

**MACHINE LEARNING IN THE CLASSIFICATION OF COMPUTER CODE**

**NAZIA TASNIM**

**Bachelor of Science, Military Institute of Science and Technology, 2010**

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

© Nazia Tasnim, 2020

# MACHINE LEARNING IN THE CLASSIFICATION OF COMPUTER CODE

NAZIA TASNIM

Date of Defence: August 27, 2020

Dr. Jacqueline E. Rice Thesis Supervisor	Professor	Ph.D.
Dr. Wendy Osborn Thesis Examination Committee Member	Associate Professor	Ph.D.
Dr. John Sheriff Thesis Examination Committee Member	Assistant Professor	Ph.D.
Dr. Yllias Chali Chair, Thesis Examination Com- mittee	Professor	Ph.D.

# Dedication

Dedicated to my parents, husband and daughter.

# Abstract

Machine learning approaches are a well-established method to analyze natural language. Sociolinguistic characteristics, such as the author's gender, experience, and age, have compelling effects on natural language use. Previous research has shown that a computer program can be analyzed using similar linguistics-based approaches. In this research, we are using machine learning techniques to analyze computer programs based on the author's programming experience. We use machine learning and statistical approaches to determine which features are most significant in the classification of a computer program according to the author's programming experience. Several experiments have been carried out on a dataset consisting of computer programs written in C++, and the results are encouraging. The experimental results estimate that the author's programming experience can be predicted with an accuracy of 69%.

# Acknowledgments

All praise be to the almighty Allah for giving me the opportunity and strength to complete this research and master's program.

I would like to express my sincerest appreciation towards my supervisor Dr. Jacqueline E. Rice, for her continuous guidance and encouragement. Dr. Rice offered her valuable suggestions and recommendations throughout the master's program. Without her inspiration and advice, this research endeavour was not possible. Thank you, Dr. Rice, for your continued support and encouragement towards me.

I want to express my sincere appreciation of Dr. Wendy Osborn and Dr. John Sheriff for being part of my supervisory committee. I am thankful for their guidance and feedback.

I must express my gratitude to my parents and my in-laws for their unconditional love and support. My family encouraged me a lot throughout my master's program.

I am thankful to my husband, Ahmed Shoeb Al Hasan and daughter, Tasmia Hasan Rayya, for being with me and support me throughout this research. I would especially like to thank my husband for always being with my side in all the difficult times of my life. Without his support and motivation, this journey was not possible.

I am grateful to the School of Graduate Studies and Dr. Rice for providing financial assistance for my graduate study and research work.

My appreciation also extends to all the members of my research group, my friends, well-wishers and my family-like members in Canada, especially, Ashif Rahman and Tanzim Haque.

# Contents

<b>Contents</b>	<b>vi</b>
<b>List of Tables</b>	<b>ix</b>
<b>List of Figures</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	3
1.2 Contributions . . . . .	4
1.3 Organization of Thesis . . . . .	5
<b>2 Background and Literature Review</b>	<b>6</b>
2.1 Sociolinguistics . . . . .	6
2.2 Machine Learning . . . . .	6
2.2.1 Types of Machine Learning . . . . .	7
2.3 Data . . . . .	10
2.3.1 Data Preparation Plan . . . . .	12
2.4 Classification Algorithms . . . . .	14
2.4.1 Decision Tree . . . . .	15
2.4.2 Naïve Bayes . . . . .	22
2.4.3 Random Forest . . . . .	27
2.4.4 K Nearest Neighbor Algorithm . . . . .	28
2.4.5 Logistic Regression . . . . .	31
2.4.6 Bagging Classifier . . . . .	32
2.5 Evaluation of Machine Learning Algorithms . . . . .	33
2.5.1 Holdout Method . . . . .	33
2.5.2 Cross-Validation . . . . .	34
2.5.3 Performance Measurement . . . . .	34
2.6 Python . . . . .	37
2.6.1 Feature Selection . . . . .	37
2.7 Related Work . . . . .	38
2.8 Summary . . . . .	40
<b>3 Methodology</b>	<b>41</b>
3.1 Data Collection . . . . .	41
3.1.1 Data Collection Process . . . . .	41
3.1.2 Drawbacks of the Data Source . . . . .	43
3.2 Dataset Creation . . . . .	44

---

3.2.1	Balanced Data Set . . . . .	48
3.3	Document Representation . . . . .	50
3.4	Feature Selection . . . . .	50
3.5	Summary . . . . .	53
<b>4</b>	<b>Experiments and Results</b>	<b>54</b>
4.1	Preliminaries . . . . .	54
4.1.1	Programming Environment . . . . .	56
4.1.2	Parameter Settings . . . . .	57
4.2	Experiments . . . . .	58
4.2.1	Experiment 1 . . . . .	58
4.2.2	Experiment 2 . . . . .	60
4.2.3	Experiment 3 . . . . .	62
4.2.4	Experiment 4 . . . . .	64
4.2.5	Experiment 5 . . . . .	65
4.2.6	Experiment 6 . . . . .	66
4.3	Results . . . . .	67
4.3.1	Experiment 1 . . . . .	67
4.3.2	Experiment 2 . . . . .	70
4.3.3	Experiment 3 . . . . .	72
4.3.4	Experiment 4 . . . . .	74
4.3.5	Experiment 5 . . . . .	76
4.3.6	Experiment 6 . . . . .	78
4.4	Discussion . . . . .	80
4.4.1	Cross-Validation Method . . . . .	80
4.4.2	Holdout Method . . . . .	82
4.4.3	Summary of Results . . . . .	84
4.5	Threats to Validity . . . . .	85
<b>5</b>	<b>Analysis of Features</b>	<b>87</b>
5.1	Feature Reduction . . . . .	87
5.2	Statistical Approach . . . . .	89
5.3	Visual Analysis of Programs . . . . .	94
5.4	Relationship Between Features . . . . .	96
5.4.1	Beginner-written Programs . . . . .	96
5.4.2	Expert-written Programs . . . . .	98
<b>6</b>	<b>Conclusion and Future Work</b>	<b>103</b>
6.1	Future Research Directions . . . . .	106
	<b>Bibliography</b>	<b>108</b>
<b>A</b>	<b>Python Library</b>	<b>112</b>
A.1	Reading Text Data . . . . .	112
A.2	Natural Language Toolkit . . . . .	112
A.3	Scikit-learn . . . . .	113

A.4	math . . . . .	113
A.5	NumPy . . . . .	113
A.6	Merits . . . . .	113
A.7	Demerits . . . . .	114



# List of Tables

2.1	Ten days observation of weather conditions. . . . .	11
2.2	Frequency of the classes (Yes and No) . . . . .	16
2.3	Frequency of Yes and No for Play depending on Weather . . . . .	17
2.4	Frequency of Yes and No for Play depending on Humidity . . . . .	18
2.5	Frequency of Yes and No for Play depending on Wind . . . . .	19
2.6	Sub table for Sunny attribute. . . . .	21
2.7	Sub table for Rainy attribute. . . . .	22
2.8	Training data for KNN. . . . .	29
2.9	Distance from training data to new data. . . . .	30
2.10	Three nearest neighbors of new point (6,9). . . . .	30
2.11	Confusion matrix . . . . .	35
3.1	Attributes of the collected data from Codeforces. . . . .	45
3.2	Selected attributes of the dataset. . . . .	47
3.3	Information about the dataset. . . . .	49
3.4	The fifteen features selected to characterize the programs in our dataset. . . . .	52
4.1	Details of the test environment. . . . .	56
4.2	Performance evaluation of six models with 15 features. . . . .	67
4.3	Confusion matrix for decision tree with 15 features. . . . .	68
4.4	Confusion matrix for naïve Bayes with 15 features. . . . .	68
4.5	Confusion matrix for random forest with 15 features. . . . .	68
4.6	Confusion matrix for K nearest neighbor with 15 features. . . . .	69
4.7	Confusion matrix for logistic regression with 15 features. . . . .	69
4.8	Confusion matrix for bagging classifier with 15 features. . . . .	69
4.9	Performance evaluation of six models with 7 features. . . . .	70
4.10	Confusion matrix for decision tree with 7 features. . . . .	70
4.11	Confusion matrix for naïve Bayes with 7 features. . . . .	70
4.12	Confusion matrix for random forest with 7 features. . . . .	71
4.13	Confusion matrix for K nearest neighbor with 7 features. . . . .	71
4.14	Confusion matrix for logistic regression with 7 features. . . . .	71
4.15	Confusion matrix for bagging classifier with 7 features. . . . .	71
4.16	Performance evaluation of six models with 4 features. . . . .	72
4.17	Confusion matrix for decision tree with 4 features. . . . .	72
4.18	Confusion matrix for naïve Bayes with 4 features. . . . .	73
4.19	Confusion matrix for random forest with 4 features. . . . .	73
4.20	Confusion matrix for K nearest neighbor with 4 features. . . . .	73
4.21	Confusion matrix for logistic regression with 4 features. . . . .	73

4.22	Confusion matrix for bagging classifier with 4 features. . . . .	73
4.23	Performance evaluation of six models with 15 features. . . . .	74
4.24	Confusion matrix for decision tree with 15 features. . . . .	75
4.25	Confusion matrix for naïve Bayes with 15 features. . . . .	75
4.26	Confusion matrix for random forest with 15 features. . . . .	75
4.27	Confusion matrix for K nearest neighbor with 15 features. . . . .	75
4.28	Confusion matrix for logistic regression with 15 features. . . . .	75
4.29	Confusion matrix for bagging classifier with 15 features. . . . .	76
4.30	Performance evaluation of six models with 7 features. . . . .	76
4.31	Confusion matrix for decision tree with 7 features. . . . .	77
4.32	Confusion matrix for naïve Bayes with 7 features. . . . .	77
4.33	Confusion matrix for random forest with 7 features. . . . .	77
4.34	Confusion matrix for K nearest neighbor with 7 features. . . . .	77
4.35	Confusion matrix for logistic regression with 7 features. . . . .	77
4.36	Confusion matrix for bagging classifier with 7 features. . . . .	78
4.37	Performance evaluation of six models with 4 features. . . . .	78
4.38	Confusion matrix for decision tree with 4 features. . . . .	79
4.39	Confusion matrix for naïve Bayes with 4 features. . . . .	79
4.40	Confusion matrix for random forest with 4 features. . . . .	79
4.41	Confusion matrix for K nearest neighbor with 4 features. . . . .	79
4.42	Confusion matrix for logistic regression with 4 features. . . . .	79
4.43	Confusion matrix for bagging classifier with 4 features. . . . .	80
4.44	Models with highest and lowest accuracy rate in experiment 1 to 3. . . . .	81
4.45	Best and worst models in classifying beginner-written programs for experiments 1 to 3. . . . .	81
4.46	Best and worst models in classifying expert-written programs for experiments 1 to 3. . . . .	82
4.47	Models with highest and lowest accuracy rate in experiments 4 to 6. . . . .	82
4.48	Best and worst models in classifying beginner-written programs for experiments 4 to 6. . . . .	83
4.49	Best and worst models in classifying expert-written programs for experiments 4 to 6. . . . .	83
5.1	Performance evaluation of six models based on five-fold cross-validation technique. . . . .	88
5.2	T-test ( $\rho$ ) values of features. . . . .	91
5.3	Comparison of feature usage in expert-written and beginner-written programs. . . . .	93
5.4	First comparison between expert-written and beginner-written programs. . . . .	94
5.5	Second comparison between expert-written and beginner-written programs. . . . .	95
5.6	Strongly connected features of beginner-written programs. . . . .	97
5.7	Strongly related features of expert-written programs. . . . .	98
5.8	Strongly related features of expert-written programs (continued). . . . .	99

# List of Figures

2.1	Supervised learning model [51] . . . . .	8
2.2	Decision tree after the first split. . . . .	21
2.3	Decision tree after split using Humidity attribute. . . . .	22
2.4	Final decision tree. . . . .	23
2.5	Random forest . . . . .	28
2.6	Bagging classifier methodology. . . . .	33
3.1	Data collection process. . . . .	42
3.2	Document representation process. . . . .	51
4.1	Steps for experiment 1. . . . .	59
4.2	Steps for experiment 2. . . . .	60
4.3	Steps for experiment 3. . . . .	62
4.4	Steps for experiment 4. . . . .	64
4.5	Steps for experiment 5. . . . .	65
4.6	Steps for experiment 6. . . . .	66
5.1	Correlation based on raw frequency of features in beginner-written programs.	101
5.2	Correlation based on raw frequency of features in expert-Written programs.	102

# Chapter 1

## Introduction

Language is a tool used by people to communicate with each other. There are many different languages in the world, such as English, Mandarin, French, and Bengali. We refer to these languages as natural languages [40]. Languages tend to be different based on geographic locations. Moreover, differences in languages are also observed based on people's age, gender and religion [28]. Apart from that, there are differences between the written version and the spoken version of any language. In general, the written version of a language is more formal than the spoken version [28].

In this modern era there is another kind of language known as artificial language. Artificial languages used to communicate with computers are called programming languages. There are different programming languages such as C, C++, Java, and Python. Programmers can use different languages to give the same instructions to a computer. Moreover, programs written to solve a specific problem may differ if the programs are written by different programmers, even if they are using the same programming language. This difference among the similar programs may occur due to the above social factors discussed for natural languages.

A programming language is used to write the instructions for computers for a specific task [21]. There are two parts to a programming language: syntax and semantics. Syntax consists of the rules that define the structure of a programming language. Syntax defines how a valid instruction can be written in a programming language. Semantics is the meaning of the instruction written using the syntax of a programming language. An instruction

can be syntactically correct, although that instruction may still be semantically wrong.

```
int x = 10;  
int y = 0;  
float z = x/y;
```

The code segment shown above is syntactically correct in the C++ programming language. However, semantically it is incorrect. The last instruction in the code segment instructs the computer to divide ten by zero, which will create an error.

Every language has its own syntax and semantics. Any program written using a specific programming language must follow the syntax and semantics of that programming language.

Sociolinguistic characteristics such as the author's gender, experience, and age have compelling effects on natural language use [48], [49]. Just as each speaker of a natural language will demonstrate their individuality through their language choices [40], so each programmer using a programming language will do the same. However, it is more difficult to identify the unique coding style, or style of language choice of a programmer as they are required to follow the syntax and semantics of a programming language to write functioning programs and, as well as being restricted by having to solve a specific problem. Every programming language has its syntax and semantics. So it is necessary to follow the syntax and semantics of a programming language to write a program. A programmer needs to follow the syntax of a programming language as the syntax is the fundamental rule of a programming language. However, particular areas of coding (or writing a computer program) can be examined to determine a programmer's unique coding style. For instance, a programmer can demonstrate a unique coding style in the layout and structure of a program, such as the number of lines, the number of blank lines, and the number of comments in a program. Moreover, the use of data structures and algorithms, preferred system calls, and the type of compiler used may reflect the unique coding style of a programmer [52].

Our research focuses on computer programs and how machine learning techniques can

be applied to analyze computer programs [36], [48], [27]. Existing research confirms that the gender (male/female) and region of the author of a computer program can be determined by machine learning techniques [32]. Machine learning is the research area that studies computer algorithms which create models by learning from training data. These models develop through knowledge. More details will be given in Chapter 2.

Our research assesses the impact of sociolinguistic characteristics, such as the author's programming experience with computer programs, by examining the computer programs created using a particular programming language. In other words, our research investigates whether it is possible to determine whether the author is a beginner programmer or is more experienced by analyzing a program that they have written.

For this research we consider only programs written in C++. These programs are text files, and machine learning is widely used for text categorization problems [49]. Different machine learning approaches are used in this work to determine the author's programming experience. Python is used as the programming language for the implementation of our research. Python has vast libraries containing different machine learning approaches, evaluation methods of machine learning models, text processing methods, and feature extraction methods. Several machine learning techniques including the decision tree, naïve Bayes, random forest, k nearest neighbor, logistic regression and bagging classifier are used to build the models.

This research will allow us to determine whether programming features or characteristics and machine learning techniques can be used to determine the programmer's level of experience.

## **1.1 Motivation**

According to Misesk-Falkoff [32], a computer program can be analyzed using the techniques of linguistics. We have also seen that machine learning can be used to analyze computer programs to find out the impact of sociolinguistic characteristics. For instance, Naz

and Rice [37] retrieved information about the influence of the author's gender on computer programs using machine learning techniques. In their work open-source implementations of machine learning algorithms were applied to C++ programs to identify the features affected by the author's gender. In another study Rafee [44] used machine learning and statistical techniques to assess the impact of two sociolinguistic characteristics, the author's gender and the author's region, on computer programs.

Our research aim is to determine whether the author is a beginner or an experienced programmer by analyzing a computer program. We have seen that the original author of a program may be determined by finding the author's gender, region, experience [25]. Thus our work on categorizing a programmer's experience may refine the use of such techniques, as well as being useful in its own right. The research outcome may also aid in plagiarism detection by finding out the original author (by detecting the gender, region ,and experience) of a computer program.

## **1.2 Contributions**

Machine learning approaches were successfully used to analyze natural language [49]. In this research we are using machine learning techniques to analyze computer programs based on the author's experience. In our work various machine learning algorithms are used including decision tree, naïve Bayes, random forest, k nearest neighbor, logistic regression and bagging classifier. The contributions of this thesis are as follows:

- Application of machine learning techniques to analyze computer programs based on the author's programming experience.
- Understanding the impacts of sociolinguistic characteristics such as the author's programming experience on computer programs.
- Determining which factors are most significant in the classification of a computer program by the author's programming experience.

