-1 and $+1$) and two available measurements (attributes) per document. In this specific example, we consider a problem with two classes. The idea behind support vector machine classification is that these measurements can be regarded as a two-dimensional space [31]. Each document is then represented by a data point in this space as expressed with black square or circle (d_1, d_2, d_n) in Figure 2.10. For the documents with two available measurements such as Feature1, Feature2 as shown in Figure 2.10, a line can be drawn to separate between the training documents.

We want to find a separating hyper-plane that separates these points (documents) into the two classes: “the positives” (class “ $+1$ ”) and “the negatives” (class “ -1 ”) (assuming they are linearly separable). But as shown in Figure 2.10 there are many possible decision boundaries which could separate these two classes; which one should be chosen? In SVM the goal is to choose the optimal decision boundary, which is where the distance to the closest negative point is equal to the distance to the closest positive point (see Figure 2.11). The distance between a point and a decision boundary (SVM) can be measured using the techniques described in [27].

Currently, SVM is widely used in text mining for its effectiveness in high dimensional spaces because the learning algorithm is independent of the dimensionality of the feature space [8]. Joachims [52] claimed that text data is an appropriate choice for SVM classification due to the sparse high dimensional nature of the text.

2.2.5 Model Evaluation

The purpose of model evaluation is to determine the best classifier, or model, and predict classifier performance on unlabeled data. To assess a model’s performance,

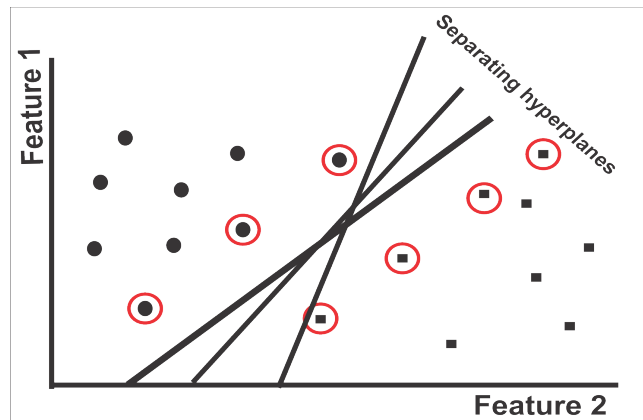


Figure 2.10: Separating hyper-planes, or possible decision boundaries, which can separate two possible classes. Circles indicate documents with “negative” labels and squares represent documents with “positive” labels.

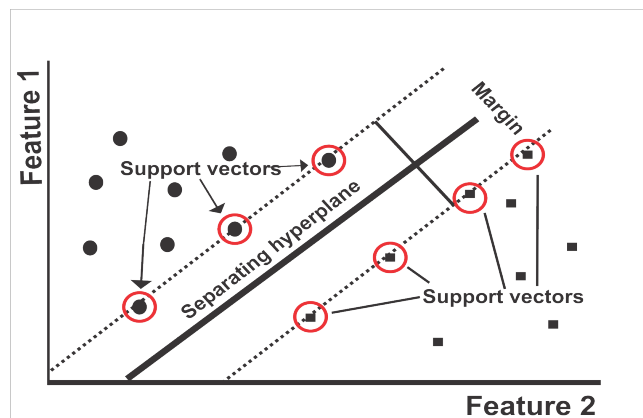


Figure 2.11: Choosing the optimal decision boundary using SVM: the distance to the closest negative (circle) point should be equal to the distance to the closest positive (square) point.

datasets are separated into two types:

1. Training set: used for result learning and making algorithms, and
2. Testing set: used for data validation.

One approach would be to manually to split the data into training and testing sets. However in this process, the presence or absence of a single sample that deviates significantly from the other samples can significantly change outcomes. Another approach is K-fold cross-validation (CV).

Cross-Validation

K-fold cross-validation (CV) is one of the most common approaches for evaluating machine learning models using multiple test sets. CV uses multiple test sets and averages the outcome values, which may give us more precise classification results. CV requires that the data are randomly divided into k equal sized folds (or train and test splits). The first fold is treated as a validation (test) set, and the method is trained on the remaining $k - 1$ folds [49]. A value of $k = 10$ is commonly used in the field of machine learning [49].

Evaluation Metrics

Evaluation metrics measure classifier effectiveness: the ability to make correct classification decisions for the largest possible number of documents. In the following, we will characterize four evaluation measures: accuracy, kappa, precision and recall. We first describe a tool referred to as a confusion matrix. The confusion matrix is a useful tool for analyzing a model's performance and helps us gain insight into the types of errors a model makes. To explain the working principle of a confusion matrix, a table of size $n \times n$ is shown in Figure 2.12, where n represents the number of classes. We assume we have a three class problem ($n = 3$), with classes A , B , and C . The predicted classes are the columns, the actual classes are the rows, and

		Prediction		
		A	B	C
Actual	A	TP_A	E_{AB}	E_{AC}
	B	E_{BA}	TP_B	E_{BC}
	C	E_{CA}	E_{CB}	TP_C

All observations in actual class A

Figure 2.12: Confusion matrix for multiple classes. This example represents a three-class problem with the classes A, B, and C.

the values inside the cells are the integer count of classifications or misclassifications. The diagonal value is the true positive (TP), or data points for which the actual label matches the predicted label. The values in other cells are the error counts. For example (E_{AB}) is the number of instances for which the actual label is class A but the prediction was class B.

By using a confusion matrix, the evaluation metrics of accuracy, recall, and kappa can be calculated as follows:

- Accuracy: sum of correct classifications divided by the total number of classifications.

$$\text{Accuracy} = \frac{TP_A + TP_B + TP_C}{TP_A + TP_B + TP_C + E_{AB} + E_{AC} + E_{BA} + E_{BC} + E_{CA} + E_{CB}} \quad (2.2.1)$$

- Kappa: compares an observed accuracy with an expected accuracy (random accuracy).

$$K = \frac{\text{Accuracy} - \text{Random Accuracy}}{1 - \text{Random Accuracy}} \quad (2.2.2)$$

$$\text{Random Accuracy} = R_A + R_B + R_C$$

$$R_A = \frac{TP_A + E_{BA} + E_{CA}}{\text{total observations}} \times \frac{TP_A + E_{AB} + E_{AC}}{\text{total observations}}$$

$$R_B = \frac{TP_B + E_{AB} + E_{CB}}{\text{total observations}} \times \frac{TP_B + E_{BA} + E_{BC}}{\text{total observations}}$$

$$R_C = \frac{TP_C + E_{AC} + E_{BC}}{\text{total observations}} \times \frac{TP_C + E_{CA} + E_{CB}}{\text{total observations}}$$

$$\text{total observation} = TP_A + TP_B + TP_C + E_{AB} + E_{AC} + E_{BA} + E_{BC} + E_{CA} + E_{CB}$$

- Recall (or sensitivity): the ratio of correctly predicted observations divided by all observations in actual class. Actual class observations are horizontal rows shown in Figure 2.12:

- Recall $A = TP_A / (TP_A + E_{BA} + E_{CA})$

- Recall $B = TP_B / (TP_B + E_{AB} + E_{CB})$

- Recall $C = TP_C / (TP_C + E_{AC} + E_{BC})$

- Precision: is the ratio of correctly predicted observations divided by the total number of observations.

- Precision $A = TP_A / (TP_A + E_{AB} + E_{AC})$

- Precision $B = TP_B / (TP_B + E_{BA} + E_{BC})$

- Precision $C = TP_C / (TP_C + E_{CA} + E_{CB})$

2.2.6 Related Work

Authorship attribution of natural language text documents is a well-explored area but too broad for a comprehensive review in this thesis. Instead, we focus on the specific area of feature extraction to allow us to summarize the computer code authorship attribution literature. Existing studies show that machine learning can be

used to determine the author of a natural language text. Argamon et al. conducted authorship attribution experiments to find writing differences based on the author's writing style. They experimented with online discussion boards [12]. Each document, on average, consisted of between 61 and 167 words. Stylometry techniques were used to identify relevant features from the dataset. Function words, Internet abbreviations, capitalization, word positioning/placement, word length, and line length features were selected to investigate author style. Argamon et al. used machine learning approaches and achieved more than 66% prediction accuracy for the two-class authorship attribution problem (classifying text sample between two authors). In [12] Argamon et al. extended their work to the multi-class problem; however the scores degraded to less than 30% percent accuracy for a twenty-class problem.

Following up, Argamon et al. conducted a comprehensive study to evaluate gender writing style differences in textual documents using stylometry techniques [10]. A collection of 600 writing samples was collected for their experiments, which allowed each author to be represented by multiple documents. Feature extraction techniques were employed to determine the writing style of the male or female programmers. Relevant features including word distribution, usage, and frequencies were selected to investigate gender writing style. Then machine learning approaches were employed to distinguish between male and female written documents. Using stylometric features, they were able to develop a model which could predict author gender with 90% accuracy.

Addressing authorship attribution (stylometry) for natural language text is a well-known problem. However, far fewer works are dedicated to authorship identification in non-natural languages, such as computer code. Code authorship attribution (code stylometry) presents a challenge due to the inflexibility of written code expressions. One of the earliest publications in the area of code stylometry was produced by Krsul and Spaford [57]. This study focuses on the classification of programmer style to

find characteristics of coding style to identify a program's author. Krsul and Spaford were the first to introduce sixty stylistic characteristics of author writing, including indentation of C statements; use of conditional compilation; choice of while, for, do loops; and the number of lines in a function. The dataset used in their study was composed of eighty-eight C programs from 29 students, staff, and faculty members (3 writing samples per author). Krsul and Spaford used LNKnet [64] software to implement the machine learning methods [57]. Two classification models were applied to the given dataset: multi-layer perception neural network and nearest neighbor. To evaluate the model, the authors used the 4-fold cross-validation method. Krsul and Spafford achieved 100% accuracy with the multi-layer perception neural network. However, Burrows believed that their methodology was questionable, as they used 4-fold cross-validation with approximately three samples of work per author. There were three samples per author, so if each fold contained one sample from a particular individual, there had to be a fold with no sample from that individual. Thus, this may cause the classifier to function incorrectly [19].

Burrows' work presented an approach based on n-gram features [20]. His work was inspired by the success of using n-grams in text-authorship identification [1]. Using n-grams made the approach programming-language independent, since it is based on low-level information (character or word) [33]. In their experiment, they used a dataset consisting of 1640 C programs written by 100 programmers. They examined the most appropriate n-gram size for the problem domain (n-grams length $\in \{1, 2, 3, \dots\}$) and also explored the effect of increasing the number of authors. Burrows' approach did not achieve high accuracy as the number of authors increased. Thus in [21] Burrows extended the work on code authorship identification by including stylometric features along with the n-gram representation of the programs. The stylometric features were white space, operator, and keywords. This combination slightly improved the earlier results.

The most relevant recent work in this area was done by Islam in 2015 and Dauber in 2018, applying the same approach [23, 29]. In 2015, Islam achieved a 94% attribution rate with 1600 authors. They collected the C++ language data from the international annual programming competition (Code Jam). To find the features that would represent a programmer’s coding style, they used two feature groups: lexical features, such as the number of the code line and spaces, and syntactic features. For their syntactic features, the authors used the abstract syntax tree (a tree representation of the structure of a piece of code) of the computer code using Fuzzy AST parser [26]. Then they applied the random forest classification to develop a model. Dauber (2018) [29] also used a similar approach to computer code author identification of Git repositories. They collected contiguous program fragments using Git Blame (<https://git-scm.com/docs/git-blame>) and organized possible metrics (features) using an analytical approach similar to the approach Islam employed in their 2015 work [29]. Dauber obtained their dataset of 104 programmers from some unknown subset of the repositories. Dauber also used 10-fold cross-validation to evaluate their models, as in Islam [23]. Using this approach Dauber [29] reported an attribution rate of around 50% accuracy from a 104-programmer dataset with at least 150 samples of at least 1 line in length. Dauber found that the approach from [23] is much less effective on this data than on Google Code Jam data. However Dauber found that they could improve classification rates from around 50% accuracy to about 90% accuracy by grouping samples in batches of 30.

Information about the influence of the author’s sociolinguistic characteristic features on computer programs was determined by a group of studies by Naz, Rafee and Rice [74, 83]. Naz et al. investigated gender differences in programming language use. Their dataset was composed of 100 C++ programming assignments from male and female programmers at the University of Lethbridge. Naz et al. present a technique for computer code attribution described in [74]. To convert the collected C++ programs

into the numeric form, they used a list of 50 features including operators, keywords, loops, and comments. They used Information Gain approach to select a small set of the most useful features. As a result, only seven features were identified that played a role in the differentiation between male-written and female-written texts. The machine-learning tool WEKA [42] was used to construct supervised learning models. Cross-validation and hold-out techniques were used to evaluate the performance of classification models. In the classification step, the K^* (nearest neighbour) model performed best with an accuracy of 71%.

Related research made further inroads into the practical categorization of computer programs based on author gender and region. In 2017 Rafee and Rice demonstrated an approach for classifying programming assignments [83]. They collected 160 C++ programs from a university in Canada and a university in Bangladesh. In this work, they used 15 from the IEEE Standard for Software Productivity [25], including total number of code lines, total number of blank lines, and comment lines. They evaluated seven classification models through 10-fold cross-validation [83]. They achieved an accuracy rate of 92% using the random forest [63] classification technique. Rafee also performed another experiment where they included the programmer's region as a feature. They used a hybrid of the region and software features in a feature set to categorize the programs based on author gender. They used the same seven classification models and evaluation technique in this experiment. The NB classification model using 16 features (15 IEEE features plus region as social factor) achieved 83% classification accuracy. However, this was reduced to 70% in the absence of the extra feature (author region).

The majority of code stylometry studies to date have used different feature categories, including content-based features such as the number of code lines, word order, and keywords, or non-content-based features such as region (explained in Section 2.3). Nevertheless, there is no consensus on the best set of features to achieve recognition

of author style. The majority of researchers used the subsets of features from different groups. For instance, the use of feature sets containing software features such as keywords and n-grams has been shown more effective for stylometry [23, 69]. Thus, using subsets of features from software metrics and n-grams categories is better than using features from only one of these categories. This section's key finding is that using subsets of features from different categories for example n-grams, lexical diversity, and similarity is generally more effective at identifying the most likely author of a document than subsets restricted to single categories of features.

Chapter 3

Methodology

In this chapter, we discuss the significant steps we used to move towards identifying authors of computer programs. We discuss data collection, transformation, and feature extraction and briefly describe our programming environment.

3.1 Data Collection

The two collections we employed in this thesis are labelled Coll-F and Coll-G. In this section, we discuss the sources of the data for each collection and the methods for building those collections.

The first source was Codeforces.com, a contest-based programming website that hosts daily contests. The daily contest problem sets consist of five problems for contestants to solve in a two-hour period. The website stores each programmer’s biographical information and their submissions. Data from this source were labeled Coll-F. Data collection for Coll-F proceeded according to procedures established in Tasnim’s work [89]. We built a collection of C++ computer programs with the author’s identity and sociolinguistic characteristic features, such as gender and region.

Dataset Coll-F contained C++ computer programs from 60 programmers. For each programmer, we stored 10 to 12 programs in the dataset, for a total of 669 programs. We used 4 attributes: handle, gender, region, and source code. Handle, region and gender are the user’s biographical information. Handle is a user’s unique login name for Codeforces (chosen by the user) and is an identifier for each programmer

Region \ Gender	Female	Male
Asia	10	10
Europe	10	10
US	10	10
Total	30	30
Total Programmers: 60		

Figure 3.1: Information about data distribution in our datasets.

(label). The region represents the region of the programmer and, the source code field contains the computer program. We attempted to classify computer programs into 60 classes: the author’s handle. We prepared our dataset in CSV format. We selected equal numbers of programs written by males and females and from three regions: Asia, Europe and the U.S. (shown in Figure 3.1).

The second source was Github (accessible at <https://github.com/>), the largest online code sharing or storage website on the internet. Data from this source were labeled Coll-G. We collected data on users, as well as source code, from GitHub and added it to the database according to procedures established in Alam’s work [7].