- Determining the best-suited machine learning approaches for the classification of computer programs based on the author's programming experience.
- Possible application in industrial or academic settings to determine the original author of a computer program, and categorize a programmer's experience.

### **1.3 Organization of Thesis**

There are six chapters in the thesis. Chapter 1 introduces the research work, followed by discussion of the motivation of this work along with the contributions made by this thesis. This chapter ends with explaining the organization of the remainder of the thesis.

Chapter 2 discusses the background and literature review. Sociolinguistics, machine learning, Python and classification algorithms are described in this chapter.

Methodologies to categorize computer programs based on the author's programming experience are illustrated in Chapter 3. At the start of the chapter, data collection procedures are described. Next the approaches we used to classify beginner and experienced programs are introduced.

Details of all experiments and the numerical results are shown in Chapter 4. The programming environment and threats to validity are also described in this chapter.

Chapter 5 describes the analysis of features to identify an author's programming experience by analyzing a computer program from our dataset.

Chapter 6 concludes the thesis with a summary and highlight of the research. Some future research directions are also offered in this chapter.



# Chapter 2

## Background and Literature Review

In this chapter, we will discuss the background and literature review. We will explain sociolinguistics, machine learning, Python, and classification algorithms here.

### 2.1 Sociolinguistics

Sociolinguistics studies the relationship between society and language [19]. Sociolinguistics is an important field of study, offering a focus on the role of language in society. According to R. Wardhaugh, “Sociolinguistics is concerned with investigating the relationships between language and society with the goal being a better understanding of the structure of language and of how languages function in communication” [55].

Language is the most influential characteristic of social conduct. Language is used by people conveys information about their identity, origin, and association. A person’s character or background can be determined based on their language, dialect, and other factors [19].

Sociolinguistics is the branch of linguistics that deals with the study of language in society and the sociocultural context. It can also be said that sociolinguistics studies the linguistic signs of culture and power [50].

### 2.2 Machine Learning

Machine learning is based on computer algorithms which can create mathematical models by learning from training data. Machine learning is a sub-field of computational intelli-

gence, and offers solutions in image recognition, natural language processing, data mining, and expert systems.

Mitchell offered a formal definition of machine learning in [33]: “A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.” For example, when a program predicts traffic patterns at a busy intersection (task T) it can apply a machine learning algorithm to data about past traffic patterns (experience E) and, if it has successfully “learned”, it will then do better at predicting future traffic patterns (performance measure P). Using past traffic patterns the machine learning model can learn several things such as peak times for heavy traffic flow and the number of vehicles causing congestion. Assuming that traffic patterns are reasonably consistent over time. When the model has learned the traffic patterns based on previous patterns it will be able to predict traffic patterns more accurately in the future.

### **2.2.1 Types of Machine Learning**

Based on the idea of learning, machine learning tasks can be classified into three general classifications [44].

#### **Supervised Learning**

Supervised learning is a category of machine learning algorithms that develops the relation between inputs and outputs based on a given dataset of training samples [14]. The input-output pair of training samples are also known as labeled training data as the output is considered as the label of the input data. An artificial system is developed in supervised learning which learns the mapping between input and output based on training data. Then this artificial system predicts the output of new input data based on the learning from training data. Haykin [17] referred supervised learning as the learning with a teacher. The most widely recognized form of supervised learning is classification. Naïve Bayes, decision tree, k nearest neighbor, neural network and support vector machine (svm) are popular

classification algorithms.

For example, a supervised machine learning model can be developed to assign bug reports to the appropriate developers to fix the bugs [1]. In this system, the input files are bug reports, which are text files. Bug reports contain different information such as the bug report title, the description of the bug, comments, and the ID of the developer who solved the bug. Bug reports which are already solved are called RESOLVED bug reports. A RESOLVED bug report contains the ID of the developer who has solved the issue reported in that bug report. Developer IDs mentioned in the bug reports may be used as labels or classes to develop a supervised machine learning model. The trained model will then assign bug reports to appropriate developers for new bug reports that are submitted [1].

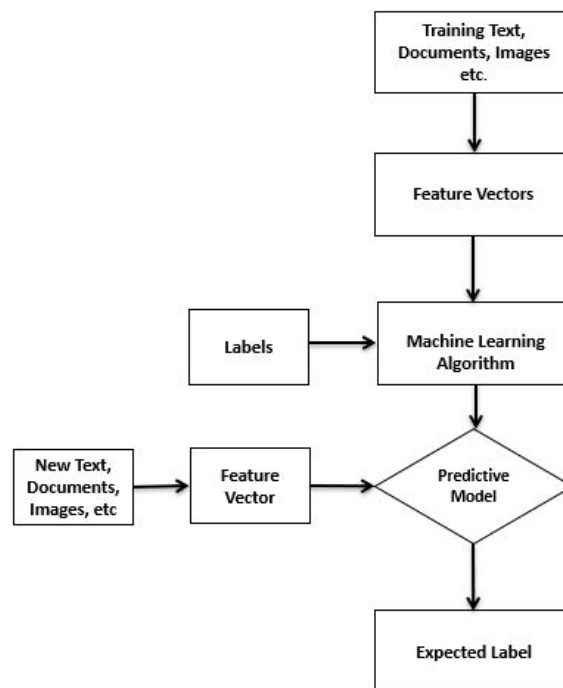


Figure 2.1: Supervised learning model [51]

Figure 2.1 shows the steps of the supervised learning model.

### **Unsupervised Learning**

In unsupervised learning there are no labels attached to the data. The learning algorithm must identify any structure or pattern in the input data. The principal objective of unsupervised learning is to find unknown structures in the data. This is known as feature learning. Clustering is the most well-known form of unsupervised learning.

For example, the input dataset of bug reports may not contain the RESOLVED bug reports [1]. That means there may be no bug reports with the developer ID who has solved the issues. In this case, there will be no label. For this scenario, a machine learning model may create clusters of developers such as cluster1, cluster2 and analyze a new bug report and the model then assign a report to a cluster.

### **Reinforcement Learning**

Reinforcement learning is another category of machine learning. A reinforcement learning algorithm maps what to do and how to do it while maximizing rewards. The difference between supervised learning and reinforcement learning is that in supervised learning, the training data has the answer key with it, so the model is trained with the correct answer whereas in reinforcement learning, there is no answer, but the reinforcement agent decides what to do to perform the given task. Without a training dataset, it will learn from its experience. The objective of unsupervised and reinforcement learning is different. The goal of unsupervised learning models is to determine any hidden structure in the input data whereas reinforcement learning knows its expected reward in the beginning and takes action to achieve that reward or goal. Learning to play chess is an example of reinforcement learning [47]. In a chess game, the final reward is already known, which is to checkmate the opponent. The two most essential characteristics of reinforcement learning are trial and error search and delayed reward [53].

## 2.3 Data

Data are the facts or details from which information is derived. In general, we use the following terms:

**Data:** Data are raw values which represent something specific, yet which are not organized and give no additional information in regards to context and pattern.

**Information:** Information is structured data with meaning. Therefore, information hence paints a more meaningful picture; it is data with importance and reason [2].

For example, there are 26 letters in the English alphabet: a to z. The characters i, d, b, r are data. However, those data have no meaning. When a word “bird” is created with these characters, then it has meaning. So the word “bird” is the information.

**Knowledge:** “Knowledge is a fluid mix of framed experience, values, contextual information, expert insight, and grounded intuition that provides an environment and framework for evaluating and incorporating new experiences and information. It originates and is applied in the mind of the knowers. In organizations, it often becomes embedded not only in documents or repositories but also in organizational routines, practices, and norms”[4]. In other words, information with experience, judgement, and skill is known as knowledge.

Machine learning algorithms require data. Machine learning approaches process the data into information and then build knowledge based on the retrieved information. Knowledge gained from the training data set is then used to make decisions based on real-world data.

For instance, bug reports contain different information such as the bug report title, the description of the bug, the comments, and the ID of the developer who solved the bug [1]. There can be a bug report having a description, “The program is crashing”. Here, “The program is crashing” is data. Suppose developer\_1 is the ID of the developer who has resolved this issue. Then the derived information is that the bug “The program is crashing” is resolved by developer\_1. The machine learning model for assigning bug reports to the appropriate developers may assign any bug having the words “program” and “crashing”

together in a sentence to developer\_1. This is the knowledge gained by the model based on data and information.

Some standard terms are used to represent different aspects of data [44]:

**Instance:** A row of a data table is known as instance and it consists of features or attributes.

**Feature:** A column of a data table is called a feature or an attribute.

**Feature Vector:** A vector that stores the feature values of an instance is known as feature vector.

Table 2.1 shows an example of ten days of observation of weather conditions. In this table the features (attributes) are weather, temperature, humidity and wind. There are 10 instances in Table 2.1 listing the weather conditions of 10 days. The feature vector for day 1 (or the 1st instance) is [Sunny, Hot, High, Weak].

Table 2.1: Ten days observation of weather conditions.

Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No

Machine learning algorithms require datasets as input. A **dataset** is an accumulation of related data. For example, the collection of bug reports in a single file is a dataset. In machine learning applications, datasets are usually partitioned into two subsets:

1. **Training Dataset:** A collection of data that is used as input to a supervised machine learning algorithm to train the machine learning model is known as a training dataset.

2. **Testing Dataset:** A collection of data used to measure the accuracy of a supervised machine learning model is known as a testing dataset.

Suppose, in the bug assignment example, 80% of the bug reports can be used to train the system, while the other 20% may be used to test the system.

### 2.3.1 Data Preparation Plan

Data are required to train and evaluate a proposed system. The data preparation plan elaborates on the procedures for collecting data. Data preparation affects the reliability of predictions. So, it is essential to process and arrange the data correctly in order to have reliable predictions from a machine learning model. There are four stages associated with preparing a dataset for use with a machine learning algorithm.

1. **Collect data:** From available sources, collect data for the proposed system. For instance, bug reports may be collected from Bugzilla, which is an open-source bug repository of Mozilla.
2. **Choose data:** Choose data that is relevant to the problem. Selecting data that are not relevant to the problem may hamper the training of the learning model. For example, there are different statuses of bug reports such as NEW, RESOLVED, and VERIFIED [1]. For the bug assignment problem, only the RESOLVED bug reports are considered as these bug reports contain the developer ID who has solved the issue. Bug reports having other statuses are filtered out.
3. **Process data:** After selecting the data, the next steps are to format, clean and sample the data.

Step 1 is to **format** the data. The raw data which has been gathered from various sources may not be in a useful configuration for use by the machine learning algorithm. The data may need to be formatted according to the needs of the machine learning algorithm.

For example, an input file may contain the bug reports for the bug assignment system, each of which are text descriptions. A machine learning algorithm is not capable of working with text data. The necessary data must be converted to a format such as integers, strings or other compatible formats for machine learning algorithms in an array or vector.

Step 2 is to **clean** the data. There might be some missing or incomplete data in the raw dataset. Also, there might be different forms of the same data which need to be mapped to the original form. A process might be necessary to fix the missing data or remove the incomplete data or map all the different forms of data to the original form to make the dataset consistent and helpful for the issue being addressed.

For instance in the bug report example, there might be many spelling mistakes. Sometimes people will write “cannot run” or “can not run” or “can’t run”. These are the same messages, so these strings must be updated to “can not run” so that the machine learning model can identify that all of the messages are conveying the same message. This is a way to clean the data.

Step 3 is to **sample** the data. There might be more data available than is required. The use of more data will increase the running time and memory requirements of the learning approach. It may be appropriate to use a reduced dataset which will represent the input dataset. This is called sampling.

For instance, suppose there are 10000 bug reports in the dataset. Instead of taking 10000 bug reports, a subset consisting of 4000 bug reports may be used to train the model and predict the developers for bug reports. If the subset (4000 bug reports) was chosen carefully the trained model will be able to predict developers for the entire dataset (10000 bug reports). This is called data sampling.

4. **Transform data:** The last step of data preparation is to transform the data. The data must be changed according to the requirements of the learning algorithm and the



knowledge of the problem domain. Scaling, feature aggregation, and feature decomposition are the most fundamental transformation techniques. Scaling is performed to convert every feature value into a similar scale in the preprocessed data. Sometimes multiple features in the preprocessed data are aggregated into a single feature to be efficient for the machine learning model. This procedure is known as feature aggregation. Alternatively, to make the machine learning model more productive, a complex feature may be required to be divided into multiple features. This method is called feature decomposition.

For example, the bug report data can be stored in arrays or vectors. After removing the stop words, there will be words that convey a message. During the process of text mining, the nonessential words that are removed from a sentence are called stop words [16]. For example, there can be a bug report having a description, “The program is crashing”. After removing the stop words from “The program is crashing”, “program” and “crashing” will be in the array. Bug reports which are already solved by a developer will have the ID of the developer. The machine learning model for this problem will be trained on RESOLVED bug reports. A feature vector must be created, consisting of the list of words after removing the stop words. The numeric feature value of a word is determined by dividing the frequency of a word in the dataset by the total number of words in the dataset. In this way, the numeric feature values of all the words in the dataset will be in the same range which is known as data scaling.

## **2.4 Classification Algorithms**

The process which predicts the class of data is known as classification. Classification algorithms are also known as classifiers. A classifier learns from training data about how a new data relates to any class. Classification algorithms are widely used in machine learning. The machine learning approaches which we used to build our models are discussed below.

### 2.4.1 Decision Tree

The decision tree algorithm is one type of supervised learning algorithm. The primary goal of the decision tree algorithm is to provide a model that learns straightforward decision rules. The value of a leaf node is predicted using rules which are inferred from the feature data. A vector of feature values is provided as input in the decision tree and the output is a single value.

A greedy approach is used in the decision tree to partition the instances into distinct classes. In the tree, interior nodes are the features or attributes. The values of the features or attributes are labelled on the branches starting from the nodes. Each level of the tree splits the data while each leaf node has a specific value. Based on the entropy, the decision tree partitions the dataset into subsets. The purpose of the entropy is to determine the best attribute from the set of attributes. The entropy of an attribute can be calculated using the following equation [47]:

$$E(S) = \sum_{i=1}^c -P_i \log_2 P_i.$$

Here,  $P_i$  is the frequency probability of an attribute for class “i”. Entropy of two variables can be computed using [47]:

$$E(S, T) = \sum_{i \in T} P_i E_i.$$

Where  $E_i$  is the entropy of an attribute “i”. The best attribute to use for an internal node can be found using information gain. The difference between the original information requirement and the new information requirement is known as information gain [29]. The equation of information gain is as follows:

$$Gain(S, T) = Entropy(S) - Entropy(T)$$

Information gain is determined based on how much information an attribute can obtain about the class label. The higher the value of information gain of an attribute, the more rel-

event it is for the classification. For every subset, this procedure is repeatedly implemented. This is known as a recursive partition. The recursive procedure stops when all the nodes of a subset have the same value as the target variable.

For example [13], using a decision tree we might decide whether we want to play badminton on a specific day. The decision to play or not will depend on the weather, humidity, and wind.

Table 2.1 in Section 2.3 shows the previous ten days of weather observations. We are not considering temperature for this example. The decision of whether to play or not can be made using this table. However, it will be difficult to decide if none of the rows of the table matches the condition of the day in question. This decision can be reached with the help of a decision tree based upon the information considered most relevant.

Decision tree is used for classification. The last column of table 2.1 is “Play?” which represents the class Yes and No.

Entropy needs to be computed to find the root node. Table 2.2 shows the frequency of the classes (Yes and No).

Table 2.2: Frequency of the classes (Yes and No)

<b>Play</b>	
<b>Yes</b>	<b>No</b>
5	5

The entropy of column Play is:

$$\begin{aligned}
 E(\text{Play}) &= E(5, 5) \\
 &= -\left(\frac{5}{10} \cdot \log_2 \frac{5}{10}\right) - \left(\frac{5}{10} \cdot \log_2 \frac{5}{10}\right) \\
 &= 1
 \end{aligned}$$

For other attributes such as Weather, Humidity, and Wind, entropy needs to be calculated after each split. Table 2.3 shows the frequency of Yes and No for Play depending on

Weather.

Table 2.3: Frequency of Yes and No for Play depending on Weather

Weather	Play		Total
	Yes	No	
Sunny	1	2	3
Cloudy	3	0	3
Rainy	1	3	4

$E(\text{Play}, \text{Weather})$  can be calculated as follows:

$$E(\text{Play}, \text{Weather}) = P(\text{Sunny}) \cdot E(\text{Sunny}) + P(\text{Cloudy}) \cdot E(\text{Cloudy}) + P(\text{Rain}) \cdot E(\text{Rain}) \quad (2.1)$$

Now the entropy of Sunny is:

$$\begin{aligned} E(\text{Sunny}) &= E(1, 2) \\ &= -\left(\frac{1}{3} \cdot \log_2 \frac{1}{3}\right) - \left(\frac{2}{3} \cdot \log_2 \frac{2}{3}\right) \\ &= 0.918 \end{aligned}$$

Then, the entropy of Cloudy is:

$$\begin{aligned} E(\text{Cloudy}) &= E(3, 0) \\ &= -\left(\frac{3}{3} \cdot \log_2 \frac{3}{3}\right) - \left(\frac{0}{3} \cdot \log_2 \frac{0}{3}\right) \\ &= 0 \end{aligned}$$

And, the entropy of Rainy is:

$$\begin{aligned} E(\text{Rainy}) &= E(1, 3) \\ &= -\left(\frac{1}{4} \cdot \log_2 \frac{1}{4}\right) - \left(\frac{3}{4} \cdot \log_2 \frac{3}{4}\right) \\ &= 0.811 \end{aligned}$$

Replacing the values of E(Sunny), E(Cloudy) and E(Rainy) in equation 2.1:

$$\begin{aligned} E(\text{Play, Weather}) &= \left(\frac{3}{10} \cdot 0.918\right) + \left(\frac{3}{10} \cdot 0\right) + \left(\frac{4}{10} \cdot 0.811\right) \\ &= 0.599 \end{aligned}$$

Table 2.4 shows the frequency of Yes and No for Play depending on Humidity.

Table 2.4: Frequency of Yes and No for Play depending on Humidity

Humidity	Play		Total
	Yes	No	
High	3	4	7
Normal	2	1	3

E(Play, Humidity) can be calculated as follows:

$$E(\text{Play, Humidity}) = P(\text{High}) \cdot E(\text{High}) + P(\text{Normal}) \cdot E(\text{Normal}) \quad (2.2)$$

Now the entropy of High is:

$$\begin{aligned} E(\text{High}) &= E(3, 4) \\ &= -\left(\frac{3}{7} \cdot \log_2 \frac{3}{7}\right) - \left(\frac{4}{7} \cdot \log_2 \frac{4}{7}\right) \\ &= 0.985 \end{aligned}$$

Then, the entropy of Normal is:

$$\begin{aligned} E(\text{Normal}) &= E(2, 1) \\ &= -\left(\frac{2}{3} \cdot \log_2 \frac{2}{3}\right) - \left(\frac{1}{3} \cdot \log_2 \frac{1}{3}\right) \\ &= 0.918 \end{aligned}$$

Replacing the values of E(High) and E(Normal) in equation 2.2:

$$\begin{aligned} E(\text{Play}, \text{Humidity}) &= \left(\frac{7}{10} \cdot 0.985\right) + \left(\frac{3}{10} \cdot 0.918\right) \\ &= 0.965 \end{aligned}$$

Table 2.5 shows the frequency of Yes and No for Play depending on Wind.

Table 2.5: Frequency of Yes and No for Play depending on Wind

Wind	Play		Total
	Yes	No	
Weak	3	1	4
Strong	2	4	6

E(Play, Wind) can be calculated as follows:

$$E(\text{Play}, \text{Wind}) = P(\text{Weak}) \cdot E(\text{Weak}) + P(\text{Strong}) \cdot E(\text{Strong}) \quad (2.3)$$

Now the entropy of Weak is:

$$\begin{aligned} E(\text{Weak}) &= E(3, 1) \\ &= -\left(\frac{3}{4} \cdot \log_2 \frac{3}{4}\right) - \left(\frac{1}{4} \cdot \log_2 \frac{1}{4}\right) \\ &= 0.811 \end{aligned}$$

Then, the entropy of Strong is:

$$\begin{aligned} E(\text{Strong}) &= E(2, 4) \\ &= -\left(\frac{2}{6} \cdot \log_2 \frac{2}{6}\right) - \left(\frac{4}{6} \cdot \log_2 \frac{4}{6}\right) \\ &= 0.918 \end{aligned}$$

Replacing the values of E(Weak) and E(Strong) in equation 2.3:

$$\begin{aligned} E(\text{Play}, \text{Wind}) &= \left(\frac{4}{10} \cdot 0.811\right) + \left(\frac{6}{10} \cdot 0.918\right) \\ &= 0.875 \end{aligned}$$

Information gain after splitting using the Weather attribute is given by:

$$\begin{aligned} \text{Gain}(\text{Play}, \text{Weather}) &= E(\text{Play}) - E(\text{Play}, \text{Weather}) \\ &= 1 - 0.599 \\ &= 0.401 \end{aligned}$$

After splitting using the Humidity attribute, information gain is given by:

$$\begin{aligned} \text{Gain}(\text{Play}, \text{Humidity}) &= E(\text{Play}) - E(\text{Play}, \text{Humidity}) \\ &= 1 - 0.965 \\ &= 0.035 \end{aligned}$$

Information gain during the split using the Wind attribute is as follows:

$$\begin{aligned} \text{Gain}(\text{Play}, \text{Wind}) &= E(\text{Play}) - E(\text{Play}, \text{Wind}) \\ &= 1 - 0.875 \\ &= 0.125 \end{aligned}$$

Information gain of Weather is the maximum after the split among all the attributes. For that, Weather is the root node. Figure 2.4 shows the tree after first split.

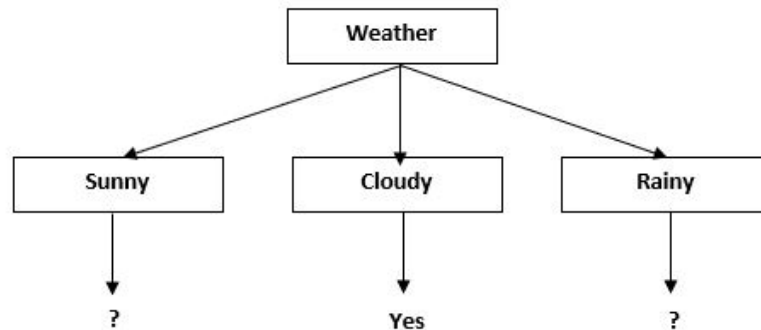


Figure 2.2: Decision tree after the first split.

From figure 2.6 we can see that, we have one leaf node. This leaf node is generated from Cloudy as it only corresponds to Yes class. The tree needs to be split further using Sunny and Rainy attributes.

Sunny attribute can be split using Humidity or Wind. Table 2.6 shows the sub table for Sunny.

Table 2.6: Sub table for Sunny attribute.

Weather	Humidity	Wind	Play?
Sunny	Normal	Strong	Yes
Sunny	High	Weak	No
Sunny	High	Strong	No

From table 2.6, we can see that Sunny can be split using Humidity as it creates a ho-



homogeneous group. When Humidity is Normal, the class is Yes and when humidity is High, the class value is No. Wind can not be used for splitting Sunny, as Wind have not created homogeneous group. Figure 2.4 shows the tree after splitting using Humidity.

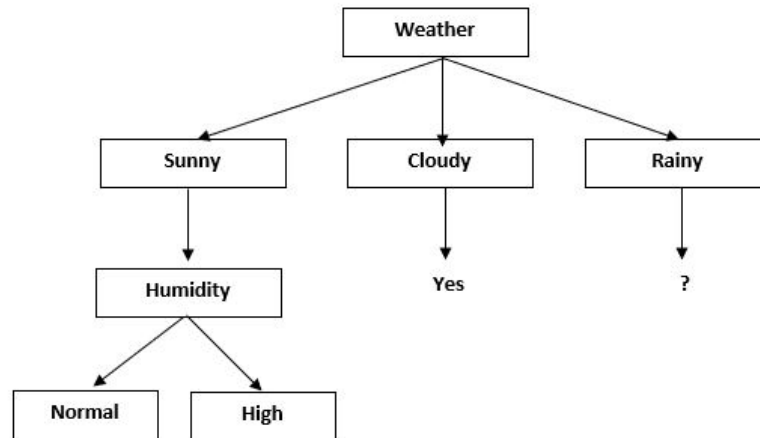


Figure 2.3: Decision tree after split using Humidity attribute.

Same procedure can be applied to split Rainy attribute. Rainy attribute can be split using Humidity or Wind. Table 2.7 shows the sub table for Rainy weather.

Table 2.7: Sub table for Rainy attribute.

Weather	Humidity	Wind	Play?
Rainy	High	Weak	Yes
Rainy	Normal	Strong	No
Rainy	High	Strong	No

We can see from table 2.7, Rainy can be split using Wind as it creates a homogeneous group. When Wind is Weak, the class is Yes and when Wind is strong, the class value is No. Figure 2.4 shows the final decision tree after splitting using Wind.

### 2.4.2 Naïve Bayes

The naïve Bayes approach is the most straightforward machine learning algorithm. Like other machine learning algorithms, this model also uses feature values. Based on instances, the feature values are created. An instance is one complete row of the data table containing

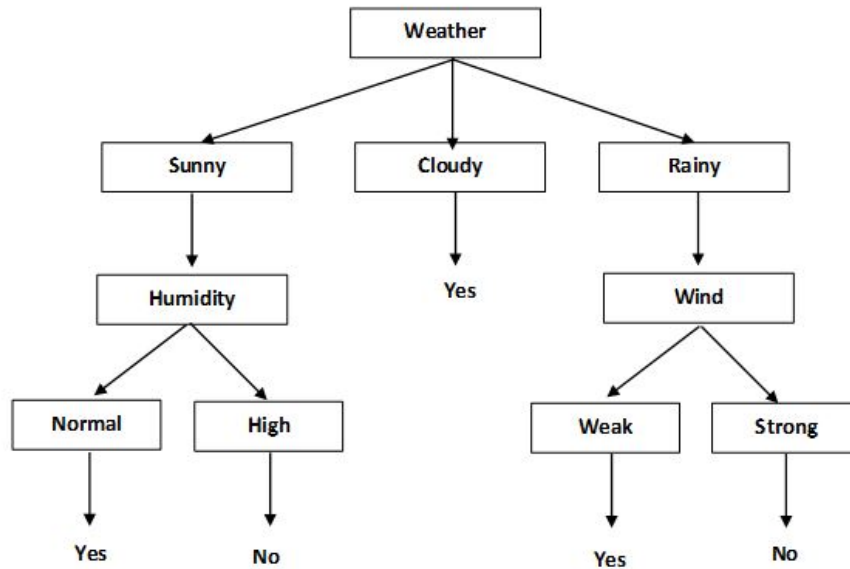


Figure 2.4: Final decision tree.

all the attribute values. For example, one bug report is an instance in the data set containing 500 bug reports. Class labels are also used in this approach. The naïve Bayes approach is composed of a group of algorithms based on a common principle. The naïve Bayes classifiers expect that values of the attributes are independent of one another. Unlike other machine learning approaches, the benefit of naïve Bayes is that this approach can estimate the parameters which are essential for classification from a small amount of training data. The probability of the features or attributes in a naïve Bayes model is processed utilizing Bayes' theorem. The conditional probability of Bayes' theorem can be expressed as follows [35]:

$$p(C_k|x) = \frac{p(C_k) p(x|C_k)}{p(x)}$$

Here,

- $p(C_k|x)$  = the probability of an instance  $x$  belonging to a class  $C_k$ ,
- $p(x|C_k)$  = the probability of  $x$  given that the class is  $C_k$ ,

- $p(C_k)$  = the prior probability of class  $C_k$ , and
- $p(x)$  = the probability that an instance  $x$  will be observed within the population of all programs.

Table 2.1 in Section 2.3 lists the weather conditions. The attributes of Table 2.1 are weather, temperature, humidity, and wind. Using naïve Bayes, we may determine whether we will play badminton on a specific day or not.

- The estimate of the weather being sunny given that we are playing badminton is

$$P(\text{sunny}|p) = \frac{1}{5}.$$

- The estimate of the weather being sunny given that we are not playing badminton is

$$P(\text{sunny}|n) = \frac{2}{5}.$$

- In the same way, for cloudy weather,  $P(\text{cloudy}|p) = \frac{3}{5}$ ,  
and  $P(\text{cloudy}|n) = \frac{0}{5}$ .

- For rainy weather,  $P(\text{rainy}|p) = \frac{1}{5}$  and  $P(\text{rainy}|n) = \frac{3}{5}$

- The estimate of the temperature being hot given that we are playing badminton is

$$P(\text{hot}|p) = \frac{2}{5}.$$

- The estimate of the temperature being hot given that we are not playing badminton is

$$P(\text{hot}|n) = \frac{2}{5}.$$

- If the temperature is mild,  $P(\text{mild}|p) = \frac{3}{5}$  and  $P(\text{mild}|n) = \frac{2}{5}$ .

- In the same manner, for cool temperature,  $P(cool|p) = \frac{0}{5}$   
and  $P(cool|n) = \frac{1}{5}$ .

- The estimate of humidity being high given that we are playing badminton is

$$P(high|p) = \frac{3}{5}.$$

- The estimate of humidity being high given that we are not playing badminton is

$$P(high|n) = \frac{4}{5}.$$

- For normal humidity,  $P(normal|p) = \frac{2}{5}$  and  $P(normal|n) = \frac{1}{5}$ .

- The estimate of playing with weak wind is  $P(weak|p) = \frac{3}{5}$ .

- The estimate of not playing with weak wind is  $P(weak|n) = \frac{1}{5}$ .

- For strong wind,  $P(strong|p) = \frac{2}{5}$  and  $P(strong|n) = \frac{4}{5}$ .

- Thereg probability of playing is  $P(p) = \frac{5}{10} = \frac{1}{2}$ .

- The probability of not playing is  $P(n) = \frac{5}{10} = \frac{1}{2}$ .

- Now suppose there is a new instance,  $X = \langle rain, hot, high, weak \rangle$ . Then the probability of sample  $X$  being classified as playing is:

$$P(X|p) \cdot P(p) = \frac{P(rainy|p) \cdot P(hot|p) \cdot P(high|p) \cdot P(weak|p) \cdot P(p)}{P(X)} \quad (2.4)$$

- The probability of sample  $X$  being classified as not playing is:

$$P(X|n) \cdot P(n) = \frac{P(rainy|n)(hot|n) \cdot P(high|n) \cdot P(weak|n) \cdot P(n)}{P(X)} \quad (2.5)$$

$P(X)$  is common in both probabilities of playing and the probability of not playing. For that, we can ignore  $P(X)$  and calculate the proportional probabilities using equation 2.4 and 2.5.

$$\begin{aligned} P(X|p) \cdot P(p) &= P(\text{rainy}|p) \cdot P(\text{hot}|p) \cdot P(\text{high}|p) \cdot P(\text{weak}|p) \cdot P(p) \\ &= \frac{1}{5} \cdot \frac{2}{5} \cdot \frac{3}{5} \cdot \frac{3}{5} \cdot \frac{1}{2} \\ &= 0.0144 \end{aligned}$$

$$\begin{aligned} P(X|n) \cdot P(n) &= P(\text{rainy}|n) \cdot P(\text{hot}|n) \cdot P(\text{high}|n) \cdot P(\text{weak}|n) \cdot P(n) \\ &= \frac{3}{5} \cdot \frac{2}{5} \cdot \frac{4}{5} \cdot \frac{1}{5} \cdot \frac{1}{2} \\ &= 0.0192 \end{aligned}$$

These numbers can be converted to probability by making the sum equal to 1 since  $P(X|p) \cdot P(p) + P(X|n) \cdot P(n) = 1$ .

$$\begin{aligned} P(X|p) \cdot P(p) &= \frac{0.0144}{0.0144 + 0.0192} \\ &= 0.43 \end{aligned}$$

$$\begin{aligned} P(X|n) \cdot P(n) &= \frac{0.0192}{0.0192 + 0.0144} \\ &= 0.57 \end{aligned}$$

Since  $P(X|n) \cdot P(n) > P(X|p) \cdot P(p)$  or  $0.57 > 0.43$ , sample X will be classified as not playing. So, when the weather is rainy, the temperature is hot, humidity is high, and the wind is weak, we predict that there will be no playing of badminton.

### 2.4.3 Random Forest

Random forest is a collection of decision trees. The decision trees in the forest are created using random data samples of the original dataset. This approach is known as random forest because samples of data are selected randomly from the dataset to create the decision trees to build the forest [23].

Random forest is an ensemble method. The machine learning approach that merges multiple instances of base classifier to construct a predictive model is known as an ensemble method [20]. The decision tree is the base classifier for the random forest. Multiple instances of base classifier are built using a decision tree by training the models with the help of different samples of data and different samples of features. For every instance of base classifier, sample data (subset of data) are taken from the original dataset, with the data samples being different for each instance of base classifier. Every instance of base classifier has different data samples to train the classifier. Also, different feature samples are collected for each instance of classifier from the original feature set. Thus all the instances of base classifier models or decision trees are built based on their data samples and feature samples. When test data is evaluated using the random forest, all the decision trees produce the classification result. For a binary classification problem, majority voting is used to derive the outcome. For continuous values, the average of the decision trees' outcomes is taken as the final classification result of the random forest [23]. Random forest shows lower variance than the decision tree. It also tends to show fewer errors during prediction for test data than the decision tree.

Figure 2.5 is showing an example of a random forest. It shows that the  $n$  data and feature samples ( $SD_1 : SF_1, SD_2 : SF_2, \dots, SD_n : S_n$ ) are created from an original training dataset

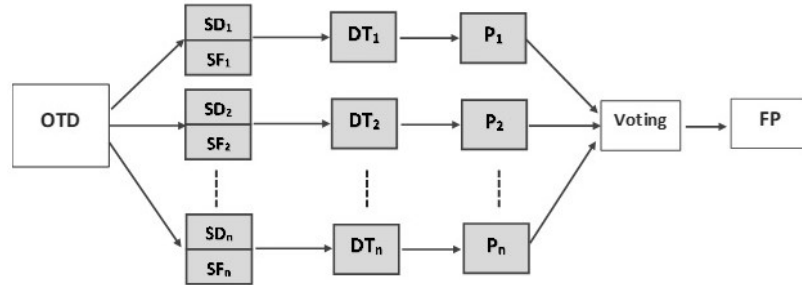


Figure 2.5: Random forest

(OTD). Multiple instances of decision tree classifiers ( $DT_1, DT_2, \dots, DT_n$ ) make predictions ( $P_1, P_2, \dots, P_n$ ) using the sample datasets and feature sets ( $SD_1 : SF_1, SD_2 : SF_2, \dots, SD_n : SF_n$ ). Next, the predicted results go through a voting procedure. The highest voted or ranked predicted result is considered the final prediction.