The Coll-G dataset contained programs from 60 programmers. Each programmer had nine programs stored in the dataset, for a total of 540 programs. In Coll-G, each program is a record in a CSV file and contains country code, gender, source code, and programmer identity. The region, gender, and programmer’s identity form the user’s information. Programmer identity (ID) is unique to each programmer (label) and is selected by the user. The source code field contains the computer program. We attempted to classify computer programs into 60 classes: the author’s identity. As for dataset Coll-F, we selected equal numbers of programs written by males and females and from three regions: Asia, Europe and U.S. (shown in Figure 3.1).

In each of the following subsections we provide an example dataset as shown in Figure 3.2 consisting of four texts written by two different authors to demonstrate

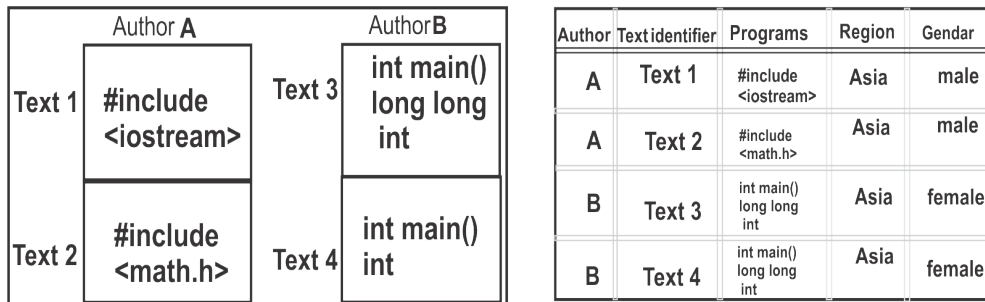


Figure 3.2: An example dataset of four texts written by two authors.

how our methodology works.

3.2 Document Representation and Preparation

To determine the authorship of computer programs, we needed to transform the text-based datasets into a numerical representation. In our research, the input feature values are numeric and class labels are nominal. There are 60 class labels: the author’s identity. All of the collected C++ programs are considered text documents. The numerical representation of a given text document is produced in two steps: tokenization and TF-IDF.

In the tokenization step (see Section 2.2.2), we used the `quanteda` package in the R ecosystem [55] to perform document representation and preparation. The `quanteda` package has a “`tokens()`” function for tokenizing text documents, which divides each program line into identified tokens. Each document (C++ program) is divided into tokens, separated by white space. With tokenization complete, we constructed a data frame where each row represented a document, each column represented a distinct token, and each cell represented a count of the tokens in the document.

We used the dataset of four texts written by two authors (discussed in Section 3.1) in order to demonstrate the tokenization approach. The tokenized programs are shown in Figure 3.3.

In the second step, in each document we calculated the TF-IDF values for individual tokens in each document in order to convert the original dataset into a numerical

	#include	iostream	int	main	long	cout	()	<>	Math.h
Text 1	1	1	0	0	0	0	0	1	0
Text 2	1	0	0	0	0	0	0	1	1
Text 3	0	0	2	1	2	0	1	0	0
Text 4	0	0	1	1	0	1	1	0	0

Figure 3.3: Tokenization process for a dataset with four documents written by two authors. The tokenization technique divides each program line into identified tokens. Each row represents a text document, each column represents a distinct token, and each cell represents a count of the tokens in the document.

representation and evaluate how relevant a token is to a particular document within the collection of documents. Token occurrences over the entire dataset were represented using term frequency and inverse document frequency (TF-IDF), as explained in 2.2.2. In this way, we transformed the text-based dataset into a numerical representation. In each dataset we added a new column representing the class label (the author’s identity) to each computer program row in the experimental dataset. After data preparation, Coll-F had 2,500 tokens (columns) and 669 rows (representing computer programs), and Coll-G had 17,890 tokens and 540 rows.

3.3 Metrics/Features Extraction

As seen in Chapter 2, a set of features is needed to classify text document data using classification algorithms. In authorship identification these features help quantify an author’s writing style. In this research we used five groups of features for authorship identification. Four of them—software metrics, n-grams, cosine similarity, and lexical diversity metrics—are content-based groups of features. Sociolinguistics characteristics—the fifth—are non-content-based features. The following subsections explain how we extracted these feature groups from the raw text.

3.3.1 Software Metrics

In this study, the first group of content-based features is software metrics. Naz [74] used machine learning to assess how sociolinguistic characteristics, including author

gender, impact computer programs. We selected 16 keywords based on [74] for our research work, as listed in Table 3.1.

In other research, Rafee [83] also used machine learning to assess how sociolinguistic characteristics, including author gender and region, impact computer programs. In our work, we also selected two features based on [83]: lines of code and the number of characters in the code. In total we developed a set of 18 software metrics as listed in Table 3.1.

Table 3.1: List of 18 software metrics.

C++ vocabulary	Features
Keywords	include, std, vector, double, char, const, void, bool, int, return, string, for, else, while, if, cout
Total lines	Count of lines (includes number of computer program lines and blank lines)
Total number of characters	Count of characters (includes number of computer program characters)

To extract keywords from the thousands of tokens in a data set, we used an R-based package [82] and the “select()” function. The critical keywords were chosen from the prepared datasets. To count the number of lines and the number of characters, we used the “readLines()” and “length()” functions from the base package.

For example, to develop a set of software metrics for the dataset of four texts written by two authors, we used tokenization and then we applied TF-IDF to prepare a numerical representation. Next, we extracted software metrics such as keywords to represent the documents as a vector of features (software metrics). Figure 3.4 illustrates the software metrics chosen for this example.

Our representation of the text documents so far has been single terms such as

Text identifier	Label	#include	iostream	cout	int	main
text1	A	0.0752575	0.150515	0.0000000	0.00000000	0.00000000
text2	A	0.0752575	0.000000	0.0000000	0.00000000	0.00000000
text3	B	0.0000000	0.000000	0.0000000	0.07525750	0.03762875
text4	B	0.0000000	0.000000	0.1003433	0.05017167	0.05017167

Figure 3.4: Example showing how 5 keywords in TF-IDF format represent the dataset of four texts with software metrics.

keywords, also known as unigrams. Not surprisingly, there are also bi-grams, tri-grams, and n-grams.

3.3.2 N-grams

An n-gram is a sequence of n consecutive words. The set of n-grams, $n \in \{1, 2, \dots, n\}$ that can be generated for a given document is the result of moving a window of n words along the text, one word at a time. In this way, the number of occurrences of each n-gram is counted. In our work, we used an n-gram model to represent the context of the document and generate features to classify the document. We used different values of $n \in \{3, 6, 10\}$ to generate and extract the n-gram features. We explored whether shorter or longer n-gram patterns are suitable for authorship attribution. To add n-grams to our features, we used the same preprocessing approach as for the software metrics. It is important to note that comments and punctuation are not removed from programming samples in datasets, so they are analyzed as well. We used the “tokens_ngrams()” function from the `quanteda` package to represent the data using n-gram features. Next we used TF-IDF to represent each n-gram feature.

We provide an example in order to clarify the n-gram approach. In the dataset of four texts written by two authors in Section 3.1, the 3-gram approach was employed to identify groups of 3 tokens that repeat throughout the sample. Each row represents a text document, each column represents a 3-gram term and each cell represents a count of the terms in the document, as shown in Figure 3.5. We then applied TF-IDF to that data frame of 3-gram features’ frequency. There were 12 columns and 4 rows

	#include < iostream	< iostream >	#include < math.h	< math.h >	int main (main () cout) cout ;	() long) long long	long long int	long int ;
text1	1	1	0	0	0	0	0	0	0	0	0	0
text2	0	0	1	1	0	0	0	0	0	0	0	0
text3	0	0	0	0	1	1	0	0	1	1	1	1
text4	0	0	0	0	1	1	1	1	0	0	0	0

Figure 3.5: 3-gram extraction process for our example dataset with four documents belonging to two authors. Each row represents a text document, each column represents a 3-gram (3 sequential tokens), and each cell represents a count of the sequence (terms) in the document.

	#include < iostream	< iostream >	#include < math.h	< math.h >	int main (main () cout) cout ;	() long) long long	long long int	long int ;
text1	0.3	0.3	0	0	0	0	0	0	0	0	0	0
text2	0	0	0.3	0.3	0	0	0	0	0	0	0	0
text3	0	0	0	0	0.05	0.05	0	0	0.1	0.1	0.1	0.1
text4	0	0	0	0	0.07	0.07	0.15	0.15	0	0	0	0

Figure 3.6: Data frame in the example dataset showing 3-gram frequencies after applying TF-IDF.

	V1	V2	V3	Label
Text1	0	-1	0.0000000	A
Text2	-1	0	0.0000000	A
Text3	0	0	-0.4472136	B
Text4	0	0	-0.8944272	B

Figure 3.7: Example dataset represented by 3-gram frequencies after applying TF-IDF and reducing the feature set using SVD.

in the numerical representation using 3-gram as reported in Figure 3.6.

One challenge of using n-gram features is learning from high-dimensional data. When documents contain a large number of words, there is a high computational burden for the learning process. Thus, it is best to perform a dimension reduction technique to reduce the text feature size and avoid large feature space dimension. We used the “irlba” package to apply the truncated SVD projection method in this research as described in Chapter 2, reducing the feature set to a smaller subset of the combination of the most valuable features. We represented each document with 20 features obtained from the truncated SVD.

To illustrate our approach, we use an example dataset of 4 texts from 2 authors. We extracted the 3-gram features’ frequency and applied the TF-IDF, as shown in Figure 3.7. Then we applied SVD, which reduced the data frame dimension from 12 columns (terms) and 4 rows to 3 columns (V1, V2, V3) and 4 rows as shown in Figure 3.7.

3.3.3 Similarity Measurement

Another metric used in our work is cosine similarity. This section presents the steps we used to calculate the similarity between documents. The process to compute the similarity between two texts t_1 and t_2 is as follows [6]:

1. Tokenize t_1 and t_2 and combine both outputs into a joint list J .
2. Calculate the feature vectors v_{t_1} and v_{t_2} for each of t_1 and t_2 using TF-IDF.
3. Obtain a similarity measure between the two texts by calculating the cosine similarity between the two vectors. We used a “cosine()” function from the “lsa” package in R.

The general hypothesis behind the similarity feature is as follows: if all of author X’s programs can be collected, it may be possible to say, on average, that any writ-

Label	Text identifier	CTTR	TTR	C	R	V_M	Similarity
A	text1	1	1.0	1	1.4	0	0
A	text2	1	1.0	1	1.4	0	0
B	text3	0.9	0.6	0.6	1.3	0.16	0.7
B	text4	1.2	1.0	1	1.7	0	0.7

Figure 3.8: The similarity and lexical diversity features of our example dataset of four texts written by two authors. We compared the similarity value between text1 and text2 written by author A and then calculated the value of similarity between text3 and text4 written by author B. As we were considering cosine distance between two texts, the similarity entries were repeated. A value of 0 indicates that the documents do not have any similarity and a value of 1 indicates that they are identical.

ten programs by author X will have higher average similarities with all of their other programs than with those belonging to other authors. Thus, we can provide that similarity measurement as a feature for classification, and it may improve the performance of the resulting model.

As an example, we calculated similarity values between 4 texts written by 2 authors as shown in Figure 3.8. As we demonstrated in section 3.3, we can calculate the cosine similarity between two documents by considering just a few keywords, for example, the frequency of “NEW” and “OLD” in Doc1 and Doc2, shown in Figure 2.6. For this example, we only considered part of the keywords (“int”, “main”, “iostream”, “math.h”), but not all of the tokens, to determine the similarity ratio between pairs of documents. This was to keep the example simple while illustrating the effect of variation similarity rate. This feature represents the similarity between text1 and text2 written by author A, and the similarity between text3 and text4 written by author B in the example dataset. Because we are considering cosine distance between two texts (combining two rows) the similarity entries are repeated as shown in Figure 3.8. A value of 0 indicates that the documents do not have any similarity and a value of 1 indicates that they are identical.

3.3.4 Lexical Diversity

Metrics in this group are considered to be indicators of lexical diversity within the computer programs written by each programmer. In our work we used 9 metrics implemented in the `quanteda` package [55]: TTR, CTTR, R Guiraud’s Root, Herdan’s C, Uber Index, Yule’s K, Yule’s I, V_m , and Simpson’s D. We applied the metrics to programs produced by the programmer. Prior to computing the metrics, we parsed all texts as a sequence of words (tokenization). Then for calculating these lexical diversity measurements of the documents we used the “`lexdiv()`” function. The five properties of lexical diversity were measured in our example dataset as shown in Figure 3.8.

3.3.5 Sociolinguistic Characteristic

The final category of features that we collected was sociolinguistic characteristics. We chose the region and gender of the programmer as features, while classifying the programs according to author’s identity. The sociolinguistic characteristic information is not a feature of a program. However, some research [83] claims that inclusion of programmer sociolinguistic characteristic features in the feature set increases model performance. Our datasets contained programs from 3 regions and two gender categories.

In the example dataset gender and region were attached to the original dataset as can be seen in Figure 3.2. Overall we used machine learning techniques to analyze five categories of features. In order to visualize all the pieces of information that we collected, Figure 3.9 illustrates all the data using the example dataset that included four texts. In this figure each of the five categories of features is shaded with a different colour.

Texts identifier	Label	Software Metrics					3-gram			Lexical diversity					Similarity		Sociolinguistic Characteristic	
		#include	iostream	cout	int	main	V1	V2	V3	CTTR	TTR	C	R	V _m	Similarity	Gender	Region	
Text1	A	0.0752575	0.150515	0.0000000	0.00000000	0.00000000	0	-1	0.0000000	1.0	1.0	1.0	1.4	0	0	Male	Asia	
Text2	A	0.0752575	0.0000000	0.0000000	0.00000000	0.00000000	-1	0	0.0000000	1.0	1.0	1.0	1.4	0	0	Male	Asia	
Text3	B	0.0000000	0.0000000	0.0000000	0.07525750	0.03762875	0	0	-0.4472136	0.9	0.6	0.6	1.3	0.16	0.7	Female	Asia	
Text4	B	0.0000000	0.0000000	0.1003433	0.05017167	0.05017167	0	0	-0.8944272	1.2	1.0	1.0	1.7	0	0.7	Female	Asia	

Figure 3.9: A visualization of five categories of features we extracted using machine learning techniques for a dataset that contained four texts from two authors. Each category was coloured a different shade.

3.4 Programming Environment

We ran all experiments on a Dell laptop with an Intel Core i7 processor, 8 GB of RAM, and 500 GB of hard disk space. The computer ran on the Windows 10 operating system. We used the R programming language to classify the programs and analyze the features of our research. R is free and open-source and is widely used in the field of machine learning. We chose R because it has a large community and lots of support [46]. Also, the wide variety of packages is one of its essential qualities: there are around 12,000 packages available in CRAN, an open-source R repository.

We repeated a 10-fold cross-validation technique five times as the basis for our model-building process. Cross-validation is a technique that gets maximum use out of our training data and allows us to create ideally representative estimates of how the model will perform on new data. Cross-validation is powerful, but its downside is that it requires more processing and, therefore, more time. We used a “caret” package to create the folds for the 10-fold cross-validation. We repeated the process five times to provide more valid estimates. To reduce total execution time we used multicore training in parallel techniques. Almost all computers have multicore processors, and as long as the computations do not need to communicate they can be implemented across multiple cores and executed in parallel, reducing computation time. However, for R—or any other language—it is hard to determine what can be parallelized. By

default, the operating system will allocate each R session to a single core. R provides several packages for parallel computing to run the program on multiple cores and we used `doSNOW` package in our work.

Each of the five 10-fold cross-validations is independent of the others. If a computer with multiple processors or cores is available, the computations could be spread across these processors to increase the computational efficiency. The “`caret`” package leverages one of the parallel processing frameworks in R to do just this [58]. To run the parallel processing technique, a separate function “`register`” is used to specify the number of cores to use. In this work we used three cores because the number of logical core in my system is four, and one core was saved for usual tasks in computer and three of them dedicated to parallel computing in R programming.