#### 2.4.4 K Nearest Neighbor Algorithm

K nearest neighbor (KNN) is one of the most popular machine learning approaches. A point  $m$  is assigned to a class with the most similar points within  $K$  nearest to  $m$  in the training set [3]. The nearest point is evaluated based on distance. The most commonly used distance function in KNN is Euclidean distance. Equation 2.6 shows the formula for Euclidean distance [14].

$$Distance = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (2.6)$$

Here  $n$  is a positive integer representing the number of samples in the dataset. When  $K = 1$  in KNN, the new data point is assigned to the class of that single nearest neighbor. The function is locally evaluated, and until the classification, every one of the calculations is carried out slowly. To make the algorithm increasingly proficient and successful, it is valuable to assign weight to the contribution of neighbors. By assigning weights, the nearest neighbor will contribute more than a distant one.

For example, we have a training dataset displayed in Table 2.8.

Assume that there is a new data point  $X = 6$  and  $Y = 9$ . Using KNN with  $K = 3$ , we can determine the class of the new data point.

Table 2.8: Training data for KNN.

<b>X</b>	<b>Y</b>	<b>Class</b>
1	2	0
4	5	0
7	8	1
10	11	1
13	15	1

The first point of the training set is (1,2). The Euclidean distance between (1,2) and (6,9) can be calculated using the following steps:

$$\begin{aligned} \text{SquaredDifference}_1 &= (X_1 - X_2)^2 \\ \text{SquaredDifference}_2 &= (Y_1 - Y_2)^2 \end{aligned} \quad (2.7)$$

or

$$\begin{aligned} \text{SquaredDifference}_1 &= (6 - 1)^2 = 25 \\ \text{SquaredDifference}_2 &= (9 - 2)^2 = 49 \end{aligned} \quad (2.8)$$

Summation of the squared differences is:

$$\begin{aligned} \text{SumSquaredDifference} &= \text{SquaredDifference}_1 + \text{SquaredDifference}_2 \\ \text{SumSquaredDifference} &= 25 + 49 = 74 \end{aligned} \quad (2.9)$$

To find the distance, we need to find the square root of SumSquaredDifference:

$$\begin{aligned} \text{Distance} &= \sqrt{\text{SumSquaredDifference}} \\ \text{Distance} &= \sqrt{74} \\ \text{Distance} &= 8.60 \end{aligned} \quad (2.10)$$



The distances between the new data point and the data points in the training dataset are listed in Table 2.9.

Table 2.9: Distance from training data to new data.

<b>X</b>	<b>Y</b>	<b>Class</b>	<b>Distance</b>
1	2	0	8.60
4	5	0	4.47
7	8	1	1.41
10	11	1	4.47
13	15	1	9.21

Suppose that we take three nearest neighbors for  $K = 3$ . The three nearest neighbors of the new point (6,9) are listed in Table 2.10.

Table 2.10: Three nearest neighbors of new point (6,9).

<b>X</b>	<b>Y</b>	<b>Class</b>	<b>Distance</b>
4	5	0	4.47
7	8	1	1.41
10	11	1	4.47

For binary classification,  $K$  must be an odd number in  $K$ -nearest neighbor so that there can not be a tie in the voting of nearest neighbors. In our example, among the three nearest neighbors, two are from class 1 and the other is from class 0. By using majority voting, the new data point (6,9) is classified in class 1.

KNN can be used for multi-class (more than two classes) classifications. For multi-class classification, the value of  $K$  can be any positive odd integers. For binary classification, the majority vote is used. But for multi-class classification, the plurality voting is considered [47]. In the majority voting, one class needs to have more than half of the vote. However, in the plurality voting, the classifier having the most votes is considered as the winner. If a tie occurs during the voting, then the new input data can be randomly classified into one of the classes, which got majority votes in the tie. Apart from random selection,  $K$ 's value can be changed dynamically until one class gets the majority votes [3].

### 2.4.5 Logistic Regression

Logistic regression is a prominent machine learning approach for binary classification. It is a predictive analysis algorithm, as it predicts based on probability [24].

Logistic regression uses a statistical function called a logistic function. This function can map any real values within 0 and 1. The formula of the logistic function is given below [24]:

$$y = \frac{1}{1 + e^{-x}} \quad (2.11)$$

Here,

- $y$  = predicted output (a value between 0 and 1),
- $e$  = base of the logarithm, and
- $x$  = numerical values that need to be transformed between 0 and 1.

The formula used in logistic regression is as follows:

$$y = \frac{e^{B_0 + B_1 \times x}}{1 + e^{B_0 + B_1 \times x}} \quad (2.12)$$

where

- $x$  = the vector of all of the scaled features,
- $y$  = predicted output,
- $B_0$  = bias or intercept term, and
- $B_1$  = coefficient for single input value.

The coefficient values ( $B$ ) for input data ( $x$ ) are learned from the training data. Maximum likelihood estimation is used to estimate these coefficients. Maximum likelihood estimation is commonly used in various machine learning approaches. In binary classification, a coefficient value close to 1 would identify one class, and the coefficient value close to 0 (less than 0.5) would identify the other class [24].

Suppose a model can predict whether a computer programmer is an expert (probability greater or equal to 0.5) or a beginner (probability less than 0.5) based on the length of the computer program written by the programmer. Assume that there is a computer program with 200 lines. We also assume that the coefficients  $B_0 = -50$  and  $B_1 = 0.5$  are learned from the training data. Then the probability that the computer programmer is an expert given the length of the program is 200 or more lines can be calculated using equation 2.12

$$y = \frac{e^{-50+0.5 \times 200}}{1 + e^{-50+0.5 \times 200}} = 0.73 \quad (2.13)$$

As the value of  $y = 0.73$  is greater than 0.5, we can predict that the programmer is an expert. On the other hand, if the probability were to be less than 0.5, that would mean the programmer is likely a beginner.

#### 2.4.6 Bagging Classifier

A bagging classifier is an ensemble method. Bagging is the acronym of bootstrap aggregating [5]. A bagging classifier creates multiple predictors using a base classifier (Decision Tree). These predictors are the multiple instances of base classifier. To create  $n$  predictors, the original dataset is sampled  $n$  times. The  $n$  predictors are trained using the  $n$  samples of the datasets. Next test data is classified using the  $n$  predictors. Finally, the prediction is aggregated using voting or averaging. Developing  $n$  models or predictors using a sample dataset is called bootstrap, and voting to get the final prediction is known as aggregating. Random forest is also an ensemble method where the base classifier is a decision tree by default. In a random forest, both data and features are sampled to train the instances of base classifiers. However, in the bagging classifier, the base classifier can be any machine learning algorithm. Also, only data are sampled to train the multiple instances of base classifier.

Figure 2.6 illustrates the working procedure of the Bagging classifier. It demonstrates that the original training dataset (OTD) randomly creates  $n$  sample datasets ( $SD_1, SD_2, \dots, SD_n$ ).

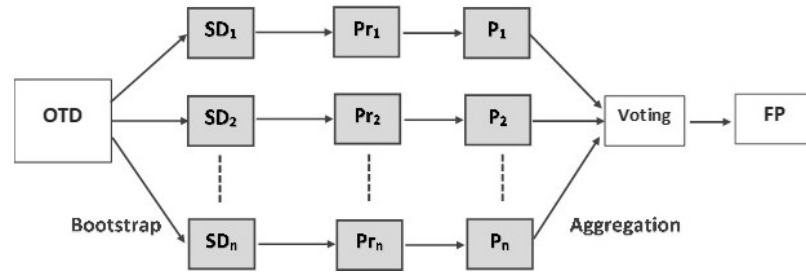


Figure 2.6: Bagging classifier methodology.

Then models (predictors) are trained using the sample datasets. Next the predictors ( $Pr_1, Pr_2, \dots, Pr_n$ ) make predictions ( $P_1, P_2, \dots, P_n$ ) using the sample datasets ( $SD_1, SD_2, \dots, SD_n$ ). Finally, the predicted results are combined using voting. The highest voted or ranked result is considered the final prediction.

## 2.5 Evaluation of Machine Learning Algorithms

Evaluation of the machine learning model is vital. Techniques to evaluate a machine learning approach are described below.

### 2.5.1 Holdout Method

In the holdout method, the dataset is partitioned into two sets (training and testing) independent of each other. The training set is used to train and develop the learning model. The learning models are evaluated by testing them using the test set. The test set is independent of the training set. The split of data between the training set and test set can be of variable percentage [22].

It is beneficial to use the holdout method when there is a constraint of resources. The holdout method requires less computational power and time to execute as it splits the dataset once. However, due to a lack of data (not entire dataset), the performance evaluation is subject to higher variance. The training partition may contain more data of one class label than another, which can make the model biased towards the class which has more data. This drawback (being biased) can be overcome by using the cross-validation method [11].

### 2.5.2 Cross-Validation

In this method, a dataset is divided into many folds where each fold contains the same number of data items [26]. If there are  $n$  folds, then a machine learning model will be trained using  $n - 1$  folds and tested using the remaining fold. This will repeatedly happen  $n$  times. Each time a different fold is used for testing. Accuracy is measured by taking the average of all the iterations.

The cross-validation method overcomes the drawback of the holdout method. Models are trained and tested using all of the data in  $n$  iterations. There is no possibility of bias towards a class using a cross-validation method. During the cross-validation method, data are split into  $n$  sets and models are trained in each iteration using  $n - 1$  sets and models are tested using remaining 1 set. As there are  $n$  iterations, models are trained and tested  $n$  times. Due to this, cross-validation requires more computational cost and time to execute than the holdout technique.

### 2.5.3 Performance Measurement

The performance measurement of a machine learning model is essential to assess the accuracy of the solution. Performance measures are used to assess how well the model performs. There are standard performance measurement units to measure the efficiency of a learning model, including accuracy, precision, recall and F-measure [22], [18].

A model's performance can also be measured using a confusion matrix. The confusion matrix shows how well a classifier correctly classifies samples according to their classes. A function defined in Python (`confusion_matrix()`) can be used to calculate the confusion matrix in the format shown in Table 2.11.

Here, the dimension of the confusion matrix is  $n \times n$  where  $n$  is the number of classes. Confusion matrix contains the values of TN, FP, FN and TP. For convenience, we have added the row and the column headings as well as the last column (N, P) and last row (N', P', P+N) with the matrix to show the row and column totals. In our work there are two

Table 2.11: Confusion matrix

Expertise	Beginner (Predicted)	Expert (Predicted)	Total
Beginner (Actual)	TN	FP	N
Expert (Actual)	FN	TP	P
Total	N'	P'	P+N

classes: expert and beginner. Expert-written programs are considered positive samples and beginner-written programs are treated as negative samples. The notations TP, TN, FP, FN, P, N, P', N' shown in Table 2.11 are explained as follows:

- **True positive (TP)** are positive samples that are correctly classified as positive by the model. In Table 2.11, true positive refers to the expert-written programs which are predicted as expert-written.
- **True negative (TN)** are negative samples that are correctly classified as negative by the model. In Table 2.11, true negative indicates the beginner-written programs which are identified as beginner-written.
- **False positive (FP)** are negative samples that are misclassified by the model as positive samples. In our work false positive refers to the beginner-written programs which are misclassified as expert-written.
- **False negative (FN)** are positive samples that are misclassified by the model as negative samples. In Table 2.11, expert-written programs which are misclassified as beginner-written are considered false negatives.

In Table 2.11, P refers to the total number of actual positive samples, N indicates the total number of actual negative samples, P' specifies the number of predicted positive samples ( $P' = TP + FP$ ) and N' is the number of predicted negative samples ( $N' = TN + FN$ ). Accuracy, precision, recall and F-measure can be described based on the confusion matrix as follows:

- **Accuracy** is the number of instances (as a percentage) that are classified accurately by the machine learning model. The accuracy of a model can be calculated as:

$$Accuracy = \frac{TP + TN}{P + N} \quad (2.14)$$

- **Precision** measures the fraction of relevant instances among all the predicted instances. Precision can be determined using the following equation:

$$Precision = \frac{TP}{TP + FP} \quad (2.15)$$

- Recall is the percentage of correctly classified positive data instances. Using the following formula, recall can be computed as

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{P} \quad (2.16)$$

- **F-measure** is the harmonic mean of precision and recall [22]. To measure F-measure, both precision and recall are required. F-measure is calculated as

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (2.17)$$

Accuracy is the preferred measure when the dataset is balanced, and F measure is the better measurement when the dataset is imbalanced [15]. As an example of how the metrics may be used, suppose a phrase “machine leaning” is searched in a search engine. The engine may return 50 web pages where 20 of them are relevant. In this case,  $TP = 20$  and  $FP = 30$ . Then we might say that the precision is:

$$Precision = \frac{TP}{TP + FP} = \frac{20}{20 + 30} = 0.4$$

The search engine may return 60 relevant pages of machine learning where the search phrase “machine learning” is not present. Here,  $TP = 20$  and  $FN = 60$ . The recall will be:

$$Recall = \frac{TP}{TP + FN} = \frac{20}{20 + 60} = 0.25$$

The F-measure can be calculated using the precision and recall:

$$\begin{aligned} F - measure &= \frac{2 \times Precision \times Recall}{Precision + Recall} \\ &= \frac{2 \times 0.4 \times 0.25}{0.4 + 0.25} \\ &= 0.31 \end{aligned} \tag{2.18}$$

## 2.6 Python

Python is a programming language and data analysis tool. It is an open-source, object-oriented programming language. Machine learning algorithms can be implemented using different programming languages such as C, C++, R, Javascript, and Python. However, Python tends to be preferred as it is an open-source programming language and has a vast library of packages available. A few of the Python libraries that we used are described in Appendix A.

### 2.6.1 Feature Selection

Computer programs are text files. For machine learning these files must be represented using numeric feature values. All the features present in the dataset may not be significant, and the inclusion of non-significant features may lead to a time-consuming learning process as well as inferior results [18]. Thus selecting a relevant set of features is essential as it may make the learning process faster. Also, using a small and relevant set of features may lead to a better result compared to the result that used all available features [48]. To select the most



useful features we used two feature selection methods defined in Python. These methods are as follows:

1. **infogain:** infogain is a statistical method that computes the information gain of features with respect to the class. Information gain is determined using the difference between the original information about the proportion of classes and the new information based on the identified useful attributes. The following equation determines the information gain [22], [56]:

$$InfoGain(Class, Attribute) = Entropy(Class) - Entropy(Class|Attribute) \quad (2.19)$$

The information gain value is determined based on how much information an attribute can obtain about the class label. Here information is the relevance between the attribute and class label. The higher the value of information gain of an attribute, the more relevant it is for the classification. Attributes that have higher values of information gain are considered more useful, and this subset of original features can enhance the performance of learning models.

2. **selectkbest:** This is a univariate feature selection method defined in scikit-learn [41]. A univariate feature selection method evaluates each feature by determining the strength of the relationship between a feature and the class variable. selectkbest is defined using the univariate statistical test. The Chi-square test is performed in the selectkbest method [54]. Scores are calculated for all the features depending on the classes. Features with higher score values are considered to be more important for the classification. This method returns the K highest scoring features among all features.

## 2.7 Related Work

Sociolinguistics is the study of language with social factors, including the differences in region, class, and occupational dialect as well as gender differences [37]. Sociolinguistics

plays a vital role in analyzing differences in the use of natural languages based on social variables [40], [28].

Existing studies show that machine learning can be used to determine the gender of an author of a natural language text. Argamon et al. [48] investigated gender differences in English literature. They experimented with English literature in the British National Corpus (BNC), including books and articles in the BNC. The algorithm was used to train a learning model on that literature. The resulting model was 90% accurate. The same group of researchers later implemented a model using a support vector machine and applied this to French literature [49]. The accuracy of that model was 90%.

Information about the influence of the author's gender on computer programs was determined by Naz and Rice [37] using machine learning techniques. Open-source implementations of machine learning algorithms were applied to C++ programs to retrieve the information about the features affected by the author's gender. M. M. H. Rafee [44] used machine learning and statistical techniques to assess the impact of an author's gender and region on computer programs. In this study, the accuracy of the models in predicting the author's gender was 83.1%, and the accuracy in predicting the author's region was 92.5%.

Coding style analysis is another way to identify the author of a computer program. This type of analysis differentiates programs written by a particular author. This area of research is also known as authorship analysis. Krsul et al. [27] used supervised and unsupervised machine learning techniques to analyze the authorship of computer programs. They used Gaussian Likelihood Classifiers and MLP Neural Networks to identify the author of a program with more than 98% accuracy for a specific set of programmers. Steven et al. [6] used statistical analysis to find the author of a program. They used the combination of n-grams with text similarity measures to find the author of computer programs written using C. The accuracy of their model for finding the original author was 67%.

Chris Piech et al. [42] presented a machine learning methodology based on K-Means clustering to show the progress of how novice programmers work on their programming

assignments. As well Qin et al. [43] developed a model to find the maturity of a software project. In this study time series machine learning was implemented on the traits data (commit, open-issue, closed-issue, contributor) of GitHub to develop the model.

## **2.8 Summary**

Sociolinguistic characteristics can be analyzed using machine learning approaches. The effect of sociolinguistic characteristics on the text document and computer program were carried out on earlier research works. We have discussed the related research works and background studies, including sociolinguistics, machine learning, data, and Python. The next chapter will describe the required methodologies to categorize computer programs based on the author's programming experience.

# Chapter 3

## Methodology

In this chapter, we discuss data collection, transformation, and feature selection process. We also illustrate the procedure to transform the text-based dataset into a numerical dataset. .

### 3.1 Data Collection

Data collection is a significant step towards classifying computer programs. We collected computer programs from Codeforces.com and built a database containing computer programs with sociolinguistic characteristics such as the author’s gender, and experience. Codeforces.com is a programming contest website which holds contests on a regular basis. Five problems are given in a contest, and contestants are required to solve them within two hours. The score for submission depends on time and number of attempts. A contestant gets a higher score when a problem is solved within a short time and with fewer submission attempts. Different programmers submit their solutions (i.e. computer programs) of programming contest problems. Other users can access Codeforces.com to collect these computer programs. Codeforces.com also stores identifying information of different programmers and their contributions. For this research, source code is essential as we want to analyze the computer programs using a machine learning model to determine whether a computer program is written by a beginner or an expert programmer.

#### 3.1.1 Data Collection Process

Codeforces.com has an API [10] which facilitates the retrieval of user and submission information. We retrieved the list of users who participated in a contest using the API. We

proceeded with the users whose country and gender information was available. For each user, submission records were retrieved using the API.

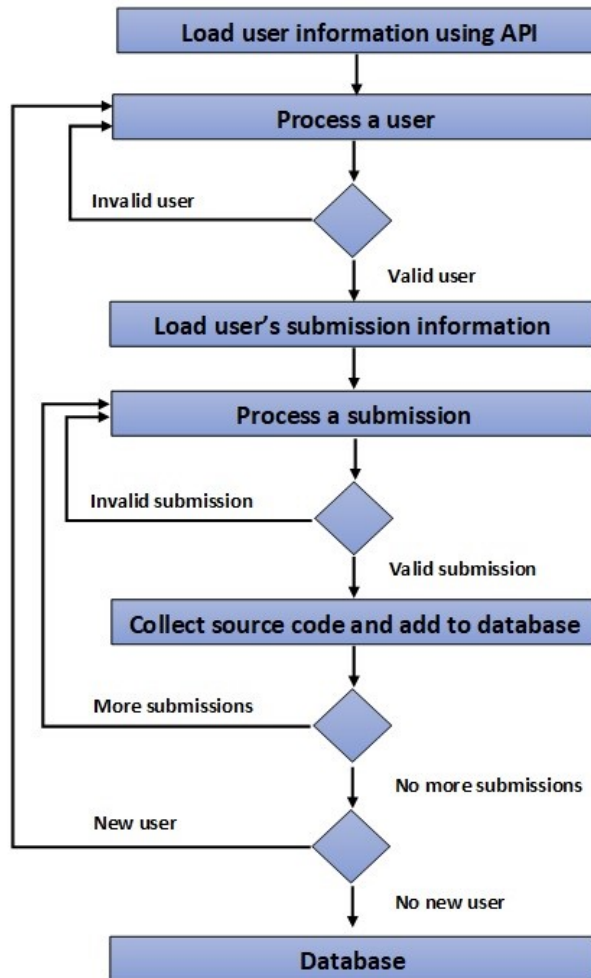


Figure 3.1: Data collection process.

Unfortunately, the API did not facilitate collection of the programs. However, in a user's profile, there are several tabs which contain contests, groups and submissions for that user. Programs are stored in the submission tab of a user's profile. The source code of a submission can be retrieved by clicking on the submission id.

Programs were collected using two methods. The first method used the Selenium web driver [10], which can open a user's submission record in a web page and then open the pop up by following the link embedded in the submission id. The submission page contains different information on a submission such as submission number, time of submission, the programming language used, problem name, verdict, and running time. The source code could be retrieved from the pop-ups. If the submission id did not contain a link, then the Selenium web driver could not retrieve the source code of that submission.

In the second method, a web link (URL) was manually created using the user and submission information. This method is used when the Selenium web driver was unable to retrieve the source code. The program could be retrieved if this manually created URL led to a web page containing it.

The collected source code was added to the database, along with user and submission information. For each user, 50 programs were added to the database. This number was used because Codeforce.com returns 50 submissions for a user in one web page. During dataset creation, each user's number of programs got reduced from 50 because programs written using C++ were only used. Also, each user's number of programs reduced again as the dataset got balanced over experience, gender, and region.

### **3.1.2 Drawbacks of the Data Source**

While Codeforces.com provides the user data and computer programs that we required, there are a few drawbacks to this data source:

#### **Time required for collection**

Collecting source code using the Selenium web driver is a time-consuming process.

Sometimes the Selenium web driver encountered issues with processing the pop-ups. When there is an issue, the pop-ups must be refreshed to complete the data collection process. Also, the alternate method that constructs URL to obtain the source code sometimes did not lead to a web page containing the source code. Therefore, time would be wasted to create and visit the link without getting the source code.

### **Style restrictions**

Codeforces contains a large number of programs. However, these programs are submitted for programming contests, which may not reflect “normal” coding. Contest coders may not follow the style guidelines as the contestants’ main aim is to solve a problem in an optimized manner.

### **Repeatability**

The last drawback is that programs are retrieved from HTML pages. We may need to change the data collection program developed for this if there are any changes in the Codeforces website. For example, programs are stored in a pop-up, the link of which is embedded in the submission id. If Codeforces.com ceases embedding the link in the submission id, we will need to change our data collection program.

## **3.2 Dataset Creation**

Dataset creation is a significant part of pursuing a sociolinguistic analysis of computer programs using machine learning.

We have collected data from Codeforces to create a data repository for our research work. Collected data are stored in our research repository using the CSV file format. The CSV file containing the Codeforces data has 52263 rows. There are 45002 computer programs written in C++, 2703 programs written in Python, and 4558 programs written in Java. These computer programs are written by programmers from 77 different countries. The collected data contains 23 attributes [10], which are shown in Table 3.1.

All the attributes (fields) except Rating, Submission id, and Gender Probability are of

Table 3.1: Attributes of the collected data from Codeforces.

Field	Type
Handle	String
First Name	String
Last Name	String
Gender	String
Gender Probability	Float
Country	String
City	String
Organization	String
Contribution	String
Rank	String
Rating	Int
Max Rank	String
Max Rating	Int
Registered	String
Submission id	Int
Source Code	String
Programming Language	String
Problem Name	String
Participant Type	String
Time	String
Year	Int
Month	Int
Day	Int

string type. Rating and Submission Id are of integer type, and Gender Probability is of float type. The handle is a unique string denoting the user's login id. First Name and Last Name are the strings containing the user's name. Gender defines whether the programmer is male or female. The gender probability is given by an API named genderize.io that takes the first name and returns the gender of the input name. It uses the number of instances they had for that name and the number of names that were male or female. Gender probability contains a float value ranging from 0.0 to 1.0. So a gender probability of 1.0 male means that for the given name, 100% of their samples were male, and a gender probability of 0.8 female means that 80% of people with that name were female and 20% were male. The country,



city, and organization reflect the geographical information of the programmer. Contribution shows a programmer's contribution to the Codeforces community. A programmer can write blogs and make comments in the Codeforces community. Other programmers can vote (like or unlike) those blogs and comments. The value of the contribution field can be positive or negative and is determined by considering votes (likes and unlikes) for a user's blogs and comments. An upvote (like) increases the contribution by 2 points and a downvote (unlike) decreases the contribution by 1 point.

Rating is an integer type attribute that contains a value from 0 to 4000. Rating is calculated based on the Codeforces.com rating system, which is close to the Elo rating [31]. The attribute Rank indicates the expertise of the programmer. Max Rank and Max Rating stores the best rank and rating achieved by a programmer on the Codeforces.com web site. These attributes are used to assess the expertise of a programmer. Registered defines the time when a programmers joined Codeforces.

Submission information of a program is represented by Submission id, Source Code, Programming Language, Participant Type, Time, Year, Month, Day. The participant type denotes the role of the programmer. In Codeforces, a programmer can participate in a contest and submit the source code of a problem of that contest. In this case, the participant type is a contestant. Also, a programmer can solve any problems posted in Codeforces and submit the source code. In this scenario, the programmer is considered as a problem solver but not a contestant. Every submission has a unique submission id. The time of the submission is broken into Time, Year, Month, and Day.

For our research, we are using 12 attributes from those listed in Table 3.1. The selected attributes are listed in Table 3.2. Handle, first name, last name, and gender are the user's basic information. Gender probability is needed to determine the gender as gender information is not available in Codeforces.com. The country represents the region of the programmer. The attributes rank and rating can determine the programmer's experience. The fields submission id and problem name aid to identify duplicate entries. Programming

Table 3.2: Selected attributes of the dataset.

<b>Field</b>	<b>Type</b>
Handle	String
First Name	String
Last Name	String
Gender	String
Gender Probability	Float
Country	String
Rank	String
Rating	Int
Submission id	Int
Source Code	String
Programming Language	String
Problem Name	String

language is required as we are focusing on programs written using C++. In addition, the field source code contains the computer program on which we will perform our analysis. We are not considering the rest of the 24 attributes listed in Table 3.1 as they are less relevant to our research. For example, city information is not needed in our dataset. We are focusing on the region: Europe and Asia. We can determine the region from the country. We do not need the city attribute from Table 3.1 to determine the region. However, we keep more columns than needed in our dataset because we may need them in future experiments.

We prepared our data set in CSV format. There is information for 952 programmers in our dataset. For each programmer, we used eight programs. We considered only the programs written using C++ during dataset creation, and the programs using Python and Java got discarded. Moreover, we balanced the dataset over experience, gender, and region, which lead to eight programs from each user. These eight programs were randomly selected from each user.

In Codeforces.com, a programmer’s experience is reflected by the rating and the rank. Programmers with a rating of more than 2900 are the most experienced in Codeforces.com [30]. The rank of the most experienced programmers is Legendary Grandmaster. Program-

mers having ratings from 0 to 1199 are given the rank of Newbie. There are ten categories of programmers based on their rating in Codeforces.com [30]. We decided to divide these ten categories into two larger groups: expert and beginner. For this work, we group programmers as beginners if their rating was less than 1600 as Codeforces used 1600 as the “novice” category when our research work started. Now the rating for “novice” changed on Codeforces to 1900. Those with ratings of 1600 or higher were considered expert programmers.

### **3.2.1 Balanced Data Set**

The performance of machine learning approaches can deteriorate if the dataset is unbalanced. If the data are not taken from all the categories equally, then it is unbalanced. In our work we are focus on three sociolinguistic features: the author’s region, gender and experience. We are attempting to classify computer programs into two classes: expert and beginner. The dataset is considered balanced when it consists of an equal number of programs written by experts and beginners. The dataset is unbalanced when it contains more programs written by one class than another. The issue of unbalanced data can be resolved by under-sampling or oversampling of data [12].

To build our dataset we used under-sampling. In under-sampling, data are randomly removed from the majority class of the dataset to make it balanced. This procedure is known as random under-sampling. The class, with a higher number of elements of classification data, is known as the majority class. The class with a lesser number of elements is referred to as the minority class. Suppose there are two classes of programmers: expert and beginner. There are 70 expert programmers and 30 beginner programmers in a dataset of 100 programmers. In this scenario, the expert class is the majority class as it has more programmers than beginner class. The beginner class is the minority class in this classification data. One of the significant drawbacks of under-sampling is that important information might be deleted from the dataset, which may affect the classification process. We used

under-sampling instead of oversampling as duplicate data of the minority class are created to make the dataset balanced in oversampling. Oversampling may lead to a more accurate result, but this accuracy is based on duplicate data [12]. We instead randomly removed data from the dataset to make it balanced over our three areas of interest: experience, gender and region.

We collected half of our data from expert programmers and the remaining half from the beginners in order to make the dataset balanced when considering programmer’s experience. We are basing our measure of experience on the rating system used in Codeforces. We also made our data set balanced over gender and region by taking equal numbers of programs written by males and females as well as taking those programs equally from two regions, Asia and Europe. For Asia programs are collected from Bangladesh, China, Hong Kong, India, Indonesia, Iran, Iraq, Japan, Jordan, Kazakhstan, Kyrgyzstan, Malaysia, the Philippines, Singapore, Taiwan, Tajikistan, United Arab Emirates, Uzbekistan, and Vietnam. For Europe, collected programs are written by the programmers of Andorra, Armenia, Austria, Belarus, Bulgaria, Croatia, Czechia, France, Georgia, Germany, Greece, Italy, Latvia, Lithuania, Macedonia, Poland, Portugal, Romania, Russia, Serbia, Spain, Sweden, Switzerland, Turkey, Ukraine, and the United Kingdom. Details of the dataset are shown in Table 3.3.

Table 3.3: Information about the dataset.

<b>Region</b>	<b>Expert: rating <math>\geq</math> 1600</b>		<b>Beginner: rating <math>&lt;</math> 1600</b>	
	<b>Male</b>	<b>Female</b>	<b>Male</b>	<b>Female</b>
<b>Europe</b>	119	119	119	119
<b>Asia</b>	119	119	119	119
<b>Total</b>	238	238	238	238
<b>Total Programmers: 952</b>				

Our dataset contains programs from 952 programmers. For each programmer, eight programs are stored in the dataset. Therefore there are 7616 programs in the dataset.

### 3.3 Document Representation

Machine learning approaches require input in the form of a vector of numeric feature values. The models then provide discrete or nominal values as output. In our research, input feature values are numeric, and class labels are nominal. The two-class labels are Beginner and Expert. All the collected C++ programs are text documents. Steven [10] collected the data from Codeforces.com and stored it in a shared data repository in CSV format. We created the dataset in a CSV file by taking programs from the shared repository. Using a python script, we created a CPP file for each program. Then a tool named LOCMetrics [39] was applied to each C++ program to retrieve the values of selected features. The selected features are listed in Table 3.4. Next, each C++ program was represented by the retrieved feature values. Lastly, the Python library's normalized function was applied to the feature values to bring all of the numeric feature values into the same range (0 to 1). In this way, the text-based dataset was transformed into a numerical dataset. This process is illustrated in Figure 3.2.

The information representing each computer program was stored in a row in the dataset. A row of data consists of 15 columns containing 15 feature values of a program. A new column representing the class label (beginner or expert) was added to each computer program row in the experimental dataset. Then this dataset was used to train and test the machine learning models using Python.

### 3.4 Feature Selection

Rafee [44] used machine learning to assess the impact of sociolinguistic characteristics, including the author's gender and region on computer programs. From this study, the accuracy of the models in predicting the author's gender was 83.1%, and the accuracy in predicting the author's region was 92.5%.

We selected fifteen features based on [44] for our research work, as our research goal is also to categorize the computer programs based on sociolinguistic characteristics. However,

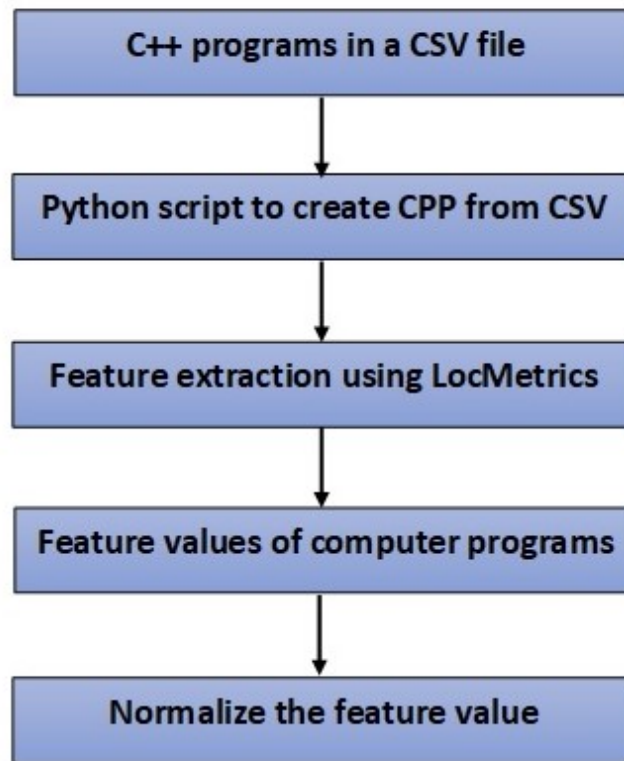


Figure 3.2: Document representation process.

our focus was to determine whether a computer program is written by a beginner programmer or by an experienced programmer. The list and the description of the selected features are given in Table 3.4.

We use the C++ program listed below to illustrate the features listed in Table 3.4.

```
#include <iostream>
using namespace std;

int add(int x, int y) //function to add two integer values
{
    return x+y;
}
```

Table 3.4: The fifteen features selected to characterize the programs in our dataset.