3.5 Summary

This chapter defined the data collections, the procedure to transform the text-based dataset into a numerical dataset and the feature extraction methods. These were the necessary steps to classify computer programs based on the author’s identity. In the next chapter, we will discuss the experiments and the results.

Chapter 4

Experiments and Results

In this chapter, we discuss the structure of our six experiments, review the results of these experiments, and conclude with a summary of the results.

4.1 Experiments

Our experiments assume that the collected C++ programs are a form of text documents. In chapters 2 and 3, we explained the procedures we used to transform the text document dataset into a numerical dataset. We applied several classification algorithms to the text documents to develop models that identify the author from a set of suspects. A vector of numeric feature values was provided as input to the classification models, which, in turn, provided outputs in the form of class labels, the author’s identity. In authorship attribution and text mining, there are several popular classification algorithms available [54, 23]. We selected three algorithms that are popular in code authorship attribution [83, 74, 89]: random forest, support vector machine, and naïve Bayes. We applied these machine learning algorithms to train and test our models, and then we evaluated the models using metrics such as accuracy, Kappa, precision and recall.

In this research, we performed six experiments using two datasets. Figure 4.1 gives an overview of these six experiments: We collected the first dataset—consisting of 540 C++ programs from 60 programmers—from the code repository GitHub.com (Coll-G). We collected the second dataset—consisting of 669 C++ programs from 60

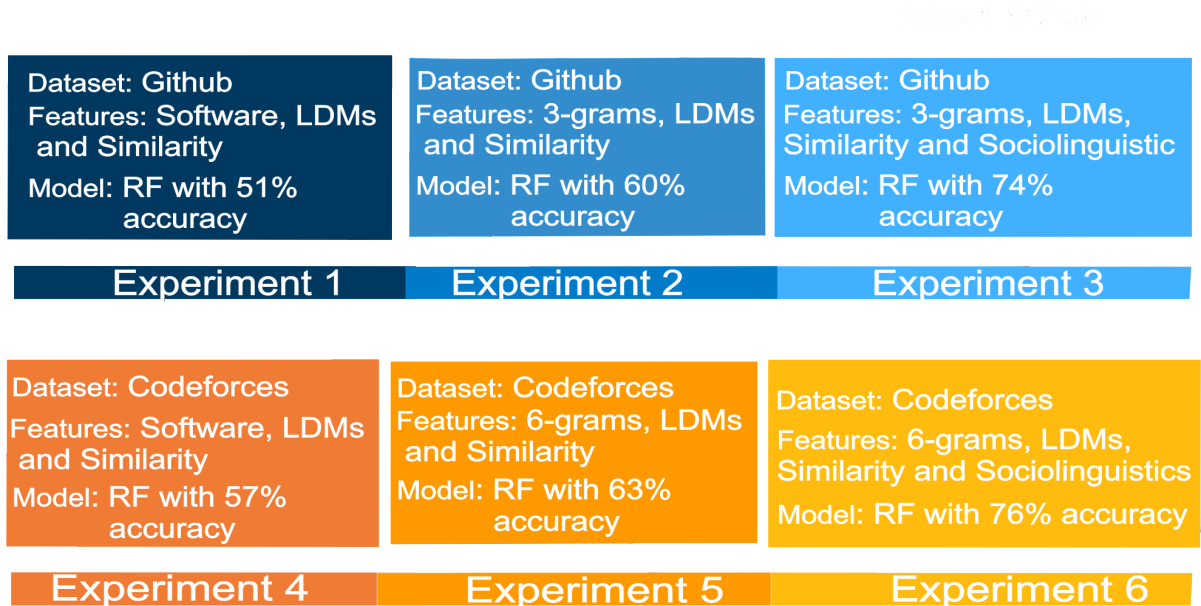


Figure 4.1: An overview of the six experiments in this thesis.

programmers—from the freelance contest website Codeforces.com (Coll-F).

We used the GitHub dataset (Coll-G) in experiments 1, 2, and 3 and the Codeforces dataset (Coll-F) in experiments 4, 5, and 6. We used different feature sets in each experiment, but we applied the same three classification algorithms in each experiment. We used software metrics, lexical diversity and similarity features in experiments 1 and 4, switched software metrics with appropriate n-gram size features in experiments 2 and 5, and added a set of 2 sociolinguistic characteristic features in experiments 3 and 6. In each of the six experiments, we used three classification models to identify the author/programmer of the text/program. We assumed that each text/program is assigned to only one author, represented by the class (label) of a given text. In all experiments, we used the 10-fold cross-validation technique and repeated it five times to evaluate the model. We collected the results in terms of the evaluation metrics. We used the R language programming to carry out our experiments. The R library package for these machine learning algorithms was “caret”. The description of these six experiments follows in the next subsections.

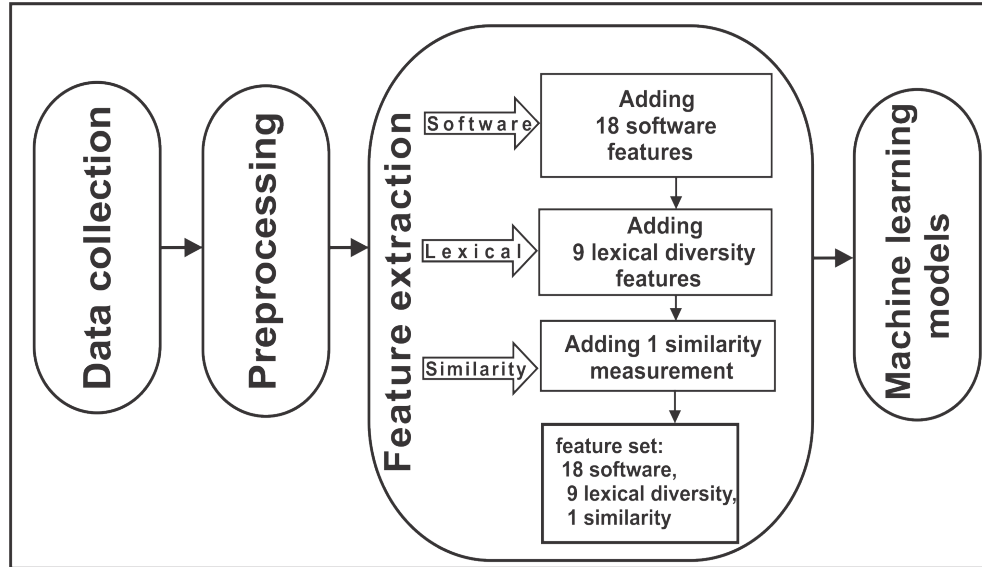


Figure 4.2: Steps for experiments 1 and 4.

4.1.1 Experiment 1

In experiment 1, our goal was to classify a computer program based on programmer identity, mostly using software metrics. We conducted the experiment on dataset Coll-G with 540 computer programs belonging to 60 programmers. We used 28 content-based features, including 18 software metrics, 9 lexical diversity, and 1 similarity features. In this research the class labels are programmer’s identity. The steps of experiment 1 are shown in Figure 4.2. We applied three machine learning algorithms to the dataset to build the models: naïve Bayes (NB), random forest (RF), and support vector machine (SVM). To evaluate the model, we applied the 10-fold cross-validation technique repeated five times. We collected the results in terms of the evaluation metrics. The performance results are reported in section 4.2.1.

4.1.2 Experiment 2

In experiment 2, we again analyzed computer programs from dataset Coll-G to determine authorship. However this experiment examined three feature sets to find the appropriate length for n-gram features. Each feature set included 30 features: 20 optimized features generated by the n-gram ($n \in \{3, 6, 10\}$), 9 lexical diversity

features, and 1 similarity feature. Steps of this experiment are shown in Figure 4.3.

We extracted three types of n-gram features (3-gram, 6-gram, and 10-gram) from the computer programs data to explore how n-gram length impacts our experiment (a review of n-gram length is given in section 3.2). However, we found that we obtained high dimensional data frames due to a high number of features extracted through the n-gram approach. For the 3-gram approach (a sequence of 3 words) in dataset Coll-G, there were 136433 features; for the 6-gram approach (a sequence of 6 words) there were 250592 features; and for the 10-gram approach (a sequence of 10 words) 303940 features. We applied SVD to each data frame in order to transfer the features into the optimized format as explained in section 3.2. Singular value decomposition or SVD is a dimensionality reduction approach that can be used as a projection method where data with m-columns (features) is projected into a subspace with fewer columns, whilst preserving the essence of the original data. The optimized SVD format of each n-gram feature set contained 20 features. For example, the data frame with 540 programs and 136433 features based on the 3-gram approach optimized to 540 programs and 20 features. Thus, in experiment 2, we built three feature sets, as listed below, to find the best n-gram length to create a numeric vector of the dataset:

1. 20 features from 3-gram, 9 lexical diversity features, and 1 similarity feature (feature set A);
2. 20 features from 6-gram, 9 lexical diversity features, and 1 similarity feature (feature set B); and
3. 20 features from 10-gram, 9 lexical diversity features, and 1 similarity feature (feature set C).

In this experiment, we tested each feature set to the three machine learning models (SVM, NB, RF) using 10-fold cross-validation technique repeated five times. The model performances are reported in Section 4.2.2.

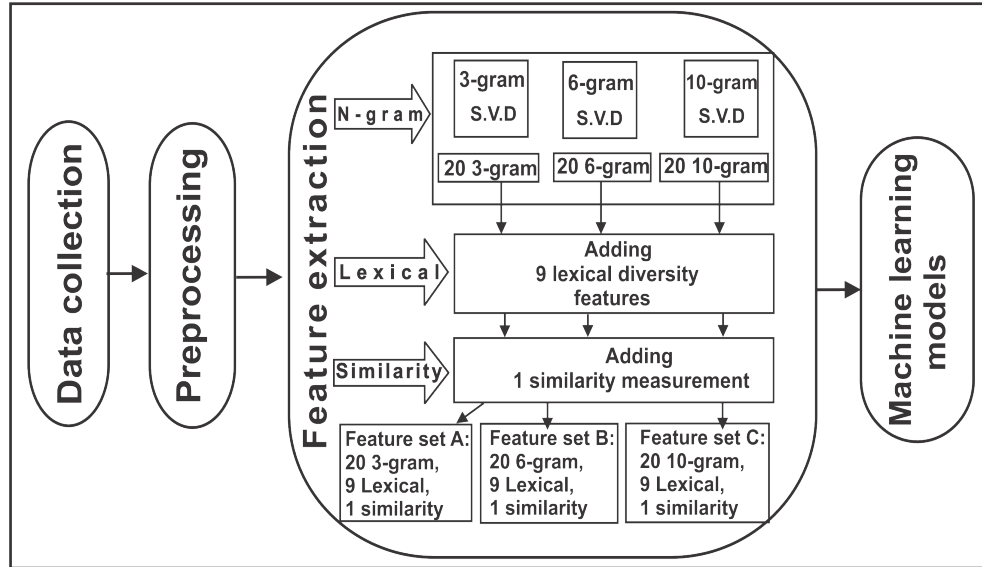


Figure 4.3: Steps for experiments 2 and 5.

4.1.3 Experiment 3

In experiment 3 we again used the Coll-G dataset, consisting of 540 C++ programs written by programmers from three regions and two genders. To improve model performance we added a set of sociolinguistic characteristic features to the feature set from the experiment with the best performing model from the previous two experiments: experiment 2 with feature set A. In this experiment, the added sociolinguistic characteristic features are the programmer’s gender and region, which may help identify a programmer’s identity [83]. We retrieved numeric feature values from the second experiment, and we added 2 non-content-based features from the collected dataset. Each computer program was thus represented by 32 feature values: 20 3-gram features, 9 lexical diversity features, 1 similarity feature, and 2 sociolinguistic characteristic features. We used the 10-fold cross-validation technique repeated five times to evaluate all of the developed models. The steps for experiment 3 are shown in Figure 4.4.

4.1.4 Experiment 4

In experiment 4, we analyzed the Coll-F dataset to determine the programmer’s identity for each program. Coll-F consists of 669 C++ programs written by 60 unique programmers and taken from a competitive programming website. In experiment 4, we categorized the C++ programs according to programmer’s identity using 28 content-based features: 18 software metrics, 9 lexical diversity features, and 1 similarity feature. As in the previous experiments we built the three classification models using NB, RF and SVM classification algorithms. We used the 10-fold cross-validation technique, repeated five times, to evaluate all the developed models. The steps for this experiment are shown in Figure 4.2.

4.1.5 Experiment 5

In experiment 5 our goal was to classify 669 computer programs with 30 features based on the the programmer’s identity. This experiment was also run on dataset Coll-F. The steps of this experiment are the same as in experiment 2, and are shown in Figure 4.3. To generate the 20 n-gram features, we explored three lengths of n-gram: 3-gram, 6-gram, and 10-gram. We then built three feature sets, as in experiment 2—A, B, and C—to find the best n-gram length. In this experiment, the learning step was constructing the same three models (SVM, NB, RF) using 10-fold cross-validation technique repeated five times for each feature set.

4.1.6 Experiment 6

In the final experiment, we classified the collected C++ programs from the Coll-F dataset using content-based features and sociolinguistic characteristic features. We added a set of sociolinguistic characteristic features (gender, and region) to the set of content-based features from experiment 5. The steps of this experiment are the same as those in experiment 3, as shown in Figure 4.4, but using the Coll-F dataset. We built the three classification models using NB, RF, and SVM classification algorithms.

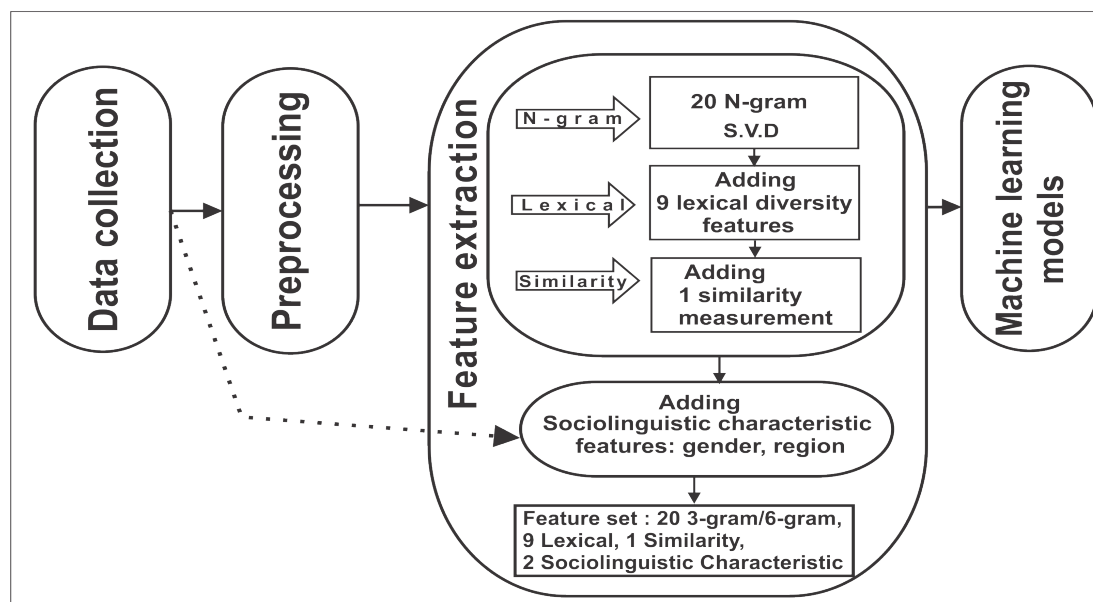


Figure 4.4: Steps for experiments 3 and 6.


We again used the 10-fold cross-validation technique repeated five times to evaluate all of the developed models.