<b>Feature</b>	<b>Description</b>
Lines of code (LOC)	Number of lines including number of source code lines, comment lines and blank lines
Source lines of code (SLOC)	Number of code lines excluding blank lines and comments lines
Percentage of source lines of code (SLOC%)	Percentage of source line code with respect to total number of lines
Blank lines of code (BLOC)	Number of empty lines
Percentage of blank lines of code (BLOC%)	Percentage of empty lines with respect to total number of lines
Comment lines of code (CLOC)	Number of comment lines. Comment lines are not executable by the compiler. Comment lines are used to describe a code
Percentage of comment lines of code (CLOC%)	Percentage of non-executable lines with respect to total number of lines
Mixed lines of code with both source and comments (C_SLOC)	Number of lines having both executable and non-executable lines
Percentage of mixed lines (C_SLOC%)	Percentage of lines having both executable and non-executable lines with respect to total number of lines
Total words in all comments (CWORD)	Total number of words in all the comments of a program
Physical executable lines of code (SLOC_P)	Number of lines of a source code excluding comments
Logical executable lines of code (SLOC_L)	Number of executable statements
The number of functions defined (Functions)	Number of functions defined in a program
Lines of code in functions (FLOC)	Total number of lines from all the functions of a program
Average lines of code per functions (FLOC average)	Number of lines of codes in functions divided by number of functions of a program

```

//main function
int main ()
{
    int sum;

```

```
    sum = add (5 ,10);  
    cout<<sum;  
    return 0;  
}
```

In the code above,

- lines of code (LOC) = 16,
- source lines of code (SLOC) = 13,
- blank lines of code (BLOC) = 2,
- comment lines of code (CLOC) = 1,
- mixed lines of code with both source and comments (C\_SLOC) = 1,
- total words in all comments (CWORD) = 8,
- physical executable lines of code (SLOC\_P) = 13,
- logical executable lines of code (SLOC\_L) = 8,
- the number of functions defined (Functions) = 2,
- lines of code in functions (FLOC) = 7, and
- average lines of code per functions (FLOC average) = 3.5.

### 3.5 Summary

This chapter explained the data collection process, the feature selection method, and the procedure to transform the text-based dataset into a numerical dataset. These were the necessary steps to categorize computer programs based on the author's programming experience. In the next chapter, we will discuss the experiments and the results.



# Chapter 4

## Experiments and Results

In this chapter we discuss the various experiments which were conducted. We also present the results of the experiments. The chapter concludes with the discussion of the results and their implications.

### 4.1 Preliminaries

To summarize the previous chapter (Chapter 3: Methodology), for our research we collected computer programs from Codeforces.com. This is a programming contest website. Programmers from different regions participate in the programming contests hosted by Codeforces.com. Programmers submit their solutions to problems on this website, and the submitted programs are viewable to all. The submitted source codes were written in different programming languages. However, we considered only programs written in C++ for our research. We categorized computer programs based on the author's programming experience using machine learning approaches. Computer programs that were treated as text documents were converted to numeric feature values. We extracted features' numeric values from a text document using the Locmetrics tool. These feature values were used as input for machine learning models. By analyzing the computer programs, machine learning models classified the programmers in two classes: expert and beginner.

We conducted six experiments. In each experiment we used six machine learning algorithms. The library functions of Python for these machine learning algorithms are given below:

- (i) Decision Tree (`tree.DecisionTreeClassifier()`),
- (ii) Naïve Bayes (`GaussianNB()`),
- (iii) Random Forest (`RandomForestClassifier()`),
- (iv) K Nearest Neighbor (`KNeighborsClassifier()`),
- (v) Logistic Regression (`LogisticRegression()`), and
- (vi) Bagging and classification via Decision Tree (`BaggingClassifier()`).

These machine learning algorithms were applied to a training set to build the various classification models. The models were then evaluated using metrics such as accuracy, precision, recall and F measure. The models were trained using 7616 computer programs with class labels expert and beginner. Short descriptions of the six experiments are given below:

- (i) In experiment 1, we used 15 features, and evaluated the models using a cross-validation technique, as described in Chapter 2.
- (ii) In experiment 2, the number of features was reduced from 15 to 7 using `infogain`, a tool from the Python library that we described in Chapter 2. As in experiment 1, the models were evaluated using a cross-validation technique.
- (iii) In experiment 3, the number of features was reduced from 15 to 4 using a univariate selection method (`select4best`), a tool from the Python library that we described in Chapter 2. Again, the performance of the models was determined using a cross-validation technique.
- (iv) All computer programs were again categorized using 15 features in experiment 4. This experiment is similar to experiment 1; however, the evaluation method is different as the holdout method was used instead of a cross-validation technique. The holdout method is described in Chapter 2.

- (v) In experiment 5, the top seven features ranked by infogain were used to classify the programs. The trained models were then evaluated on the test set using the holdout method.
- (vi) In experiment 6, select4best was applied to select the 4 best features. The select4best method is described in Chapter 2. Machine learning algorithms were then applied to the training set to build the models. Again the models were evaluated on the test set using a holdout method.

In summary, all six experiments used 7616 computer programs. The performances of the machine learning models developed in experiments 1, 2, and 3 were evaluated using the cross-validation technique, and the performances of machine learning models developed in experiments 4, 5 and 6 were evaluated using the holdout method.

#### 4.1.1 Programming Environment

All experiments were carried out on an HP Pavilion machine. The details of the device are listed in Table 4.1.

Table 4.1: Details of the test environment.

Processor	Operating System	RAM	HDD
Intel Core i5 7200U CPU 2.70 GHz	Windows 10	8 GB	1 TB

Python 3.8.0 was used to apply the machine learning approaches to build the models. Microsoft Excel© was used for the statistical analysis of the features.

Python is a popular programming language for the application of machine learning approaches. It is an open-source platform. As indicated earlier, various functions from the Python libraries were used in our research work. In addition, some filters defined in the Python library were used in the data processing, and various built-in evaluation metrics of the Python library were used to evaluate the performance of the models. Two feature

selection methods in Python, `infogain` and `selectkbest` (using  $k=4$ ), were used for feature reduction.

Microsoft Excel<sup>©</sup> was used to analyze the features statistically. The goal was to determine which features are significant in identifying an expert-written program versus a beginner-authored program. The details of the statistically significant features are given in Chapter 5.

### 4.1.2 Parameter Settings

In the experiments, various built-in libraries of Python were used. The description of these libraries is given below.

- **Randomize:**

We used a randomize filter [46] to randomly shuffle the dataset before the applying machine learning algorithms. The first 3808 programs listed in the dataset were written by beginners, and the remaining 3808 programs were written by experts. We shuffled the dataset so that each data point have their own impact on the model. Shuffling makes sure that the model is not getting biased from the same points before them. For each execution, the dataset was shuffled in a new order before applying machine learning algorithms.

- **Normalize:**

The Normalize [8] method was applied to prepare the data for machine learning. We used the normalized method to scale the values of all features into the same range (0 to 1). If the feature values are not normalized, then the features having higher values dominate the result. For example, in our dataset, there were two features lines of code (LOC) and the number of functions defined (Functions). Suppose there might be a computer program that has 1000 lines of code and 2 functions. The feature value of LOC (1000) is larger than the number of functions defined (2). The LOC will influence the result if the data are not normalized. In this scenario, the number

of functions defined is also vital for determining whether a beginner or an expert programmer has written the program. Both LOC and Functions must be scaled to a single unit such as 0 to 1 so that both features have equal importance in the result. Therefore, data must be normalized before applying machine learning approaches.

- **infogain:**

In experiment 2 and experiment 5, infogain was used to reduce the number of features. Less relevant features and noisy data were removed from the decision making process. Also, infogain returns the information gain value of each feature with respect to its class. We selected seven features having the top seven infogain values from all of features.

- **selectkbest:**

selectkbest is a univariate selection method. The univariate feature selection method evaluates each feature by determining the strength of the relationship between the feature and the class variable. In our research, we used  $k=4$ . That means, the 4 best features were returned.

- **Cross-Validation:**

We applied a cross-validation [41] method to evaluate the six machine learning models. We have used 5 fold cross-validation. The dataset was divided into 5 folds where a fold is a group of equal-sized data [26]. Models were evaluated in 5 iterations. In each iteration, the training set contained four folds of data and the test set contained one fold of data. Details of the cross-validation method are given in Chapter 2.

## 4.2 Experiments

### 4.2.1 Experiment 1

The goal of experiment 1 was to categorize the computer programs based on the author's programming experience. The experiment was conducted on 7616 computer programs. In

this experiment, 15 features were used. Six machine learning algorithms were applied on the dataset to build the models. The steps of experiment 1 are displayed in Figure 4.1.

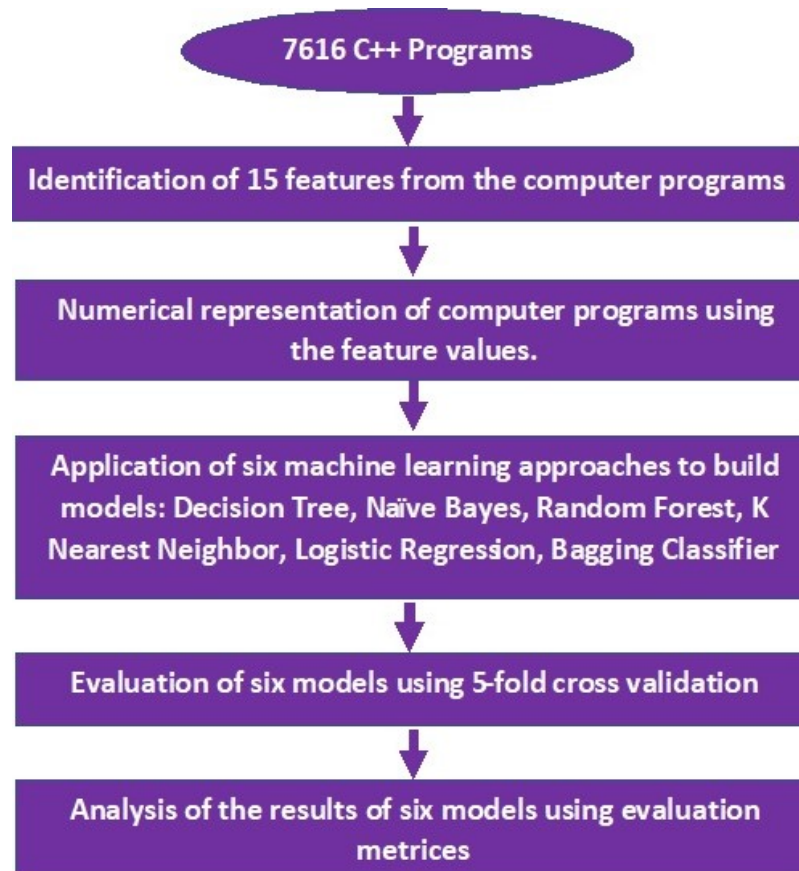


Figure 4.1: Steps for experiment 1.

Machine learning models classified the computer programs in two classes: expert and beginner. The results are listed in Section 4.3.

### 4.2.2 Experiment 2

Numeric feature values of 15 features were retrieved, and each computer program was represented by these feature values. infogain was applied to each feature to get the infogain values of each feature with respect to class labels. Based on the infogain values, the top seven features were selected. This reduced the number of features from 15 to 7. The steps carried out for experiment 2 are shown in Figure 4.2.

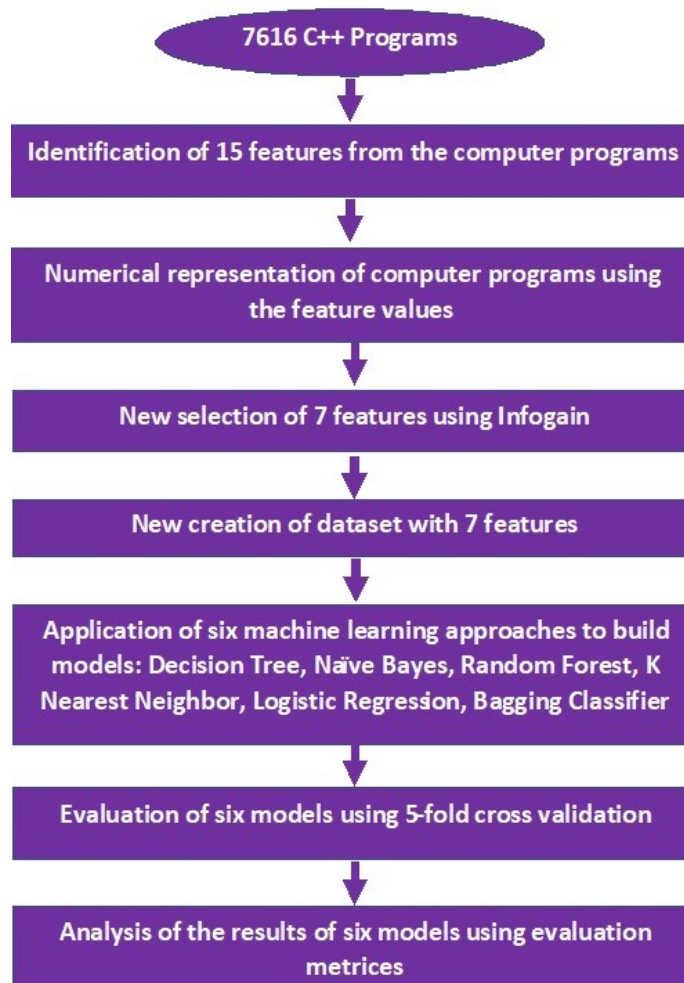


Figure 4.2: Steps for experiment 2.

The top 7 features according to rank were:

- Percentage of source lines of code (SLOC%)
- Percentage of blank lines of code (BLOC%)

- Percentage of comment lines of code (CLOC%)
- Average lines of code per functions (FLOC AVERAGE)
- Lines of code (LOC)
- Physical executable lines of code (SLOC-P)
- Source lines of code (SLOC)

A new dataset was created using the numeric values of these 7 features for each program. Then six machine learning algorithms were applied. The performance of these models was evaluated using 5 fold cross-validation technique. The results are described in Section 4.3.



### 4.2.3 Experiment 3

In experiment 3, computer programs were analyzed to determine whether a program is written by a beginner or an expert. The steps of experiment 3 are given in Figure 4.3.

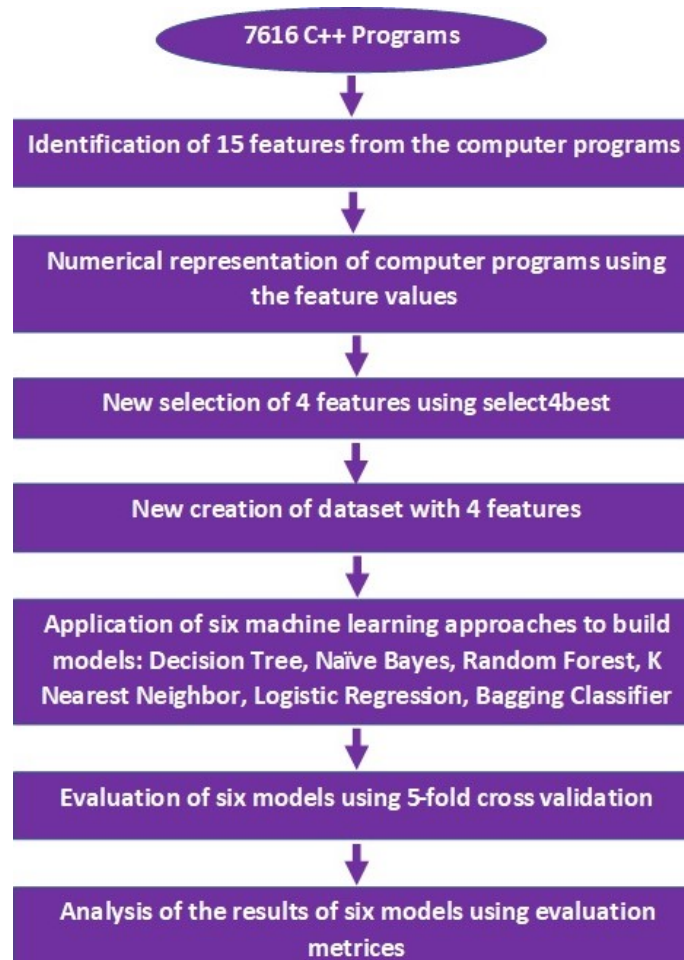


Figure 4.3: Steps for experiment 3.

selectkbest (using  $k=4$ ) method was applied to the 15 original features with respect to their class labels. selectkbest returned 4 features which were:

- Lines of code (LOC)
- Physical executable lines of code (SLOC-P)
- Logical executable lines of code (SLOC-L)
- Source lines of code (SLOC)

A new dataset was created using the feature values for LOC, SLOC-P, SLOC-L and SLOC. Then the six machine learning algorithms were applied to build classifying models. These models were evaluated using a 5 fold cross-validation technique. The results are shown in Section 4.3.

#### 4.2.4 Experiment 4

In experiment 4, computer programs were analyzed using 15 features. The steps for experiment 4 are shown in Figure 4.4.