4.2 Results

The goal of all six experiments was to categorize computer programs based on programmer’s identity. We conducted experiments 1, 2, and 3 on dataset Coll-G and experiments 4, 5, and 6 on dataset Coll-F. In all experiments, class labels were the programmer’s identity. We evaluated the performance of the machine learning models in all experiments using a 10-fold cross-validation technique, repeated five times. We used accuracy, Kappa, precision, and recall to evaluate the model.

Note that in our work, the authorship attribution approach categorized programs into one of 60 classes, which is called a multi-class classification model. Multi-class classification is the task of classifying programs into one of three or more classes. The performance of a multi-class model can be evaluated by averaging the performance of the individual classes as shown in Figure 4.5. Furthermore, when the number of

	Class: author 1	Class: author 2	Class: author 3
Accuracy	96%	85%	100%
Precision	93%	99%	100%
Recall	80%	72%	100%
⋮			
	Class: author 4	Class: author 5	Class: author 6
Accuracy	84%	93%	85%
Precision	66%	100%	89%
Recall	64%	87%	71%
⋮			
Accuracy			
Precision		• • •	
Recall			
⋮			
	Class: author 58	Class: author 59	Class: author 60
Accuracy	89%	95%	77%
Precision	88%	82%	58%
Recall	80%	92%	56%
⋮			



Overall Model Performance	
Accuracy	85%
Precision	99%
Recall	72%
⋮	

Figure 4.5: Confusion matrix for multi-class machine learning model in our work.

classes increases, performance may decrease as explained in Section 2.1. Table 4.1 compares multiple contributions in authorship attribution among 7 to 229 authors and ranges of accuracy from (53% to 100%). Most of the research has found that models are less accurate when there are more authors (labels). In our work, there are 60 authors in each dataset. We used different feature sets to improve model performance in each experiment. The highest accuracy rate for classifying programs into 60 classes in each dataset is given in Table 4.1. As discussed further on, this level of accuracy was achieved in Experiments 3 and 6.

Table 4.1: Comparison of previous contributions [54].

Reference	Authors	Classification	Accuracy	Features	Sample
Macdonell et al.	7	Case-based reasoning	88%	26	351
Frantzeskou et al.	8	Nearest neighbor	100%	1500	107
Elenbogen and Seliya	12	Decision tree	74.7%	8	83
Krsul and Spafford	29	Discriminant analysis	73%	50	88
Ding and Samadzadeh	46	Nearest neighbor	55%	56	225
Islam et al.	229	Random forest	53.9%	120000	2061
Ours (Coll-G)	60	Random forest	74%	32	540
Ours (Coll-F)	60	Random forest	76%	32	669

4.2.1 Experiment 1 Results

In experiment 1, we used 28 content-based features and three classification algorithms to build our three models. As shown in Table 4.2, the RF model performed best with 51.76% accuracy. That means this model could correctly classify half of the programs from dataset Coll-G according to programmer identity. The NB model classified programs with 25.09% accuracy, while the SVM model achieved accuracy of 42.15%. The RF model also achieved the highest Kappa rate: 50.86%. Next was the SVM model, which achieved 41.09% Kappa rate. Precision and recall for the RF were 46.07% and 48.15%, respectively. The NB model was in last position among the three classification models in terms of correctly classifying programmer’s identity of the programs. Precision and recall for the NB model were 28.93% and 21.74%, respectively.

Table 4.2: Experiment 1 results.

Models	Accuracy	Kappa	Precision	Recall
RF model	51.76%	50.86%	46.07%	48.15 %
SVM model	42.15%	41.09%	39.51%	40.13 %
NB model	25.09%	23.56%	28.93%	21.74 %

4.2.2 Experiment 2 Results

In experiment 2, we turned our attention to word orders using n-gram features. We explored three feature sets (A, B, C) to choose an appropriate n-gram size. Each feature set consisted of 20 n-gram, 9 lexical diversity, and 1 similarity feature. For our first representation of Coll-G in experiment 2, we transformed the computer programs into vectors using the feature set A (3-grams). As shown in Table 4.3, the RF model performed best with 59.86% accuracy. That means this model classified 59.86% of the programs of our dataset correctly according to programmer identity. Classification accuracy of the SVM model was 41.04%. The NB model achieved almost 44% accuracy. The RF model achieved the highest Kappa rate scoring 59.10%. The SVM and NB models achieved 39.95% and 42.13% Kappa, respectively.

Table 4.3: Results from experiment 2 with feature set A.

Models	Accuracy	Kappa	Precision	Recall
RF model	59.86%	59.10%	55.22%	55.35 %
SVM model	41.04%	39.95%	38.02%	37.10 %
NB model	43.17%	42.13%	42.4%	39.42 %

For our second representation of Coll-G in experiment 2, we transformed the computer programs into vectors using feature set B (6-grams). As shown in Table 4.4, the RF model performed best, classifying 55.72% of the programs of our dataset correctly according to programmer identity. The NB model achieved accuracy rates of about 33%. The SVM model was slightly better with accuracy rates around 35%. The RF model also achieved the highest Kappa rate, scoring 54.89%. Next was the SVM model, which achieved 35% Kappa. Precision and recall for the RF model were both approximately 52%.

For our third representation of Coll-G in experiment 2, we transformed the computer programs into vectors using feature set C (10-grams), as shown in Table 4.5. The RF model classified 53.23% of the programs of our dataset correctly according

to programmer identity (53.23% accuracy). The classification accuracy of the SVM model was 30.2%. The NB model had an accuracy rate of less than 10%. The RF model also achieved the highest Kappa rate, scoring 52.36%. Next was classification via the SVM model, which achieved 28.96% Kappa.

Overall, we found that the RF model using feature set A (with 3-gram features) performed better than feature sets B and C.

Table 4.4: Results from experiment 2 with feature set B.

Models	Accuracy	Kappa	Precision	Recall
RF model	55.72%	54.89%	52.23%	52.49 %
SVM model	34.67%	33.48%	31.39%	30.69 %
NB model	32.93%	31.73%	33%	29.54 %

Table 4.5: Results from experiment 2 with feature set C.

Models	Accuracy	Kappa	Precision	Recall
RF model	53.23%	52.36%	47.62%	48.17 %
SVM model	30.2%	28.96%	27.10%	26.59 %
NB model	6%	4%	7%	5 %

4.2.3 Experiment 3 Results

In experiment 3, again our goal was to classify the programs according to programmer identity. We expanded the feature set from 30 to 32. Features in experiment 3 included 20 3-gram, 9 lexical diversity, 1 similarity, and 2 sociolinguistic characteristics (region and gender). The 3-gram features were chosen because that feature set performed the best in Experiment 2. We used three classification algorithms to build the three models. As shown in Table 4.6, the RF model performed best (74.78% accuracy), classifying correctly 74.78% of the programs in our dataset according to programmer identity. Classification accuracy for the SVM model was 59.11%, and for the NB model, 45.44%. The RF model also achieved the highest Kappa rate: 74.31%. Next was classification via the SVM model, which achieved 58.34% Kappa.

It is important to note that we carried out three different experiments using dataset Coll-G; experiments 1, 2, and 3. The RF model in experiment 3, with the two added non-content-based features improved significantly over all the other models we used in our previous experiments 1 and 2.

Table 4.6: Experiment 3 results.

Models	Accuracy	Kappa	Precision	Recall
RF model	74.78%	74.31%	73.59%	71.70 %
SVM model	59.11%	58.34%	60.02%	56.19%
NB model	45.44%	44.44%	43.83%	41.63 %

Table 4.7: Experiment 4 results.

Models	Accuracy	Kappa	Precision	Recall
RF model	57.10%	56.32%	55.11%	55.88 %
SVM model	51.62%	50.74%	52.57%	51.55 %
NB model	22.75%	21.34%	19.21%	22.54 %

4.2.4 Experiment 4 Results

In experiment 4, we switched to dataset Coll-F. The 669 programs were classified into 60 classes using 28 content-based metrics: 18 software metrics, 9 lexical diversity features, and 1 similarity feature. The performance of the three machine learning models using the 10-fold cross-validation technique repeated five times is given in Table 4.7. RF and SVM performed well for classifying programs based on programmer identity with accuracy of 57.10% and 51.62%, respectively. This means that each model was able to classify slightly more than 50% of the programs of our dataset Coll-F correctly according to the identity of the programmer. NB’s accuracy of 22.75% was lowest among all models. In addition, RF led based on Kappa scores (56.32%). The NB model had an inferior Kappa score to other models (21.34%).

4.2.5 Experiment 5 Results

Table 4.8: Results from experiment 5 with feature set A (3-grams).

Models	Accuracy	Kappa	Precision	Recall
RF model	60.54%	59.81%	59.20%	59.69 %
SVM model	55.96%	55.16%	57%	55.41%
NB model	50.4%	49.49%	53%	49.75 %

Table 4.9: Results from experiment 5 with feature set B (6-grams).

Models	Accuracy	Kappa	Precision	Recall
RF model	62.27%	61.58%	60.79%	61.70 %
SVM model	51.45%	50.55%	51.79%	50.37%
NB model	52.23%	51.34%	54.71%	51.28 %

In experiment 5 we classified dataset Coll-F using three feature sets: A, B, and C as explained in Section 4.1.5. We used 30 features in each feature set: 20 n-gram features with a specific length $n \in \{3, 6, \text{or } 10\}$, 9 lexical diversity, and 1 similarity features.

With feature set A, the RF model performed better than other models, with a classification accuracy of 60.54%. That means that the model would classify more than 60% of the programs of our dataset correctly according to programmer identity. The model developed using SVM classifier had an accuracy of 55.96%. The NB model had the lowest accuracy: 50.40%. RF model had the highest Kappa score (59.81%) while NB model had the lowest (49.49%) as shown in Table 4.8.

Table 4.9. reports the performance of three machine learning models using feature set B. The RF and NB performed better than the SVM model in this representation, with an accuracy of 62.27% and 52.23%, respectively. The RF model also had the top Kappa score (61.58%). The accuracy of the SVM model was lower than other models: 51.45%. The Kappa score of SVM was 50.55%, which was the lowest among all of the models.

Performance of the three machine learning models using feature set C are shown in Table 4.10. The RF model surpassed all other learning models in terms of accuracy, precision, Kappa, and recall. It achieved 58.51% accuracy, while the other models' accuracy was between 40% and 50%. This indicates that RF model was able to classify 58.51% of the programs of our dataset Coll-F correctly, according to the identity of the programmer.

In experiment 5, for our three feature sets A, B, and C, we used n-grams of $n \in \{3, 6, 10\}$ respectively, to investigate the optimal length of n (explained in 4.1.5). Results showed that the machine learning models that had the best results used 6-grams in feature set B.

Table 4.10: Results from experiment 5 with feature set C (10-grams).

Models	Accuracy	Kappa	Precision	Recall
RF model	58.51%	57.75%	56.37%	57.90 %
SVM model	42.39%	41.32%	41.94%	41.17%
NB model	52.41%	51.52%	53.55%	51.54%

Table 4.11: Experiment 6 results.

Models	Accuracy	Kappa	Precision	Recall
RF model	76.47%	76.04%	74.73%	74.80%
SVM model	66.37%	65.75%	66.91%	65.34%
NB model	55.36%	54.54%	57.13%	54.18 %

4.2.6 Experiment 6 Results

Experiment 6 again used dataset Coll-F. We retrieved a total of 30 content-based programming features same as for Experiment 3; 20 6-gram, 9 lexical diversity, 1 similarity. The 6-gram features were chosen because that feature set performed the best in Experiment 5. Also, we added 2 sociolinguistic characteristics to the feature set. We then classified the programs using the three machine learning models. We used the 10-fold cross-validation technique repeated five times to evaluate the machine

learning models. The results of experiment 6 are in Table 4.11. The accuracy of most of the machine learning classifiers increased significantly in experiment 6, compared to experiment 5. The increased accuracy of the machine learning classifiers was an outcome of adding 2 attributes from non-content-based features. The accuracy of RF and SVM classifiers increased the most, at around 14% and 15%, respectively. The RF classification accuracy was 76.57%, meaning this model would classify 76.57% of the programs of our dataset correctly based on the programmer identity. Of the three classifiers, NB saw a slight increase in accuracy (about 3%). The RF model also achieved the highest Kappa rate of 76.04%. Next was the SVM model, which achieved 65% Kappa and 66% accuracy rate. Precision and recall for the RF model were both around 75% while they were considerably lower for the NB model: 57% and 54% respectively.

It is important to note that we carried out three different experiments on dataset Coll-F; experiments 4, 5, and 6. The RF model in experiment 6 improved significantly over all the other models we applied in our previous experiments 4 and 5.

Chapter 5

Discussion

The goal of this work was author identification. We performed 6 experiments in total, 3 on each of our two datasets (Coll-G and Coll-F). In experiments 1, 2, 4, and 5, we comparatively analyzed the different groups of content-based features using three learning models, while experiments 3 and 6 evaluated how adding non-content-based features to content-based features impacts learning-model performances.

5.1 Performance Discussion of Experiments 1, 2, and 3 on the Github Dataset.

In experiments 1, 2, and 3, our goal was to identify authorship on computer programs on the Coll-G/Github dataset. In experiment 1, we extracted 18 software metrics, 9 lexical diversity features, and 1 similarity feature to transform the written computer programs into vectors. We used three learning models for classification. With an overall accuracy score of 50%, the RF model performed the best among the three learning models in experiment 1 as shown with EX1 in Figure 5.2. This percentage was the average of 60 individual class accuracy as shown in Figure 5.3. That means in average that model was able to correctly classify 50% of programs of each programmer in our dataset according to the identity of the programmer. In experiment 1, this was the best performance. However, the accuracy rate needed to be improved to identify authors more correctly.

For experiment 2, we replaced the 18 software metrics with 20 n-gram features

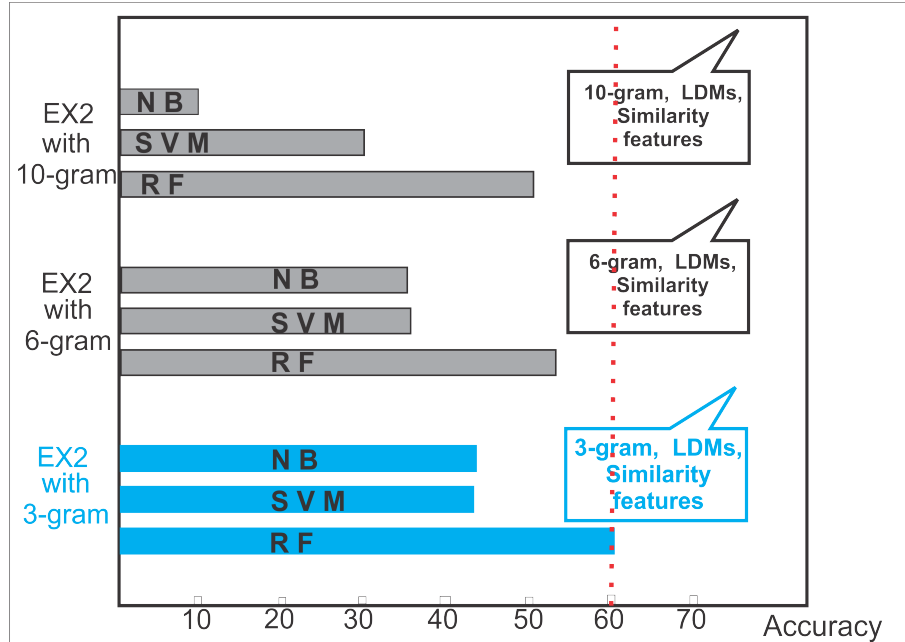


Figure 5.1: Experiment 2 using $n \in \{3, 6, 10\}$) to establish optimal settings for the choice of n-gram length. Three classification models were employed, and their performance was demonstrated with each feature set.

to see whether that improved model performance. To implement the n-gram approach, we needed to establish optimal settings for the choice of n-gram size. Some studies have suggested that choosing an efficient n-gram size could affect authorship attribution and improve model performance [19]. We examined three n-gram lengths $n \in \{3, 6, 10\}$ for dataset Coll-G to test whether shorter or longer string patterns (terms) were good indicators of authorship.