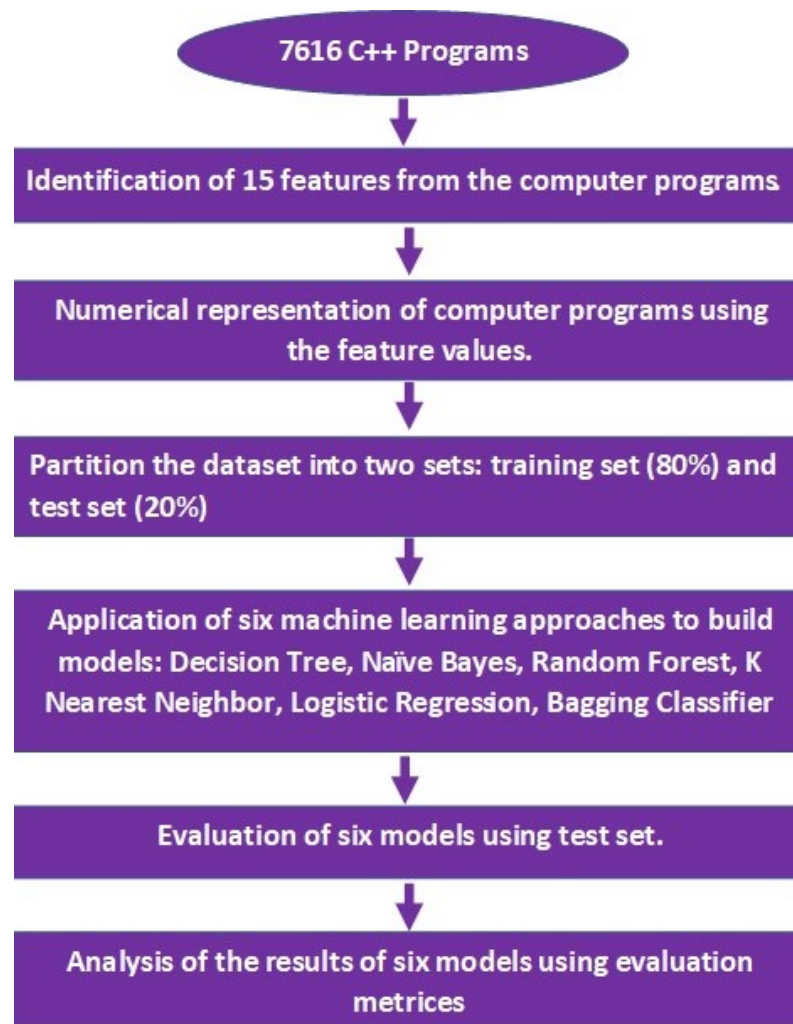


Figure 4.4: Steps for experiment 4.

Experiment 4 is similar to experiment 1. However, the evaluation method of experiment 4 is different from experiment 1. Instead of 5 fold cross-validation, the holdout method was used to evaluate the machine learning models. The dataset was partitioned into two sets: training set (80% of data) and test set (20% of data). The six machine learning algorithms were applied on the training set to build the models. Then the models were evaluated on the test set. Results are given in Section 4.3.

### 4.2.5 Experiment 5

In experiment 5, computer programs were analyzed using seven features to find out the author's experience. The steps for experiment 5 are shown in Figure 4.5.

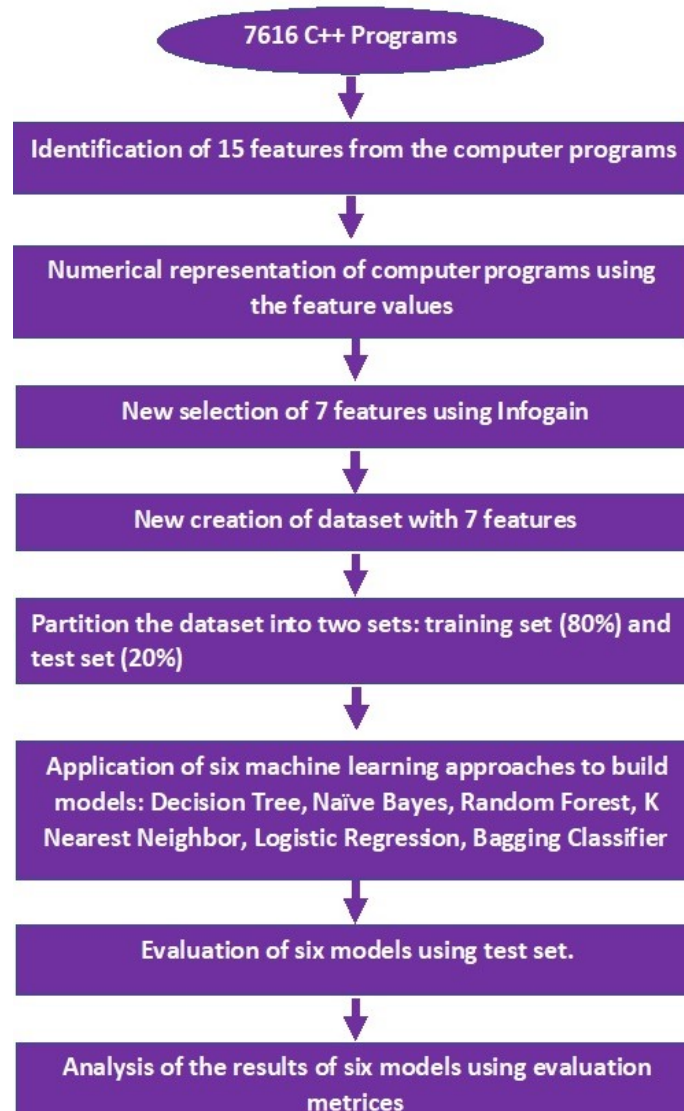


Figure 4.5: Steps for experiment 5.

The number of features was reduced from 15 to 7 using infogain. The selected features were SLOC%, BLOC%, CLOC%, FLOC AVERAGE, LOC, SLOC-P, and SLOC. A new dataset was created using the values of the selected 7 features. Then the six machine learning algorithms were applied on the training set to build the models. The models were evaluated on the test set using the evaluation metrics. Section 4.3 contains the result of

experiment 5.

#### 4.2.6 Experiment 6

In this experiment, four features were used. These features (LOC - lines of code, SLOC-P - physical executable lines of code, SLOC-L - logical executable lines of code, SLOC - source lines of code) were selected using the selectkbest (with k=4) method. The steps for experiment 6 are displayed in Figure 4.6.

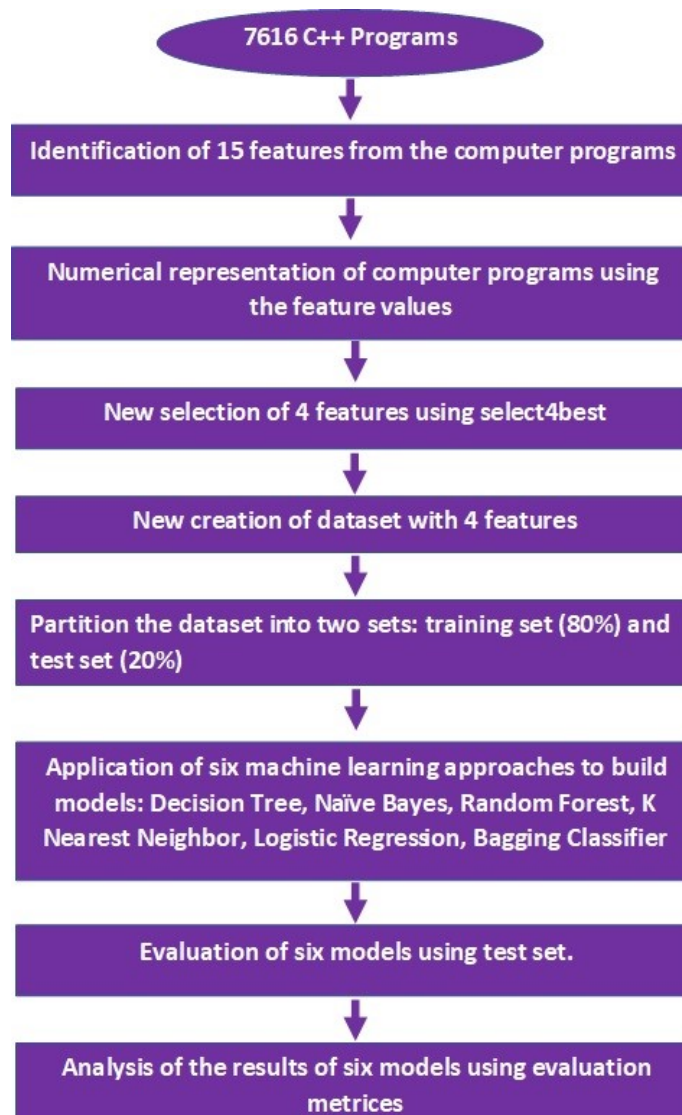


Figure 4.6: Steps for experiment 6.

The new dataset containing values of LOC, SLOC-P, SLOC-L and SLOC were divided

into training and test set. machine learning algorithms were applied on the training set to build the models. Then the developed models were tested on the test set using the hold-out method. Results of experiment 6 are given in Section 4.3

### 4.3 Results

The results of the six experiments are discussed in this section. As a reminder, the goal of all six experiments was to categorize computer programs based on the author's programming experience. All the experiments were conducted on a dataset consisting of 7616 computer programs. In these experiments the class labels were beginner and expert. The performance of the machine learning models developed in experiments 1, 2, and 3 were evaluated using a cross-validation technique, and the performance of machine learning models developed in experiments 4, 5 and 6 were evaluated using a holdout method.

#### 4.3.1 Experiment 1

Experiment 1 used 15 features and evaluated the results using the 5-fold cross-validation technique. The performance of the resulting models is reported in Table 4.2.

Table 4.2: Performance evaluation of six models with 15 features.

<b>Model</b>	<b>Accuracy (%)</b>	<b>Precision(%)</b>	<b>Recall(%)</b>	<b>F-Measure(%)</b>
Decision Tree	64%	64%	64%	63%
Naïve Bayes	50%	50%	99%	66%
Random Forest	64%	67%	56%	61%
K Nearest Neighbor	64%	65%	63%	64%
Logistic Regression	65%	69%	55%	61%
Bagging Classifier	67%	68%	62%	65%

The model developed using bagging classifier had an accuracy of 67%. That means the bagging classifier model accurately classified 67% of the data. The logistic regression model also performed well with an accuracy of 65%, unlike naïve Bayes, which classified only 50% of the data accurately.

Considering the F-measure score, naïve Bayes model performed well (F-measure = 66%) as it had a very high recall (99%). However, the bagging classifier model also had a good F-measure score (65%).

Confusion matrices of the six models developed in experiment 1 are shown in Tables 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8. Confusion matrices illustrate how many programs are correctly classified to their class label (expert or beginner). Confusion matrices are represented using the format shown in Table 2.11 in Section 2.5.3.

Table 4.3: Confusion matrix for decision tree with 15 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	2450	1358	3808
<b>Expert (Actual)</b>	1392	2416	3808
<b>Total</b>	3842	3774	7616

Table 4.4: Confusion matrix for naïve Bayes with 15 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	39	3769	3808
<b>Expert (Actual)</b>	56	3752	3808
<b>Total</b>	95	7521	7616

Table 4.5: Confusion matrix for random forest with 15 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	2743	1065	3808
<b>Expert (Actual)</b>	1686	2122	3808
<b>Total</b>	4429	3187	7616

The logistic regression model given the best performance for classifying beginner-written programs by classifying 2874 programs correctly out of 3808 programs. The naïve Bayes model performed abysmally by classifying only 39 beginner-written programs correctly among 3808 programs. Naïve Bayes model is sometimes biased towards a specific class [45]. In this case, naïve Bayes appears biased towards class expert.

Table 4.6: Confusion matrix for K nearest neighbor with 15 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	2509	1299	3808
<b>Expert (Actual)</b>	1407	2401	3808
<b>Total</b>	3916	3700	7616

Table 4.7: Confusion matrix for logistic regression with 15 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	2874	934	3808
<b>Expert (Actual)</b>	1731	2077	3808
<b>Total</b>	4605	3011	7616

Table 4.8: Confusion matrix for bagging classifier with 15 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	2694	1114	3808
<b>Expert (Actual)</b>	1433	2375	3808
<b>Total</b>	4127	3489	7616

Naïve Bayes performed very well to classify expert-written programs. This model correctly classified 3752 expert-written programs among 3808 programs. Logistic regression model classified the least number of expert-written programs (2077) correctly. However, the accuracy of the logistic regression model was better than naïve Bayes as the logistic regression performed better to classify the beginner-written programs.



### 4.3.2 Experiment 2

The number of features was reduced to seven from fifteen in experiment 2. Seven features were selected using infogain. The performance of the models evaluated using the 5 fold cross-validation method is reported in Table 4.9.

Table 4.9: Performance evaluation of six models with 7 features.

Model	Accuracy (%)	Precision(%)	Recall(%)	F-Measure(%)
Decision Tree	61%	62%	58%	60%
Naïve Bayes	58%	55%	87%	67%
Random Forest	63%	66%	53%	59%
K Nearest Neighbor	62%	62%	61%	62%
Logistic Regression	65%	69%	52%	60%
Bagging Classifier	63%	64%	60%	62%

The logistic regression model performed better than other models with accurate classification of 65% of the programs. The Bagging classifier and random forest also performed well with an accuracy of 63%. The Naïve Bayes model had the lowest accuracy of 58%.

The F-measure score of naïve Bayes was the topmost (67%). The F-measure of the random forest was the lowest with 59%.

Confusion matrices of the six models for experiment 2 are listed in Tables 4.10 to 4.15.

Table 4.10: Confusion matrix for decision tree with 7 features.

Expertise	Beginner (Predicted)	Expert (Predicted)	Total
Beginner (Actual)	2446	1362	3808
Expert (Actual)	1610	2198	3808
Total	4056	3560	7616

Table 4.11: Confusion matrix for naïve Bayes with 7 features.

Expertise	Beginner (Predicted)	Expert (Predicted)	Total
Beginner (Actual)	1064	2744	3808
Expert (Actual)	492	3316	3808
Total	1556	6060	7616

Table 4.12: Confusion matrix for random forest with 7 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	2748	1060	3808
<b>Expert (Actual)</b>	1778	2030	3808
<b>Total</b>	4526	3090	7616

Table 4.13: Confusion matrix for K nearest neighbor with 7 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	2394	1414	3808
<b>Expert (Actual)</b>	1475	2333	3808
<b>Total</b>	3869	3747	7616

Table 4.14: Confusion matrix for logistic regression with 7 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	2931	877	3808
<b>Expert (Actual)</b>	1819	1989	3808
<b>Total</b>	4750	2866	7616

Table 4.15: Confusion matrix for bagging classifier with 7 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	2540	1268	3808
<b>Expert (Actual)</b>	1525	2283	3808
<b>Total</b>	4065	3551	7616

The logistic regression model performed better than other models to correctly classify beginner-written programs. It classified 2931 programs correctly as beginners out of 3808 programs. The naïve Bayes model was inferior to other models in classifying beginner-written programs. It classified only 1064 programs correctly as beginner-written among 3808 programs.

The naïve Bayes model was the superior model to classify expert-written programs. It correctly classified 3316 programs as experts out of 3808 programs whereas logistic regression performed poorly by correctly classifying 1989 programs as experts among 3808

programs.

### 4.3.3 Experiment 3

In experiment 3, the select4best method was applied to reduce the number of features to four from fifteen. The performance of the models using the 5 fold cross-validation method are shown in Table 4.16.

Table 4.16: Performance evaluation of six models with 4 features.

Model	Accuracy (%)	Precision(%)	Recall(%)	F-Measure(%)
Decision Tree	60%	61%	56%	58%
Naïve Bayes	63%	63%	64%	63%
Random Forest	63%	66%	54%	59%
K Nearest Neighbor	60%	60%	59%	59%
Logistic Regression	63%	67%	53%	59%
Bagging Classifier	61%	61%	58%	60%

Naïve Bayes, random forest and logistic regression performed well to categorize programs based on the author's programming experience with an accuracy of 63%. The accuracy (60%) of decision tree and k nearest neighbor was lowest among all models.

Naïve Bayes was also the leading model based on F-measure (63%). The performance of the decision tree was inferior to other models based on the F-measure (58%) score.

Confusion matrices of the six models for experiment 3 are listed in Tables 4.17 to 4.22.

Table 4.17: Confusion matrix for decision tree with 4 features.

Expertise	Beginner (Predicted)	Expert (Predicted)	Total
Beginner (Actual)	2438	1370	3808
Expert (Actual)	1678	2130	3808
Total	4116	3500	7616

The logistic regression model correctly classified the most beginner-written programs. Logistic regression classified 2803 programs correctly as beginner of a total 3808 beginner-written programs. K nearest neighbor performed poorly with respect to the other models, classifying 2319 programs as beginner among 3808 programs.

Table 4.18: Confusion matrix for naïve Bayes with 4 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	2354	1454	3808
<b>Expert (Actual)</b>	1360	2448	3808
<b>Total</b>	3714	3902	7616

Table 4.19: Confusion matrix for random forest with 4 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	2742	1066	3808
<b>Expert (Actual)</b>	1761	2047	3808
<b>Total</b>	4503	3113	7616

Table 4.20: Confusion matrix for K nearest neighbor with 4 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	2319	1489	3808
<b>Expert (Actual)</b>	1572	2236	3808
<b>Total</b>	3891	3725	7616

Table 4.21: Confusion matrix for logistic regression with 4 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	2803	1005	3808
<b>Expert (Actual)</b>	1803	2005	3808
<b>Total</b>	4606	3010	7616

Table 4.22: Confusion matrix for bagging classifier with 4 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	2415	1393	3808
<b>Expert (Actual)</b>	1589	2219	3808
<b>Total</b>	4004	3612	7616

The naïve Bayes model performed better than other models to categorize expert-written programs. This model identified 2448 programs correctly as experts out of 3808 expert-written programs. On the other hand, logistic regression identified only 2005 programs as

expertly written (of 3808 programs).