We used three feature sets with different n-gram lengths, feature sets A, B, and C. Three classification models were employed, and their performance was analyzed with each feature set. The accuracy results showed that for all three models, best results were achieved with the 3-gram features. This suggests that smaller n-gram sizes (3-gram) are better for program authorship attribution in dataset Coll-G (Github), as illustrated in Figure 5.1. We also observed the lowest classification performance in machine learning models when we used the largest n-gram size in experiment 2. We hypothesize that perhaps, when programmers write a computer program for a GitHub

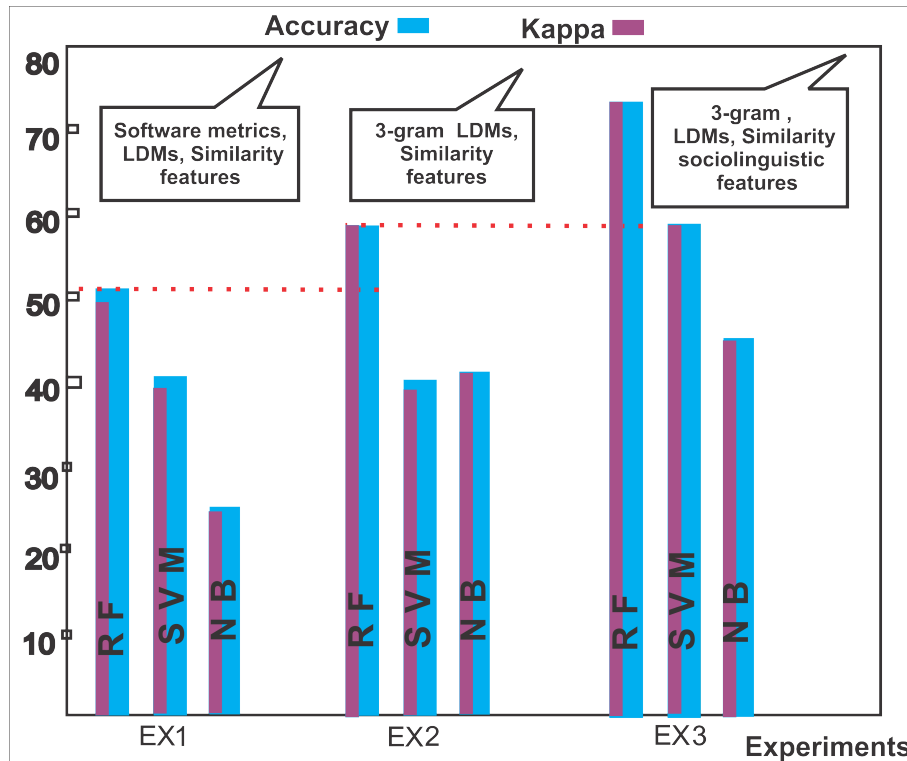


Figure 5.2: Comparative results of experiments 1,2, and 3. The results of experiment 1, experiment 2, and experiment 3 respectively appear in EX1, EX2, and EX3.

repository, they do not have the pressure of time and results, so they write the code in their own style and reveal their individuality. For instance, they are free to name variables or punctuate as they like. Therefore, smaller n-grams may be better able to capture these unique patterns. N-grams of length 2 or 3 have also been successful in identifying the author of a given text in natural language [39].

We have also demonstrated that replacing software metrics with 3-grams in the feature set improved the accuracy in all models except the SVM model. Comparative results are shown in Figure 5.2. One reason for the performance consistency in the SVM results is the property of its classification algorithm to be able to learn independent of the dimensionality of the feature space. The SVM measures the complexity of hypotheses based on the margin with which features separate the data, not the number of features [52] (See Section 2.3.1).

In experiment 2, with the use of 3-gram and lexical diversity features and similarity

5. PERFORMANCE DISCUSSION OF EXPERIMENTS 1, 2, AND 3 ON THE GITHUB DATASET.

Experiment 1: 28 features with RF model	Labels	Class: author 1	Class: author 2	Class: author 3	• • •	Class: author 60
	Accuracy	66%	89%	56%	• • •	60%
Accuracy of Experiment 1: 50%						
Experiment 2: 30 features with RF model	Labels	Class: author 1	Class: author 2	Class: author 3	• • •	Class: author 60
	Accuracy	49%	88%	59%	• • •	91%
Accuracy of Experiment 2: 59%						
Experiment 3: 32 features with RF model	Labels	Class: author 1	Class: author 2	Class: author 3	• • •	Class: author 60
	Accuracy	72%	87%	92%	• • •	97%
Accuracy of Experiment 3: 74.78%						

Figure 5.3: Each of the three experiments evaluated the accuracy performance of the RF model by averaging individual accuracy scores of 60 classes.

feature, the performance of the RF model showed a significant improvement, with around 60% accuracy and Kappa rate. Using only content-based features, this was the highest percentage of classification we achieved for dataset Coll-G as shown in Figure 5.2.

In line with experiment 2, to improve the model’s performance in experiment 3, we added a set of two sociolinguistic characteristics—region and gender—to the content-based feature set in experiment 2. In experiment 3, the average performance of all machine learning models improved. The RF model achieved the highest accuracy and Kappa: 74%. Figure 5.2 shows that the RF model performed better in each of the experiments, which is due to the underlying concept of the RF classification algorithm. The RF classification algorithm is an ensemble of decision trees. This classifier combines the power of several decision-tree classifiers, and the final decision is the majority class. The RF algorithm is widely used in text mining and has been demonstrated to be efficient for dealing with high-dimensional text data [54].

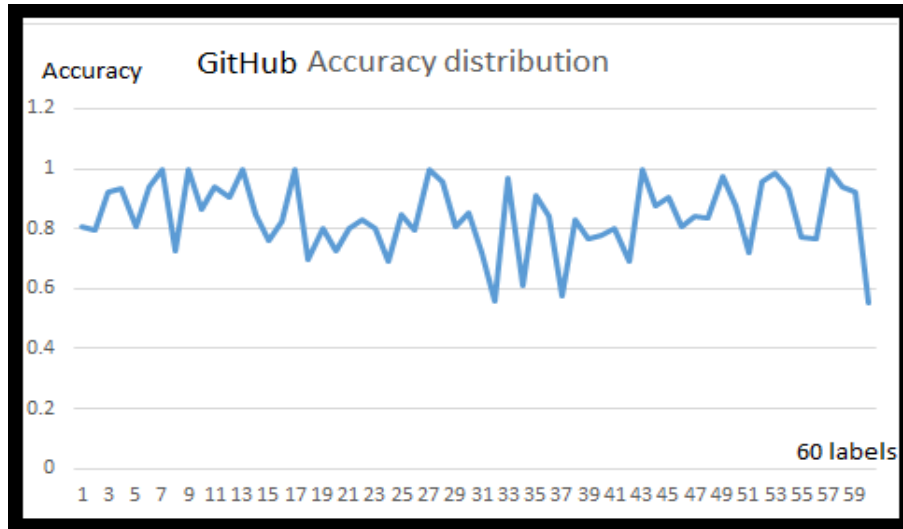


Figure 5.4: Accuracy distribution by class for the GitHub dataset.

5.1.1 Accuracy distribution

For the GitHub dataset, the overall classification accuracy is 74.5%. There are 8 peaks in the line chart shown in Figure 5.4 that indicate 100% accuracy, but there are also several other classes with accuracy between 99% and 80%. There are a few classes with accuracy from 60% to 80%; however, there are no classes where the accuracy is less than 50%.

5.1.2 Gender-based Analysis

Table 5.1: The RF label distribution of individual accuracy from experiment 1, 2, 3 grouped by gender.

Accuracy	Accuracy > 80%		Accuracy < 80%	
	Female	Male	Female	Male
RF from Experiment 1	10	12	20	18
RF from Experiment 2	11	16	19	14
RF from Experiment 3	11	17	19	13

Because RF models performed best in the three experiments using the Coll-G dataset, we examined the label distribution in these RF models. In this work, we

grouped the programmers based on their individual class accuracy levels into two groups based on gender, then examined the distribution of labels within each group. (See Figure 5.3)

As Figure 5.5 illustrates, in all experiments the number of male programmers for which the model achieved individual accuracy of over 80% was higher than the number of female programmers. This suggests that the models were able to identify male-authored programs more accurately. In the accuracy range of less than 80%, the number of female authors was higher as shown in Table 5.1. We hypothesize that male programmers were more predictable in non-competitive situations such as programming for the Github repository. Gilligan [35] claimed women may display different behavior in different contexts, whilst men tend to display behavior that is less context-sensitive and more rule-based. In another study, Wood [92] claimed that women attempt to apply their own rules, whereas men prefer to follow predefined rules. This suggests that maybe male programmers tend to follow specific rules and use a more consistent style in writing different computer programs while female programmers change their style more frequently. These habits could be a possible reason we were unable to classify female-written programs with a better prediction accuracy in non-competitive situations.

Leming’s research may also explain why we were not able to identify women-authored programs as accurately as male-authored programs in dataset Coll-G [50]. He ran experimental tests in academic space under two conditions (high and low risk) in a undergraduate college ($N = 153$). In a high-risk situation, there were pressures from time and proctors, and in low risk, there was no pressure. His results showed that in low-risk situations, women cheated significantly more than men [50]. This could also be one of the reasons we have been able to accurately label fewer female-authored programs. It is possible that female programmers in some cases may have cheated in non-competitive situations, which caused misclassification and mislead the

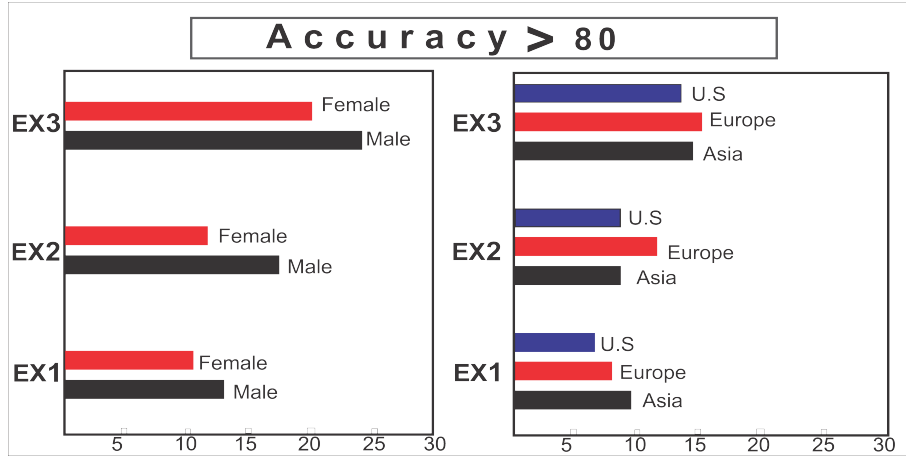


Figure 5.5: Label distribution of individual accuracy over 80% when grouped by gender and region. For experiments 1, 2, and 3 (respectively) results are labeled EX1, EX2, and EX3.

machine learning process.

Table 5.2: The RF label distribution of individual accuracy from experiment 1, 2, 3 grouped by region.

Experiments	Accuracy > 80%			Accuracy < 80%		
	Asia	Europe	U.S.	Asia	Europe	U.S.
RF from Experiment 1	9	7	6	11	13	14
RF from Experiment 2	8	11	8	12	9	12
RF from Experiment 3	14	15	13	6	5	7

5.1.3 Region-based Analysis

In terms of region, when the individual class accuracy was more than 80%, the number of U.S. programmers was less than those in the other two regions, as shown in Figure 5.5. The inaccurate prediction performance in this region as shown in Table 5.2 may be due to population variety. According to the United Nations [15], the United States has the highest number of immigrants from different regions in Asia and Europe. Similar to the overall immigrant population, college-educated immigrants are concentrated in the U.S [77]. We hypothesize that this variety may affect our author identification research.

For author identification, we used content-based features like keywords as well as non-content-based features like region. Alam [7] indicates that sometimes there is an association between the features of programs and programmers' regions. For example, Asian programmers were found to use more comparison operators than American programmers. There is a possibility to misclassify someone from Asia with Asian style (such as using more comparative operators) who migrated to the US and is now programming in the US. In this case the program features we extracted from the programmer's texts and the region of the programmer can conflict with each other, and in some cases, this may be misleading the learning model. In other words, there is a higher probability that we have wrong information about the authors' region in the US and as this information is used for classification, inaccurate information can result in a lower model accuracy rate.

5.2 Performance Discussion of Experiments 4, 5, and 6 on the Codeforce Dataset.

For our next three experiments on dataset Coll-F/Codeforce we used content-based features for experiments 4 and 5, and combined content-based and non-content-based features for experiment 6. The RF model had the highest accuracy rates in all three experiments: 57% in experiment 4, 62% in experiment 5, and 76% in experiment 6 as shown in Figure 5.6.

In experiment 4, we transformed the computer programs into numerical vectors using software metrics, lexical diversity features, and similarity feature. We applied three machine learning algorithms to classify 699 computer programs into 60 classes. Except for the NB, all models had an overall accuracy of over 50% in experiment 4 as shown in Figure 5.8. This means that each model (except NB) correctly classified just over half of the programs of the programmer according to their identity. Since this percentage was not satisfactory for our work, we continued to investigate different

5. PERFORMANCE DISCUSSION OF EXPERIMENTS 4, 5, AND 6 ON THE CODEFORCE DATASET.

Experiment 4 28 features with RF model	Labels	Class: author 1	Class: author 2	Class: author 3	• • •	Class: author 60
	Accuracy	84%	59%	90%	• • •	61%
	Accuracy of Experiment 4: 57.10%					
Experiment 5: 30 features with RF model	Labels	Class: author 1	Class: author 2	Class: author 3	• • •	Class: author 60
	Accuracy	88%	68%	97%	• • •	61%
	Accuracy of Experiment 5: 62.28%					
Experiment 6: 32 features with RF model	Labels	Class: author 1	Class: author 2	Class: author 3	• • •	Class: author 60
	Accuracy	99%	81%	97%	• • •	68%
	Accuracy of Experiment 6: 76.47%					

Figure 5.6: Each of the three experiments 4,5 and 6 evaluated the accuracy performance of the RF model by averaging individual accuracy scores of 60 classes.

content-based features.

In experiment 5, we replaced the software metrics with 20 n-gram features. We examined three n-gram sizes ($n \in \{3, 6, 10\}$) using three different feature sets A, B, and C. Three classification models were employed, and their performance was analyzed with each feature set. The accuracy results showed that the best results were achieved with the 6-gram features as shown in Figure 5.7.

In the experiment 5 (dataset Coll-F), the n-gram size of 6 proved to be the most efficient. Perhaps the underlying reason for getting good results with 6-gram features lies in the structure of programming competition. Competitive programming resources offer guidelines to programmers to use more prepared code blocks or templates [61]. They believe that creating algorithms from scratch is not as efficient because templates or standard library functions are already available. Templates are most commonly used during online competitions where speed is extremely important [61]. We hypothesized that programmers under the pressure of competition may tend to use more prepared blocks of code instead of writing code from scratch and using their creativity; this could be why a bigger n-gram may act better as features, and it could reveal programmer identity with higher performance. This would mean

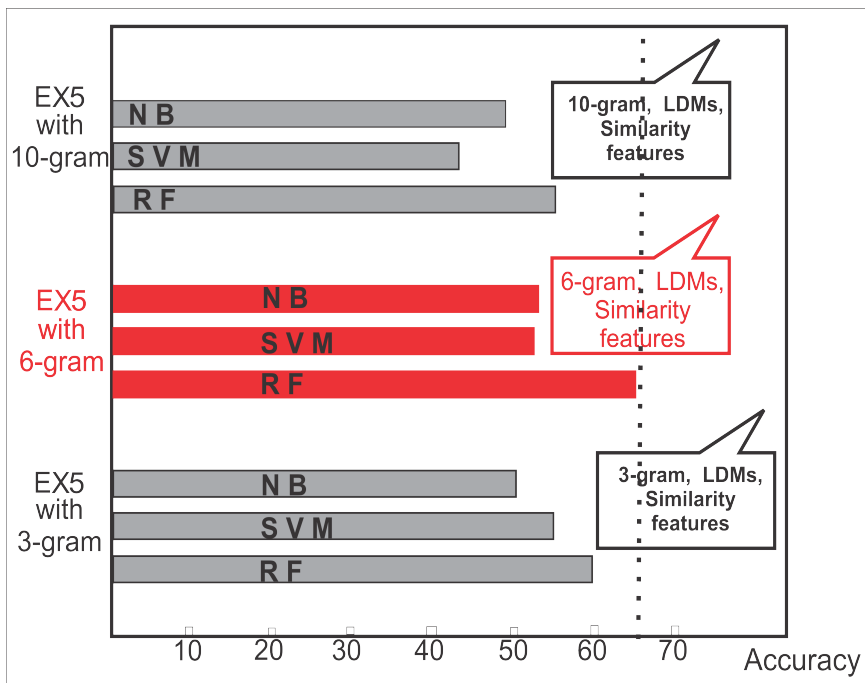


Figure 5.7: Experiment 5 using $n \in \{3, 6, 10\}$ to establish optimal settings for the choice of n-gram length. Three classification models were employed, and their performance was demonstrated with each feature set.

that larger n-grams are beneficial in the capturing patterns of the prepared blocks of code used by each programmer in their programming styles. Alternatively, larger n-grams may allow for the model to discover patterns in the number of special libraries, comments, variables programmers may use in competition.

To improve model performance from the last experiment, we added author gender and region to the feature set and re-applied the machine learning models with a feature set of 33 features: 20 6-gram features, 9 lexical diversity features, 1 similarity feature, and 2 sociolinguistic features. As we can expect, adding non-content-based features improved the performance of the machine learning models with performance increasing 10% compared to experiment 5. Results are shown in Figure 5.8. The RF had the highest rate of accuracy, with around 76% accuracy. All of the models had more than 50% accuracy.

5. PERFORMANCE DISCUSSION OF EXPERIMENTS 4, 5, AND 6 ON THE CODEFORCE DATASET.

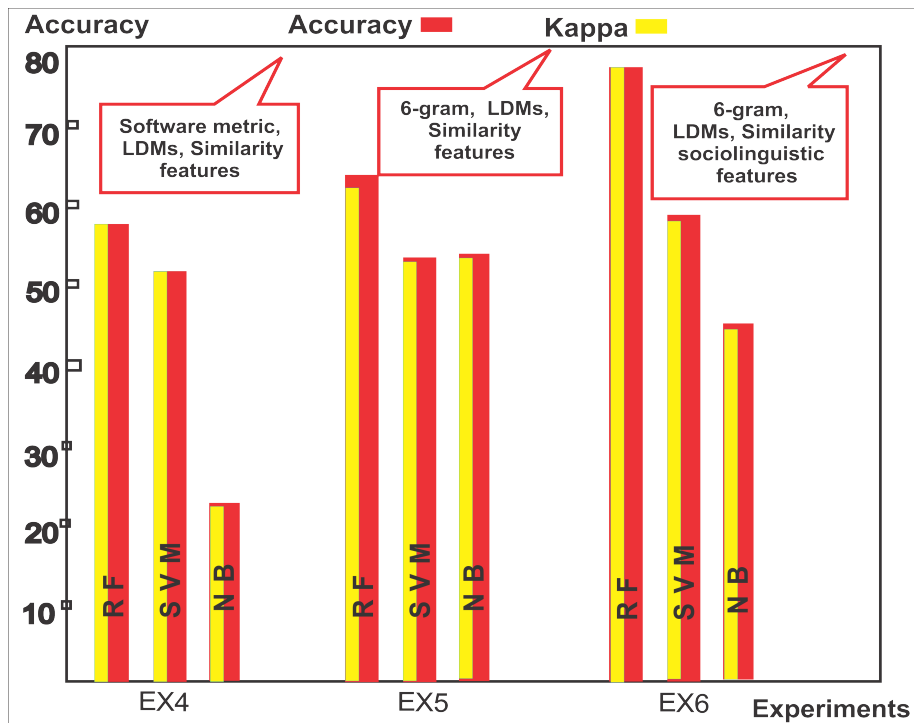


Figure 5.8: Comparative results of experiment 4 (EX4), experiment 5 (EX5), and experiment 6 (EX6).

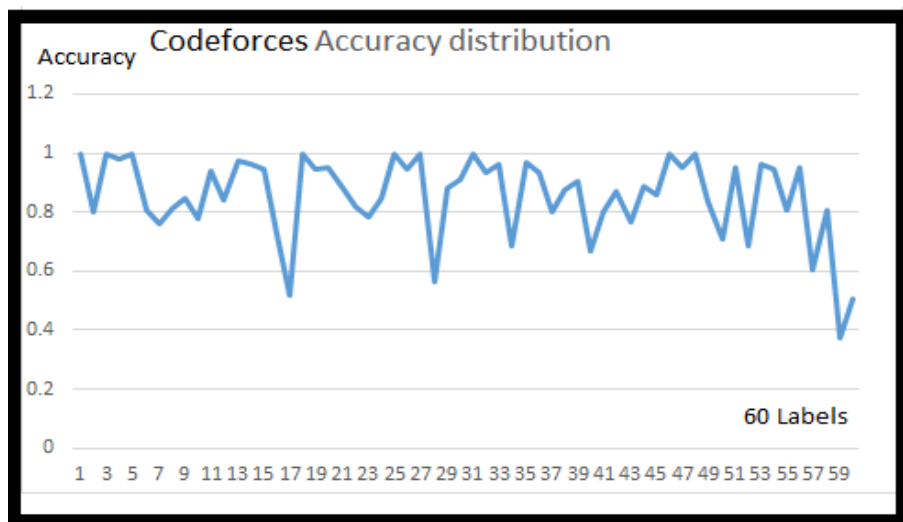


Figure 5.9: Accuracy distribution by class for the Codeforces dataset.

5.2.1 Accuracy distribution

The codeforce dataset has an overall classification accuracy of 76%. The line chart shown in Figure 5.9 has 11 peaks that represent 100% accuracy. Approximately 19 classes have a classification accuracy between 99% and 80%. As in the GitHub dataset, several classes have accuracies ranging from 80% to 50%. A few classes in this experiment have accuracy rates of less than 50%.

5.2.2 Gender-based Analysis

As shown in Figure 5.10 we grouped the programmers based on their individual class accuracy levels into two groups based on gender, then examined the distribution of labels within each group.

Table 5.3: The RF label distribution of individual accuracy from experiment 4, 5, 6 grouped by gender.

Accuracy	Accuracy > 80%		Accuracy < 80%	
	Female	Male	Female	Male
RF from Experiment 4	11	17	19	13
RF from Experiment 5	17	15	13	15
RF from Experiment 6	25	22	5	8

In experiment 5 and 6, the RF model classified the programs of 25 and 17 female programmers with more than 80% accuracy respectively. That means this model was able to correctly classify at least 80% of programs of each female-authors according to the identity of the programmer. This is a slightly higher number of programmers compared to male programmers with the individual class accuracy between 80% and 100% as shown in Table 5.3. Based on our review of previous research [40, 36, 84], it appears that females are more predictable in the competitive situations in both academic and nonacademic competitions. Similarly, our results in experiments 5 and 6 on dataset Coll-F indicated that the models could identify a slightly higher percent-

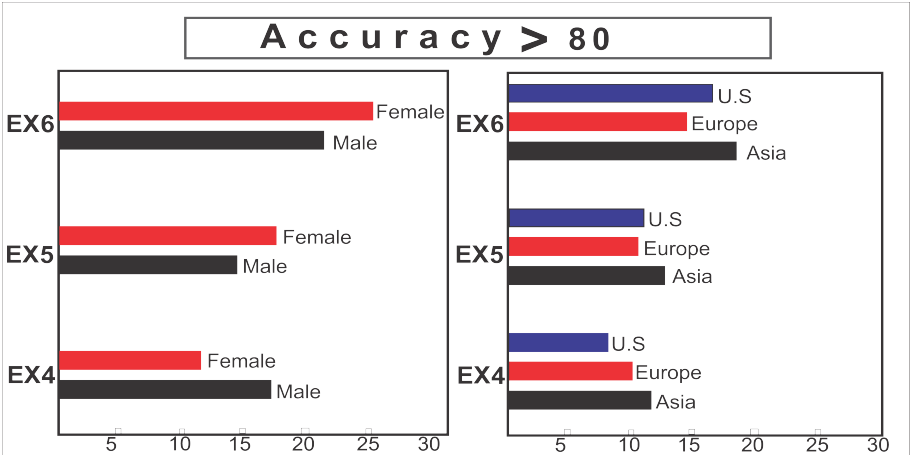


Figure 5.10: Label distribution of individual accuracy more than 80% concerning gender and region in experiment 4 (EX4), experiment 5 (EX5), and experiment 6 (EX6).

age of female programmers in a competitive situation with the individual accuracy between 80% and 100%.

5.2.3 Region-based Analysis

Table 5.4: The RF label distribution of individual accuracy from experiment 4, 5, 6 grouped by region.

Experiments	Accuracy > 80%			Accuracy < 80%		
	Asia	Europe	U.S.	Asia	Europe	U.S.
RF from Experiment 4	11	9	8	9	11	12
RF from Experiment 5	13	9	10	7	11	10
RF from Experiment 6	17	14	16	3	6	4

In Figure 5.10 we analyzed label distribution concerning region of the programmer. In terms of region, when we could predict the authors with the individual accuracy more than 80%, the numbers of Asian authors was higher in all cases, comparing to the other two regions. The low author identification performance in Europe and the US could be the result of population variations in those regions as we explained in Section 5.1.2.

5.3 Feature Analysis

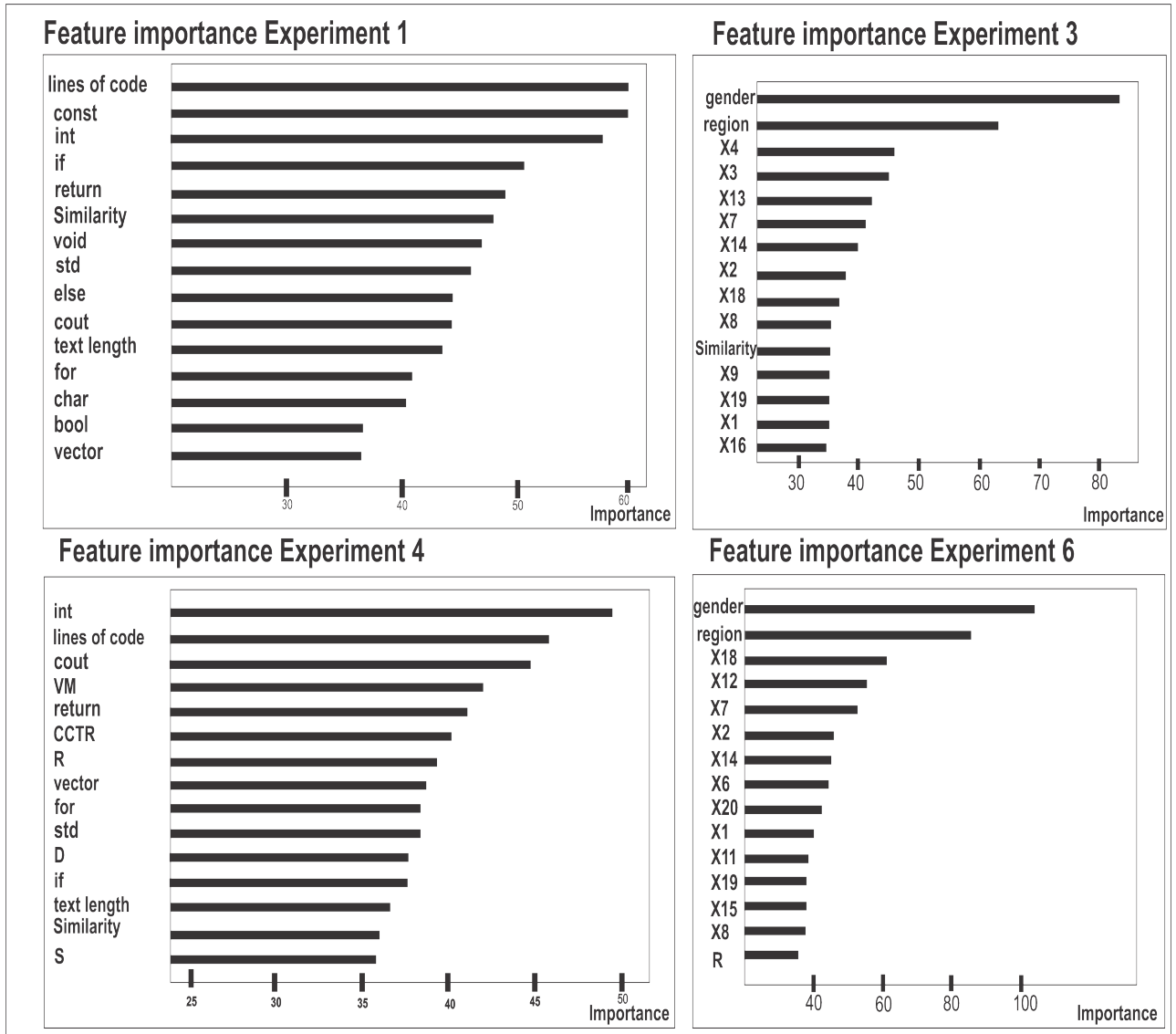
In this section, we discuss features used in our work and their importance in determining program authorship. To find the most significant features for our learning models, we applied a feature evaluator: random forest feature importance. We used the “varimp()” function in the R “random forest” package. RF feature importance determines the importance of features for each experiment individually and assigns a numerical value to each feature based on its efficiency for RF classification. Using this feature evaluator, we plotted the top 15 highest-ranked features. Note that we plotted feature importance in experiments 1, 3, 4 and 6, as experiments 2 and 5 had nearly the same feature importance as experiments 3 and 6.

In experiments 1 and 4, we used software metrics, lexical diversity features, and similarity feature. Experiment 1 used dataset Coll-G, a repository website for programming. In this repository, there is no pressure on time and results while programmers write (non-competitive). Experiment 4 used dataset Coll-F, a website for competitive programming; programmers are under pressure for time and accuracy while they participate in the programming contest.

We chose a subset of the top 15 highest-ranked features out of 28 features in each of experiments 1 and 4 as shown in Figure 5.11. These were the most important features, according to the random forest importance technique. There were 9 common features between experiments 1 and 4. Note that *lines of code* and *int* were the first two important features in both experiments.

Although in experiment 1 (Coll-G) there were no lexical diversity features among the top 15 highest-ranked features, in experiment 4 (Coll-F) 5 lexical diversity features appeared in the top 15: VM, CCTR, R, D, S. This may support the idea that programmers are more likely to use prepared blocks of code or template in competitive programming. In this case, if unlike others, a programmer writes a computer program spontaneously using different names for functions or variables rather than

Figure 5.11: Top 15 highest-ranked features in Experiments 1,3,4 and 6.



the predefined names present in prepared blocks of code, the diversity of that author’s vocabulary and keywords will make them stand out and easy to classify.

Experiments 3 and 6 showed that adding non-content based features to content-based features improved performance significantly. We selected a subset of the top 15 highest-ranked features out of 32 features in each of experiments 3 and 6 shown in Figure 5.11. In both experiments, the authors’ gender and region (both non-content-based features) were by far the most important features.

In these two experiments, except for similarity, all remaining top-ranked features were the result of a mathematical combination of n-gram features. Similarity was one of the top 15 highest-ranked features in experiment 3 (Coll-G), but in contrast it did not appear in the set of the most important features for experiment 6 (Coll-F). This may be because many programmers use the same prepared block of code, so their programming styles look similar and therefore, it becomes harder to distinguish between programs developed by different programmers.

5.4 Author Identification using Top-ranked Features

As discussed in Section 5.3, to determine the most significant features for the learning models, we applied a feature evaluator: random forest feature importance. In experiment 3 and experiment 6 we used this to select the top 15 highest-ranked features for classifying programs according to the programmer’s identity.

We reduced the feature set in each experiment to the subset of the top 15 highest-ranked features and re-applied the learning models. Our goal was to determine whether a reduced feature set is sufficient to classify the programs according to the programmer’s identity. To evaluate the impact of the reduced feature set, we performed a comparative analysis between the models developed with all 32 features and the models with 15 features, as shown in Table 5.5 and Table 5.6.

Table 5.5: Comparing the results of experiment 3 using the 15 most important features and the 32 initial features.

Models	32 features		15 features	
	Accuracy	Kappa	Accuracy	Kappa
RF	74.78%	74.31%	74.61%	74.07%
SVM	59.11%	58.34%	56.93%	56.07%
NB	45.44%	44.44%	47.05%	46.03%

Table 5.6: Comparing the results of experiment 6 using the 15 most important features and the 32 initial features.

Models	32 features		15 features	
	Accuracy	Kappa	Accuracy	Kappa
RF	76.47%	76.04%	76.98%	76.56%
SVM	66.37%	65.75%	73.8%	72.69%
NB	55.37%	54.54%	61.51%	60.77%

Table 5.5 shows the performance of the three models with all 32 features and the performance of the three models with the reduced 15 features in experiment 3. Performances are shown in terms of accuracy and kappa of the models. From Table 5.5, we can see that the classification models with a reduced number of features gave similar performance to the models with all the 32 features. In experiment 3, of the three classification models, the RF had the highest accuracy and Kappa with all 32 features and also with a subset of 15 high-ranked features. The NB model achieved slight performance increases after reducing the features. Only the SVM model had a lower model performance with the reduced feature set.

In experiment 6, we used 32 features to categorize computer programs based on the author identity in Coll-F, and then we reduced the feature set to 15 features by selecting the 15 highest-ranked features using the random forest feature importance evaluator. We used the top 15 features and re-applied experiment 6. The results

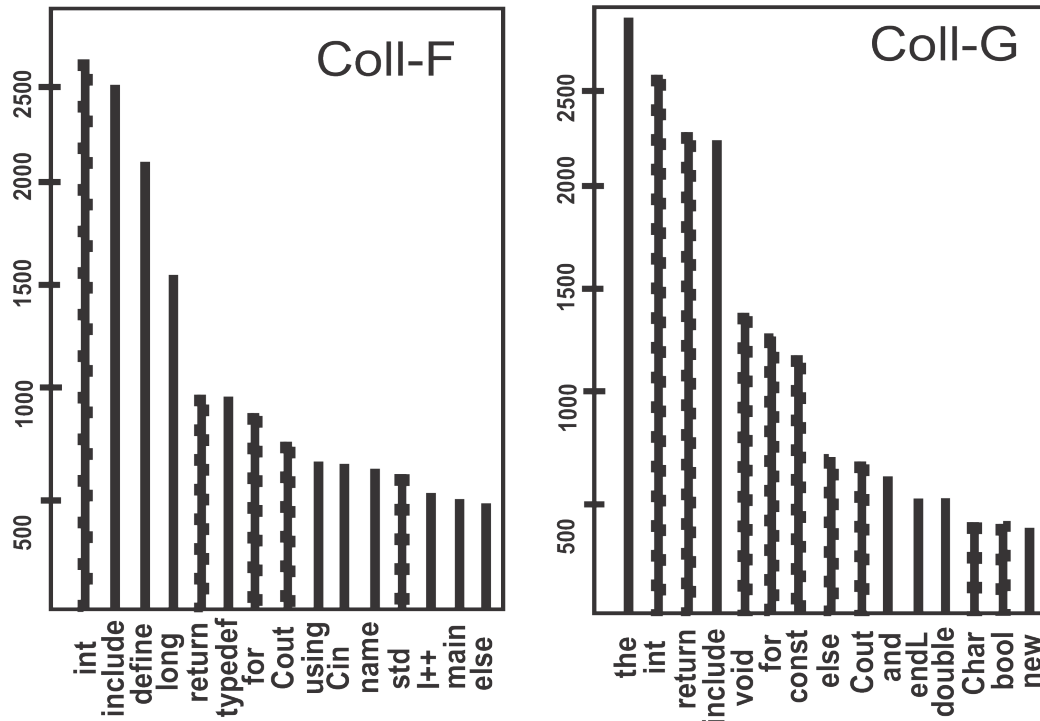
are shown in Table 5.6. Removing some features appeared to help remove the noisy data from the dataset and actually helped the classification model to predict the class more accurately. The RF models achieved similar model performance with both feature sets, but the SVM and NB models achieved a higher model performance with the reduced feature set.

5.5 Word Frequencies

In authorship attribution research in natural language, features have always played a critical role. As in the literary domain, some research claims that words frequently appearing in the text without having much content information (e.g. prepositions, conjunctions, etc) should be removed from that text prior to feature extraction. Similarly, words occurring quite often in the text are said to provide little information for distinguishing between documents and author style [8]. In contrast to previous work in this field, which used content terms to represent documents, recent work has proposed methods based on stop words, conjunctions, and adverbs used in natural language. Any language (natural language) has a large set of conjunctions that can be used to link words, phrases, and clauses. Such conjunctions can be used in different ways without modifying the meaning of the text. For example, the sentence “Michael is rich, but Alexis is poor”, could be written in several ways using other conjunctions (for example, “Michael is rich; however, Alexis is poor.” or “Michael is rich; on the other hand, Alexis is poor.”). Some researchers claim that how conjunctions, stop words, and the most frequent words are used is unique to each author, and thus should be used as features in this types of work [80, 56, 88].

In this section, we list the most frequently occurring words in the computer programs in datasets Coll-F and Coll-G. We also discuss whether these frequently occurring words play a key role in author identification using content-based features or whether we should focus on the approach, that suggests removing these words. We

Figure 5.12: Top 15 frequently occurring words from datasets Coll-G and Coll-F.



extracted the 15 most frequent words in written programs from each data collection as shown in Figure 5.12.

Next, we compared the most frequently occurring words in Coll-F with the most important keywords in Coll-F. Experiment 4 was the only experiment that included Coll-F’s keywords in its feature set. Therefore, to find the most important keywords in Coll-F, we examined the top-ranked features of experiment 4 (from Section 5.3). As shown in Figure 5.13 the top 15 highest-ranked features of experiment 4 included seven keywords: *int*, *cout*, *return*, *vector*, *for*, *std*, *if*.

We see that among the seven important keywords, five of them also appeared in the frequency list of dataset Coll-F. We highlight the common keywords in the frequency charts in Figure 5.12.

Using dataset Coll-G, we compared the most frequent words to the important features in experiment 1. In experiment 1, we used 18 software metrics, 9 lexical diversity features, and 1 similarity features. We plotted the most important features

Figure 5.13: Top-ranked features of experiment 4. The red line indicate the keywords.

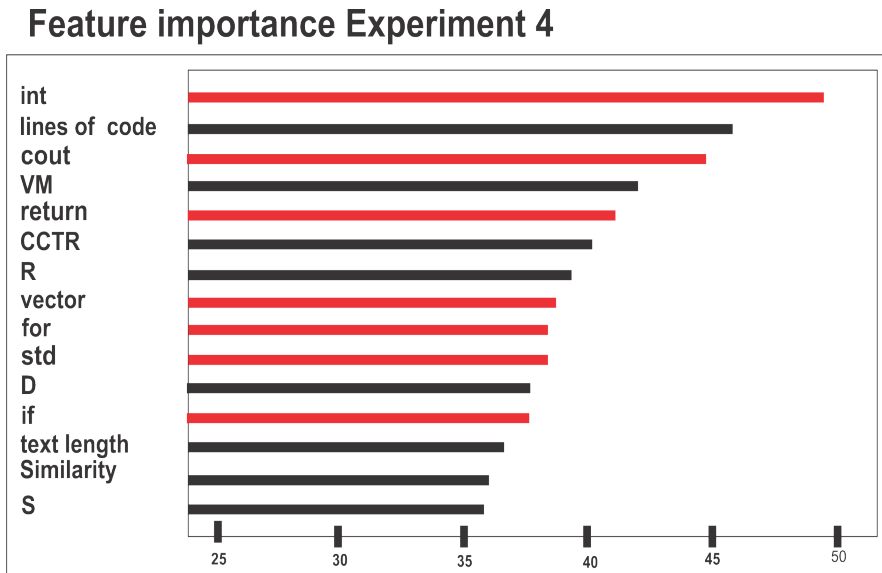
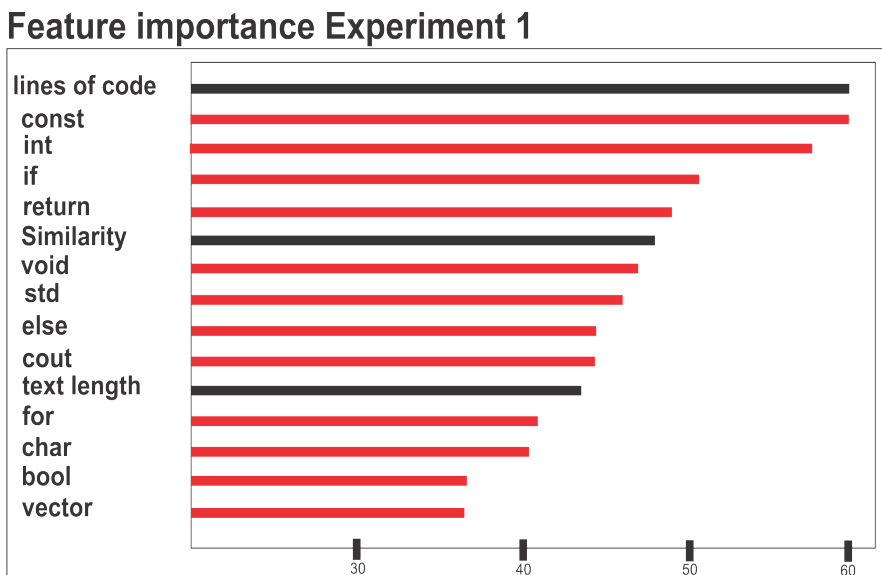


Figure 5.14: Top-ranked features of experiment 1. The red line indicate the keywords.



in this experiment in Figure 5.14. Out of 15 features, there are 12 keywords as shown in Figure 5.14 with red lines. We see that there are 9 common words between the frequency word list of dataset Coll-G and the most important features of dataset Coll-G, which highlighted in Figure 5.12.

It seems that in this work, a stylometry approach that uses frequently occurring words reveals the better model performance. This approach takes full advantage of the words occurring quite often in the text, instead of following the common practice of eliminating words that frequently appear in the text.

Chapter 6

Conclusion

Based on Burrows' research [19], text mining approaches can be used to determine the authorship of computer programs. Burrows stated that each programmer's unique fingerprint is revealed with the programming language choice. Researchers [83, 74, 7, 89] have demonstrated that sociolinguistic factors such as the author's gender, region and experiences affect how computer programs are written by the author.

Our research is mainly focused on the use of machine learning techniques to classify computer programs based on the identity of the author. In this study, two separate datasets with the concept of a competitive and non-competitive environment were collected. On each dataset, we performed three experiments using three machine learning models that used different types of features. The features we used were content-based features derived from the authors' programs and sociological factors derived from their programming profiles. Then, we evaluated the effectiveness of these features in classifying computer programs.

The feature analysis and machine learning experiments revealed some interesting results:

- All six experiments demonstrate that the RF model performed better than other machine learning models on both datasets because it makes predictions using multiple machine learning algorithms, as we noted in Section 2.3.
- In experiment 2 on the dataset Coll-G and experiment 5 on the dataset Coll-F, the performance of machine learning models was significantly improved by

using appropriate n-gram features rather than software metrics. In experiment 2, 3-grams appeared to be a valuable feature because due to the non-competitive environment of dataset Coll-G, shorter n-gram lengths (3-gram) could be used to capture the pattern of the text very accurately. The performance of the models improved between 5% to 10% by using the 3-gram features in experiment 2. The 6-gram proved to be an appropriate fit for n-gram length in experiment 5 as Coll-F is a competitive environment, and there may be a greater probability of using the prepared block of code. Replacing software metrics with 6-gram features improved model performance by more than 8%.

- Experimental 3 and 6 showed that machine learning model performance was increased significantly using a feature set that included both content-based and none-content-based features. In each experiment we added gender and region to the content-based features. The performance of the models improved significantly in both experiments after combining these two groups of features.
- In our work, the highest performance for author identification was achieved in experiment 3 for dataset Coll-G and experiment 6 for dataset Coll-F using a set of 32 features. We used a Random Forest feature evaluator to reduce the dimension of feature set in each experiment, and only the 15 most important features were used to re-apply machine learning models. Although the reduced feature set was used, the change in performance of the machine learning models was not significant. This shows that it is possible to achieve almost the same performance by selecting the most important features in computer program classification.
- In each dataset, the keywords that were most important for classifying the author of unknown programs were the same as the keywords that were frequent in programs of that dataset.

- In order to evaluate our methodology, we applied the same methodology to two datasets with different concepts (competitive and non-competitive) and different data collection procedures. We found that overall performance was about the same regardless of whether we used dataset Coll-G or dataset Coll-F. But we found a higher number of male authors with more accurate results than female authors in the Github dataset. In Codeforce the trend was converse and the female author's identification was more accurate.
- Author identification of computer code, which we performed in our work, is a special case of text mining. It is likely that we will be able to use the techniques we used in our work in other application of text mining as well. It should be noted that the addition of sociolinguistic features to the content-based features would likely improve natural language processing models, but some of the other features might not carry over due to the unique nature of the features found in code.

6.1 Future work

There is scope for much work to extend this thesis. Below we have listed up everything we believe is appropriate recommendations for further work.

- In the future it would be interesting to use additional social variable (non-content-based) features in our feature set, such as the level of programmer's experience, age, and academic degree. It could be even useful to see the companies that the programmers had prior work experience with.
- Author identification performance may depend mostly on the number of authors and the number of samples collected for each author. Further research could focus on determining a threshold of the appropriate number of authors with an adequate number of samples.

- One aim of our study was to evaluate various lengths of n-grams to determine which is the best to select as a feature. In the future, it would be interesting to find out how combinations of different n-grams lengths perform. For example, instead of using only 6-gram features, a combination of 6-grams and 3-grams might perform better.
- Using appropriate n-grams and sociolinguistic features, it is possible to categorize C++ programs based on the author's identity. On programming websites such as Codeforce and Github, programmers may use more than one programming language to write programs in a repository or to solve problems in competitions. In other words, an author on GitHub might have some samples in C++ language and others in Java or Python. A proposed feature selection in our work is independent of the language of programming, so we can make use of all the program samples provided by authors in various programming languages. Further research will take into consideration how our methodology performs when applied to a dataset consisting of author samples in different programming languages.
- Our experiments are closed-world problems. This means that the identification of the author is restricted to programmers whose codes are analyzed for authorship attribution. For example, if there are n number of programmers in the experiments (which means n label), the solution must be from those n labels. The solution is not general and, thus, cannot be valid for other authors who do not exist in the dataset. It would be possible in the future to work on open world problems with the use of unsupervised learning and the same combination of features from our study.
- Our work demonstrates the efficiency of ensemble models such as the RF in author identification. It will be interesting in the future to develop an ensemble

machine learning model that combines supervised learning algorithms such as RF and NB to classify computer programs.

Bibliography

- [1] M. Abuhamad, T. AbuHmed, A. Mohaisen, and D. Nyang. Large-scale and language-oblivious code authorship identification. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18*, page 101–114. Association for Computing Machinery, 2018.
- [2] D. Adams. Oracle database online documentation 12c release 1 (12.1), 2014. <https://docs.oracle.com/en/cloud/paas/exadata-express-cloud/csdbk/known-issues-oracle-database-exadata-express-cloud-service.pdf>.
- [3] S. Afroz. *Deception in authorship attribution*. PhD thesis, Department of Computer Science, Drexel University, Philadelphia, PA, 2013. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.715.9382&rep=rep1&type=pdf>.
- [4] C.C. Aggarwal and C. Zhai. *An Introduction to Text Mining*. Springer US, Boston, MA., 2012.
- [5] H. Ahmed. *Detecting opinion spam and fake news using n-gram analysis and semantic similarity*. PhD thesis, Department of Electrical and Computer Engineering, University of Victoria, 2017. <https://dspace.library.uvic.ca//handle/1828/8796>.
- [6] H. Ahmed, I. Traore, and S. Saad. Detection of online fake news using N-gram analysis and machine Learning Techniques. In *Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments*, volume 10618, pages 38–127. Springer International Publishing, 2017.
- [7] S. Alam. Computer program complexity and its correlation with program features and sociolinguistics. Master’s thesis, Department of Mathematics and Computer Science, University of Lethbridge, 2020.
- [8] M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E.D Trippe, J.B Gutierrez, and K. Kochut. A brief survey of text mining: Classification, clustering and extraction techniques. *ArXiv:1707.02919 [Cs]*, 2017.
- [9] E.L. Allwein, R.E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *J. Mach. Learn. Res.*, 1:113–141, September 2001.
- [10] S. Argamon, J.B. Goulain, R. Horton, and M. Olsen. Vive la différence! text mining gender difference in french. *Digital Humanities Quarterly*, 3(2), 2009.

-
- [11] S. Argamon, M. Koppel, J. Fine, and A. R. Shimoni. Gender, genre, and writing style in formal written texts. *Text-The Hague Then Amsterdam Then Berlin*, 23(3):321–346, 2003.
- [12] S. Argamon, M. Šarić, and S.S. Stein. Style mining of electronic messages for multiple authorship discrimination: First results. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, page 475–480, New York, NY, USA, 2003. Association for Computing Machinery.
- [13] E. Backer and P. Kranenburg. On musical stylometry—a pattern recognition approach. *Pattern Recognition Letters*, 26(3):299–309, 2005. In Memoriam: Azriel Rosenfeld.
- [14] R. Bali, D. Sarkar, B. Lantz, and C. Lesmeister. *R: Unleash Machine Learning Techniques*. Packt Publishing, Birmingham, UK, 2016.
- [15] J. Batalova. The top sending regions of immigrants in Australia, Canada, and the United States. *Migrationpolicy.Org*, 2013. <https://www.migrationpolicy.org/programs/data-hub/top-sending-regions-immigrants-australia-canada-and-united-states>.
- [16] A. Bhowmick, A. Biswas, N. AnveshKumar, and R. Kottath. Identification/segmentation of indian regional languages with singular value decomposition based feature embedding. *Applied Acoustics*, 176:107864, 2021.
- [17] C.M Bishop. *Pattern recognition and machine learning*. Information science and statistics. Springer, New York, NY, 2006.
- [18] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Jan. 2001.
- [19] S. Burrows. *Source code authorship attribution*. PhD thesis, Department of Computer Science and Information Technology, RMIT University, Melbourne, Australia, 2010. <https://researchbank.rmit.edu.au/view/rmit:10828>.
- [20] S. Burrows and S.M. Tahaghoghi. Source code authorship attribution using n-grams. In *Proceedings of the twelfth Australasian document computing symposium, Melbourne, Australia, RMIT University*, pages 32–39, Jan. 2007.
- [21] S. Burrows, A.L. Uitdenbogerd, and A. Turpin. Application of information retrieval techniques for source code authorship attribution. In *International Conference on Database Systems for Advanced Applications*, volume 5463, pages 699–713, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- [22] S. Burrows, A.L. Uitdenbogerd, and A. Turpin. Temporally robust software features for authorship attribution. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, volume 1, pages 599–606, 2009.

-
- [23] A. Caliskan-Islam, R. Harang, A. Liu, A. Narayanan, C. Voss, F. Yamaguchi, and R. Greenstadt. De-anonymizing programmers via code stylometry. In *Proceedings of the 24th USENIX Conference on Security Symposium, SEC'15*, page 255–270, USA, 2015. USENIX Association.
- [24] J.G. Carbonell, R.S Michalski, and T.M Mitchell. 1 - an overview of machine learning. In *Machine Learning*, pages 3–23. Morgan Kaufmann, San Francisco (CA), 1983.
- [25] L. Cheikhi, R.E. Al-Qutaish, and A. Idri. Software productivity: Harmonization in ISO/IEEE software engineering standards. *JOURNAL OF SOFTWARE*, 7(2):462–470, 2012.
- [26] P. Cingolani and J. Alcalá-Fdez. jfuzzylogic: a robust and flexible fuzzy-logic inference system language implementation. In *2012 IEEE International Conference on Fuzzy Systems*, pages 1–8, 2012.
- [27] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20(3):273–297, 09 1995.
- [28] M. A. Covington and J. D. McFall. Cutting the gordian knot: The moving-average type-token ratio (mattr). *Journal of Quantitative Linguistics*, 17(2):94–100, 2010.
- [29] E. Dauber, A. Caliskan-Islam, R. Harang, G. Shearer, M. Weisman, F. Nelson, and R. Greenstadt. Git blame who?: Stylistic authorship attribution of small, incomplete source code fragments. *Proceedings on Privacy Enhancing Technologies*, 2019(3):389–408, Jul 2019.
- [30] A. Dreweke, I. Fischer, T. Werth, and M. Wörlein. *Text Mining in Program Code*. IGI global, Boston, MA., 2009.
- [31] J. Dukart. Basic concepts of image classification algorithms applied to study neurodegenerative diseases. In Arthur W. Toga, editor, *Brain Mapping*, pages 641–646. Academic Press, Waltham, 2015.
- [32] I. El Naqa and M.J. Murphy. What is machine learning? In *Machine Learning in Radiation Oncology: Theory and Applications*, pages 3–11. Springer International Publishing, Cham, 2015.
- [33] G. Frantzeskou, E. Stamatatos, S. Gritzalis, and S. Katsikas. Effective identification of source code authors using byte-level information. In *Proceedings of the 28th International Conference on Software Engineering, ICSE '06*, page 893–896, New York, NY, USA, 2006. Association for Computing Machinery.
- [34] K. C. Fraser, J. A. Meltzer, and F. Rudzicz. Linguistic features identify alzheimer’s disease in narrative speech. *Journal of Alzheimer’s Disease*, 49(2):407–422, 2016.

-
- [35] C. Gilligan. *In A Different Voice: Psychological Theory And Women's Development*. Harvard University Press, 1993.
- [36] O. Gokcekus, A. Godet, and H. Ramsey. Are women more predictable than men? *Applied Economics*, 42(5):641–645, 2010.
- [37] J. Good and K. Howland. Natural language and programming: Designing effective environments for novices. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 225–233, 2015.
- [38] C. Gregori-Signes and B. Clavel-Arroitia. Analysing lexical density and lexical diversity in university students' written discourse. *Procedia-Social and Behavioral Sciences*, 198:546–556, 2015.
- [39] J. Grieve. Quantitative authorship attribution: An evaluation of techniques. *Literary and Linguistic Computing*, 22(3):251–270, 07 2007.
- [40] AL. Gross, J. Faggen, and K. McCarthy. The differential predictability of the college performance of males and females. *Educational and Psychological Measurement*, 34(2):363–365, 1974.
- [41] A. Gungor. Benchmarking authorship attribution techniques using over a thousand books by fifty victorian era novelists. Master's thesis, Department of Computer and Information Science, Purdue University, Indiana, U.S., 2018.
- [42] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H Witten. The WEKA data mining software: an update. *ACM SIGKDD Expl. Newslett*, 11(1):10–18, 2009.
- [43] K. Hancock. 1997 Canadian copyright act revisions. *Berkeley Tech. L.J.*, 13:517, 1998.
- [44] G. Herdan. A new derivation and interpretation of yule's 'characteristic' k. *Journal of Applied Mathematics and Physics (ZAMP)*, 6(4):332–339, 1955.
- [45] J. Holmes. *An Introduction to Sociolinguistics*. Routledge, 4th edition, 2013.
- [46] R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.
- [47] International Research Institute. Statistical techniques in the data library: A tutorial. <https://iridl.ldeo.columbia.edu/dochelp/StatTutorial/SVD/index.html>.
- [48] G. Izmirlian. Application of the random forest classification algorithm to a selditof proteomics study in the setting of a cancer prevention trial. *Annals of the New York Academy of Sciences*, 1020(1):154–174, 2004.

- [49] G. James, D. Witten, T. Hastie, and R. Tibshirani. Classification. In *An Introduction to Statistical Learning: with Applications in R*, pages 127–173. Springer New York, New York, NY, 2013.
- [50] S. L. James. Cheating behavior, subject variables, and components of the internal-external scale under high and low risk conditions. *The Journal of Educational Research*, 74(2):83–87, 1980.
- [51] S. Jarvis. Capturing the diversity in lexical diversity. *Language Learning*, 63(1):87–106, Feb. 2013.
- [52] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Machine Learning: ECML-98*, pages 137–142. Springer Berlin Heidelberg, 1998.
- [53] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.
- [54] V. Kalgutkar, R. Kaur, H. Gonzalez, N. Stakhanova, and A. Matyukhina. Code authorship attribution: Methods and challenges. *ACM Computer Survey*, 52(1):1–36, 2019.
- [55] B. Kenneth, K. Watanabe, H. Wang, P. Nulty, A. Obeng, and A. Müller, S. and Matsuo. Quanteda: An R package for the quantitative analysis of textual data. *Journal of Open Source Software*, 3(30):774, 2018.
- [56] JA. Khan. A model for style change detection at a glance. *Training*, 593(293):113, 1981.
- [57] I. Krsul and E.H. Spafford. Authorship analysis: identifying the author of a program. *Computers and Security*, 16(3):233–257, 1997.
- [58] M. Kuhn. Building predictive models in R using the caret package. *Journal of Statistical Software, Articles*, 28(5):1–26, 2008.
- [59] V. Y. Kulkarni and P. K. Sinha. Pruning of random forest classifiers: A survey and future directions. In *2012 International Conference on Data Science Engineering (ICDSE)*, pages 64–68, 2012.
- [60] P. Kwangil, J.S. Hong, and W Kim. A methodology combining cosine similarity with classifier for text classification. *Applied Artificial Intelligence*, 34(5):396–411, 2020.
- [61] A. Laaksonen. Chapter 1 - introduction. In *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests*, pages 1–7. Springer International Publishing, Cham, 2020.
- [62] J. Leskovec, A. Rajaraman, and J. D. Ullman. Chapter 11 - dimensionality reduction. In *Mining of Massive Datasets*, pages 405–424. Cambridge University Press, USA, 2014.

- [63] A. Liaw and M. Wiener. Classification and regression by random forest. *R News*, 2(3):18–22, 2002.
- [64] R. P. Lippmann, L. Kukulich, and E. Singer. Lnknet: neural network, machine-learning, and statistical software for pattern classification. Technical Report 2, Massachusetts Inst of Tech Lexington Lincoln Lab, 1993.
- [65] P. Lissón and N. Ballier. Investigating lexical progression through lexical diversity metrics in a corpus of french l3. *Discours. A Journal of Linguistics, Psycholinguistics and Computational Linguistics*, (23), Dec. 2018.
- [66] H. Liu, R. H. Chan, and Y. Yao. Geometric tight frame based stylometry for art authentication of Van Gogh paintings. *Applied and Computational Harmonic Analysis*, 41(2):590–602, 2016. Sparse Representations with Applications in Imaging Science, Data Analysis, and Beyond, Part II.
- [67] J. E. Mason. *An n-gram based approach to the automatic classification of web pages by genre*. PhD thesis, Department of Computer Science, Dalhousie University Halifax, 2009. <https://dalspace.library.dal.ca/handle/10222/12351>.
- [68] A. Matyukhina, N. Stakhanova, M. Dalla Preda, and C. Perley. Adversarial authorship attribution in open-source projects. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, CODASPY '19*, pages 291–302. Association for Computing Machinery, 2019.
- [69] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, pages 41–48, 1998.
- [70] P. M. McCarthy and S. Jarvis. vocd: A theoretical and empirical evaluation. *Language Testing*, 24(4):459–488, 2007.
- [71] P. M McCarthy and S. Jarvis. Mtdl, vocd-d, and hd-d: A validation study of sophisticated approaches to lexical diversity assessment. *Behavior Research Methods*, 42(2):381–392, May 2010.
- [72] G. Miner, J. Elder IV, A. Fast, T. Hill, R. Nisbet, and D. Delen. Chapter 3 - conceptual foundations of text mining and preprocessing steps. In *Practical Text Mining and Statistical Analysis for Non-structured Text Data Applications*, pages 43–51. Academic Press, Boston, 2012.
- [73] T.M. Mitchell. Does machine learning really work? *AI Magazine*, 18(3):11, 1997.
- [74] F. Naz and J. E. Rice. Sociolinguistics and programming. In *2015 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 74–79, 2015.
- [75] R Nisbet, J Elder, and G Miner. Chapter 4 - data understanding and preparation. In *Handbook of Statistical Analysis and Data Mining Applications (Second Edition)*, pages 55–82. Academic Press, Boston, second edition edition, 2018.

- [76] A. Nyström. When words are not enough an evaluation of character n-grams and function words in author identification of musical artists. Master's thesis, Department of Computer Science, Umeå University, Sweden, 2018. <https://www.diva-portal.org/smash/get/diva2:1286107/FULLTEXT01.pdf>.
- [77] K. Olsen-Medina and J Batalova. College educated immigrants in the United States. *Migrationpolicy.Org*, 2020. <https://www.migrationpolicy.org/article/college-educated-immigrants-united-states-2018>.
- [78] P W. Oman and C. R. Cook. A taxonomy for programming style. In *Proceedings of the 1990 ACM Annual Conference on Cooperation, CSC '90*, page 244–250. Association for Computing Machinery, 1990.
- [79] H. S. Parry Husbands and C. H. Ding. On the use of the singular value decomposition for text retrieval. *Computational information retrieval*, 5:145–156, 2001.
- [80] D. Pavelec, L.S Oliveira, E.JR Justino, and L.V. Batista. Using conjunctions and adverbs for author verification. *j-jucs*, 14(18):2967–2981, oct 2008.
- [81] E. Pekalska and R. P.W Duin. Dissimilarity representations allow for building good classifiers. *Pattern Recognition Letters*, 23(8):943–956, 2002.
- [82] R Core Team. R: A language and environment for statistical computing, 2013. <http://www.R-project.org/>.
- [83] M. H. Rafee. Computer program categorization with machine learning. Master's thesis, Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, AB, 2017.
- [84] H.G Seashore. Women are more predictable than men. *Journal of Counseling Psychology*, 9(3):70–261, 1962.
- [85] H. Sharma and S. Kumar. A survey on decision tree algorithms of classification in data mining. *International Journal of Science and Research (IJSR)*, 5(4):2094–2097, 2016.
- [86] E. H Simpson. Measurement of diversity. *Nature*, 163(4148):688, 1949.
- [87] E.H. Spafford and S.A. Weeber. Software forensics: Can we track code to its authors? *Computers and Security*, 12(6):585–595, 1993.
- [88] E. Stamatatos. Plagiarism detection using stopword n-grams. *Journal of the American Society for Information Science and Technology*, 62(12):2512–2527, 2011.
- [89] N. Tasnim. Machine learning in the classification of computer code. Master's thesis, Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, AB, 2020.

- [90] F. J. Tweedie and R. H. Baayen. How variable may a constant be? measures of lexical richness in perspective. *Computers and the Humanities*, 32(5):323–352, 1998.
- [91] A. K. Uysal and S. Gunal. The impact of preprocessing on text classification. *Information Processing and Management*, 50(1):104–112, 2014.
- [92] J. T. Wood. Gendered interaction: Masculine and feminine styles of verbal communication. *New York: Wadsworth*, pages 18–29, 1995.