#### 4.3.4 Experiment 4

The dataset of 7616 programs was analyzed and classified using 15 features in experiment 4. The models were evaluated using the holdout method. The dataset was partitioned into two sets: a training set and a test set. The learning models were trained on a training set (80% of data) and then evaluated using a test set (20% of data). Table 4.23 demonstrates the performance of six machine learning models in experiment 4.

Table 4.23: Performance evaluation of six models with 15 features.

<b>Model</b>	<b>Accuracy (%)</b>	<b>Precision(%)</b>	<b>Recall(%)</b>	<b>F-Measure(%)</b>
Decision Tree	64%	66%	62%	64%
Naïve Bayes	63%	67%	57%	62%
Random Forest	65%	68%	60%	64%
K Nearest Neighbor	65%	67%	64%	66%
Logistic Regression	65%	69%	58%	63%
Bagging Classifier	69%	73%	65%	68%

The accuracy of the bagging classifier model was 69%. That means the bagging classifier model correctly classified 69% programs to their class label. The F-measure value of this model was 68%, which is superior to the other models.

The accuracy of naïve Bayes was lowest among all models. It classified 63% of programs correctly. In addition, the F-measure score (62%) of the naïve Bayes model was minimum compared to other models.

Confusion matrices of the models developed using decision tree, naïve Bayes, random forest, k nearest neighbor, logistic regression and bagging classifier are shown in the Tables 4.24 to 4.29.

The bagging classifier model performed better than other models to classify beginner-written programs correctly. It correctly classified 545 programs among 738 beginner-written programs. The decision tree model's performance was lowest as it correctly classified only 489 programs out of 738 beginner-written programs.

Table 4.24: Confusion matrix for decision tree with 15 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	489	249	738
<b>Expert (Actual)</b>	302	484	786
<b>Total</b>	791	733	1524

Table 4.25: Confusion matrix for naïve Bayes with 15 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	516	222	738
<b>Expert (Actual)</b>	336	450	786
<b>Total</b>	852	672	1524

Table 4.26: Confusion matrix for random forest with 15 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	514	224	738
<b>Expert (Actual)</b>	312	474	786
<b>Total</b>	826	698	1524

Table 4.27: Confusion matrix for K nearest neighbor with 15 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	493	245	738
<b>Expert (Actual)</b>	281	505	786
<b>Total</b>	774	750	1524

Table 4.28: Confusion matrix for logistic regression with 15 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	536	202	738
<b>Expert (Actual)</b>	332	454	786
<b>Total</b>	868	656	1524

The bagging classifier model performed better than other models in classifying expert-written programs. It correctly classified 509 programs as an expert out of 786 expert-written programs. The performance of naïve Bayes was inferior to other models for classifying

Table 4.29: Confusion matrix for bagging classifier with 15 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	545	193	738
<b>Expert (Actual)</b>	277	509	786
<b>Total</b>	822	702	1524

expert-written programs. The naïve Bayes model correctly classified 450 expert-written programs of a total of 786 expert-written programs.

#### 4.3.5 Experiment 5

In experiment 5, the number of features was reduced to 7 from 15 using infogain. A new dataset was created using the feature values of SLOC%, BLOC%, CLOC%, FLOC AVERAGE, LOC, SLOC-P, SLOC. Table 4.30 reports the performance of six machine learning models using the holdout method.

Table 4.30: Performance evaluation of six models with 7 features.

<b>Model</b>	<b>Accuracy (%)</b>	<b>Precision(%)</b>	<b>Recall(%)</b>	<b>F-Measure(%)</b>
Decision Tree	61%	63%	59%	61%
Naïve Bayes	67%	69%	64%	66%
Random Forest	66%	71%	56%	63%
K Nearest Neighbor	63%	64%	63%	63%
Logistic Regression	67%	70%	61%	65%
Bagging Classifier	63%	64%	61%	62%

Logistic regression and naïve Bayes performed better than the other models in experiment 5, with an accuracy of 67%. The naïve Bayes model also had the top F-measure score (66%).

The accuracy of the decision tree model was lower than other models. This model had an accuracy of 61%. The F-measure score of the decision tree was 61%, which was the lowest among all the models.

The confusion matrices of six models are shown in Table 4.31 to 4.36.

Table 4.31: Confusion matrix for decision tree with 7 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	470	276	746
<b>Expert (Actual)</b>	316	462	778
<b>Total</b>	786	738	1524

Table 4.32: Confusion matrix for naïve Bayes with 7 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	521	225	746
<b>Expert (Actual)</b>	283	495	778
<b>Total</b>	804	720	1524

Table 4.33: Confusion matrix for random forest with 7 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	570	176	746
<b>Expert (Actual)</b>	340	438	778
<b>Total</b>	910	614	1524

Table 4.34: Confusion matrix for K nearest neighbor with 7 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	469	277	746
<b>Expert (Actual)</b>	289	489	778
<b>Total</b>	758	766	1524

Table 4.35: Confusion matrix for logistic regression with 7 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	548	198	746
<b>Expert (Actual)</b>	307	471	778
<b>Total</b>	855	669	1524

Random forest identified the highest number of beginner-written programs compared to other models. This model identified 570 beginner-written programs correctly out of 746 programs. K nearest neighbor performed poorly in identifying beginner-written programs



Table 4.36: Confusion matrix for bagging classifier with 7 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	483	263	746
<b>Expert (Actual)</b>	307	471	778
<b>Total</b>	790	734	1524

as it correctly classified only 469 programs among 746 programs.

The naïve Bayes model was the top model in classifying expert-written programs. It correctly categorized 495 programs as expert-written out of 778 programs. Random forest performed poorly in this regard, correctly classifying only 438 programs as expert-written out of 778 programs.

#### 4.3.6 Experiment 6

In experiment 6, the dataset was classified using four features. The feature set was reduced to four from fifteen by using the select4best method, and then the machine learning models were trained. The evaluation results are listed in Table 4.37.

Table 4.37: Performance evaluation of six models with 4 features.

<b>Model</b>	<b>Accuracy (%)</b>	<b>Precision(%)</b>	<b>Recall(%)</b>	<b>F-Measure(%)</b>
Decision Tree	55%	55%	48%	51%
Naïve Bayes	50%	49%	84%	62%
Random Forest	58%	55%	66%	60%
K Nearest Neighbor	54%	53%	53%	53%
Logistic Regression	56%	55%	59%	57%
Bagging Classifier	56%	56%	53%	54%

None of the models had accuracy over 60%. Random forest performed better than other models with an accuracy of 58%. Naïve Bayes model lead in the F-measure score (62%).

The naïve Bayes model had the lowest accuracy of 50% among all the models. The decision tree model performed particularly poorly given the F-measure score (51%).

Confusion matrices of all the models are listed in Table 4.38 to 4.43.

Table 4.38: Confusion matrix for decision tree with 4 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	489	293	782
<b>Expert (Actual)</b>	386	356	742
<b>Total</b>	875	649	1524

Table 4.39: Confusion matrix for naïve Bayes with 4 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	143	639	782
<b>Expert (Actual)</b>	122	620	742
<b>Total</b>	265	1259	1524

Table 4.40: Confusion matrix for random forest with 4 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	386	396	782
<b>Expert (Actual)</b>	251	491	742
<b>Total</b>	637	887	1524

Table 4.41: Confusion matrix for K nearest neighbor with 4 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	440	342	782
<b>Expert (Actual)</b>	352	390	742
<b>Total</b>	792	732	1524

Table 4.42: Confusion matrix for logistic regression with 4 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	421	361	782
<b>Expert (Actual)</b>	307	435	742
<b>Total</b>	728	796	1524

The decision tree performed adequately to classify beginner-written programs by correctly classifying 489 programs as beginners out of 782 programs. On the contrary, naïve Bayes performed appallingly by correctly classifying only 143 programs as beginner-written

Table 4.43: Confusion matrix for bagging classifier with 4 features.

<b>Expertise</b>	<b>Beginner (Predicted)</b>	<b>Expert (Predicted)</b>	<b>Total</b>
<b>Beginner (Actual)</b>	468	314	782
<b>Expert (Actual)</b>	350	392	742
<b>Total</b>	818	706	1524

among 782 programs.

Naïve Bayes performed reasonably well to classify expert-written programs. It correctly classified 620 expert-written programs out of 742 programs. The decision tree model performed appallingly by correctly classifying only 356 expert-written programs among 742 programs.

## 4.4 Discussion

The goal of this work is to classify computer programs into two classes: expert and beginner. We evaluated the performance of the models in experiments 1, 2 and 3 using the cross-validation method and we used the holdout method to evaluate the models built in experiments 4, 5 and 6. In the holdout method, the dataset is split into two partitions where the training set contains 80% data, and the test set is consist of 20% data. The training partition may contain more data of one class label than another, which can make the model bias towards the class. This drawback can be overcome by using the cross-validation method [11]. Therefore, we used both evaluation methods to determine the difference between them.

### 4.4.1 Cross-Validation Method

In the cross-validation method, the entire dataset was used in  $n$  iterations to train and test the machine learning models. The first three experiments were evaluated using the five-fold cross-validation method. Table 4.44 shows the best and worst model of experiments 1, 2 and 3 based on accuracy.

The bagging classifier model accurately identified more programs compared to other

Table 4.44: Models with highest and lowest accuracy rate in experiment 1 to 3.

<b>Experiment</b>	<b>Model with highest accuracy</b>	<b>Model with lowest accuracy</b>
<b>Experiment 1</b>	Bagging Classifier	Naïve Bayes
<b>Experiment 2</b>	Logistic Regression	Naïve Bayes
<b>Experiment 3</b>	Naïve Bayes, Logistic Regression, Random Forest	Decision Tree, K Nearest Neighbor

models in experiment 1. In all three experiments, the logistic regression model performed well. This model correctly classified more programs in experiment 2 compared to other models based on accuracy. Random forest and naïve Bayes performed well, along with logistic regression in experiment 3.

Although naïve Bayes produced a good result in experiment 3, this model performed poorly in experiment 1 and experiment 2. In other words, naïve Bayes only performed well with four features selected using select4best method. Naïve Bayes did not perform well with 15 features (experiment 1) and 7 features (experiment 2). Therefore, feature reduction increased the accuracy for the naïve Bayes model. Because with more features, likelihood might be distributed and may not follow the Gaussian or other distribution.

Table 4.45 shows the best and worst model of experiments 1, 2 and 3 based on correctly identifying beginner-written programs.

Table 4.45: Best and worst models in classifying beginner-written programs for experiments 1 to 3.

<b>Experiment</b>	<b>Best model</b>	<b>Worst model</b>
<b>Experiment 1</b>	Logistic Regression	Naïve Bayes
<b>Experiment 2</b>	Logistic Regression	Naïve Bayes
<b>Experiment 3</b>	Logistic Regression	K Nearest Neighbor

The logistic regression model correctly classified most beginner-written programs compared to other models in all three experiments. In experiments 1 and 2, the naïve Bayes model performed poorly when attempting to classify beginner-written programs correctly. In experiment 3, k nearest neighbor performed poorly when attempting to classify beginner-

written programs correctly.

Table 4.46 shows the best and worst model of experiments 1, 2 and 3 based on identifying expert-written programs.

Table 4.46: Best and worst models in classifying expert-written programs for experiments 1 to 3.

<b>Experiment</b>	<b>Best model</b>	<b>Worst model</b>
<b>Experiment 1</b>	Naïve Bayes	Logistic Regression
<b>Experiment 2</b>	Naïve Bayes	Logistic Regression
<b>Experiment 3</b>	Naïve Bayes	Logistic Regression

Naïve Bayes model correctly classified most expert-written programs compared to other models in all three experiments because the naïve Bayes model sometimes develops a bias towards a class [45]. The logistic regression model correctly classified the least number of expert-written programs compared to other models (in experiments 1 to 3). However, the difference between the logistic regression and the rest of the models in terms of correctly classified expert-written programs was not significant.

#### 4.4.2 Holdout Method

Experiments 4, 5 and 6 were evaluated using the holdout method. In this method, the dataset was partitioned into two sets: training set (80% of data) and test set (20% of data). Machine learning models were built using a training set and evaluated using a test set. Experiments 4, 5 and 6 were conducted using 15, 7 and 4 features respectively.

Table 4.47 shows the best and worst model of experiments 4, 5 and 6 based on accuracy.

Table 4.47: Models with highest and lowest accuracy rate in experiments 4 to 6.

<b>Experiment</b>	<b>Model with highest accuracy</b>	<b>Model with lowest accuracy</b>
<b>Experiment 4</b>	Bagging Classifier	Naïve Bayes
<b>Experiment 5</b>	Naïve Bayes, Logistic Regression	Decision Tree
<b>Experiment 6</b>	Random Forest	Naïve Bayes

The bagging classifier model performed well based on accuracy in experiment 4 using 15 features as bagging performs better using all available features [34]. Naïve Bayes and logistic regression were the leading model in experiment 5. Logistic regression performs reliably with noisy data [47]. Random forest had the highest accuracy in experiment 6 using 4 features. The reduction of less important features from 15 to 4 possibly significantly impacted the classification performance of random forest.

Table 4.48 shows the best and worst model of experiments 4, 5 and 6 based on identifying beginner-written programs.

Table 4.48: Best and worst models in classifying beginner-written programs for experiments 4 to 6.

<b>Experiment</b>	<b>Best model</b>	<b>Worst model</b>
<b>Experiment 4</b>	Bagging Classifier	Decision Tree
<b>Experiment 5</b>	Random Forest	K Nearest Neighbor
<b>Experiment 6</b>	Decision Tree	Naïve Bayes

Bagging classifiers performed reasonably well to identify beginner-written programs in experiment 4. Random forest was the leading model to correctly identify beginner-written programs in experiment 5 using 7 features. The decision tree performed well using 4 features in experiment 6 to classify beginner-written programs correctly. However, decision tree performed poorly using 15 features in experiment 4 to classify beginner-written programs correctly.

Table 4.49 shows the best and worst model of experiments 4, 5 and 6 based on identifying expert-written programs.

Table 4.49: Best and worst models in classifying expert-written programs for experiments 4 to 6.

<b>Experiment</b>	<b>Best model</b>	<b>Worst model</b>
<b>Experiment 4</b>	Bagging Classifier	Naïve Bayes
<b>Experiment 5</b>	Naïve Bayes	Random Forest
<b>Experiment 6</b>	Naïve Bayes	Decision Tree

The bagging classifier performed well to classify expert-written programs in experiment

4. The naïve Bayes model was the leading model to identify expert-written programs in experiments 5 and 6. However, naïve Bayes did not perform well using 15 features in experiment 4 to classify expert-written programs correctly.

#### 4.4.3 Summary of Results

By considering all six experiments, the most interesting results are as follows:

- The logistic regression model performed better than other models based on accuracy as it had the best accuracy in three experiments. Logistic regression yielded better results when models were evaluated using five-fold cross-validation, as the training set consisted of the entire dataset instead of the holdout method, where the training set contained 80% of data. Therefore, logistic regression performs better when the training size is bigger [38]. The confusion matrices of logistic regression for experiments 1 to 6 are given in Tables 4.7, 4.14, 4.21, 4.28, 4.35 and 4.42. Logistic regression showed a stable performance to classify both expert-written and beginner-written programs correctly.

In most cases, when a model did very well to classify one class of programs correctly, it did not perform well to classify other classes of programs. For example, from Table 4.4, we can see that, in experiment 1, the naïve Bayes model performed better to correctly classify expert-written programs by correctly classifying 3752 expert-written programs among 3808 programs. However, naïve Bayes performed poorly to identify beginner-written programs as it identified only 39 beginner-written programs among 3808 programs. In experiment 1, from Table 4.7 we can see that logistic regression correctly classified 2077 expert-written programs among 3808 programs and correctly identified 2874 beginner-written programs among 3808 programs. In all of the experiments logistic regression showed a reliable performance to classify programs of both classes correctly.

- The bagging classifier model performed better using 15 features. Experiments 1 and 4

were carried out using 15 features. It had better accuracy (67%) than other models in experiment 1. Also, the bagging classifier model was the leading model in experiment 4, with an accuracy of 69%. It was the superior model to identify beginner-written and expert-written programs in experiment 4 correctly as bagging performs better using all available features [34].

- The naïve Bayes model was the leading model to classify expert-written programs correctly. This model was superior to other models to correctly classify expert-written program in all experiments except experiment 4. The naïve Bayes model sometimes developed a bias towards a class [45]. We can see from Table 4.39 that naïve Bayes was biased towards class expert, as it also misclassified 639 out of 782 beginner-written programs as expert-written in experiment 6.
- The logistic regression model performed better compared to other models to identify beginner-written programs correctly. It was the top model in experiments 1,2, and 3 to correctly classify beginner-written programs.
- The random forest model performed well in experiments 3 and 6, where a reduced number of features (4 features) were used. In both experiments, the random forest model was the leading model based on accuracy. The reduction of less important features from 15 to 4 possibly significantly impacted the classification performance of random forest.

## 4.5 Threats to Validity

There are some potential threats to validity with some of this work in analyzing computer programs [7].

- The dataset of our research was not large as it consisted of 7616 C++ programs. In most of the cases, the machine learning model might be trained more effectively if the dataset was bigger.



- Only C++ programs are considered for this research. We were unable to demonstrate different types of results as other kinds of programming languages were not considered. This could be a threat to the validity of the results of the machine learning models.
- Another threat to validity is the coding style of the participants of Codeforces.com. Programs are only submitted for the programming contests. So, there might be differences between the coding style of contest programs and regular programs.
- Our models were trained using a limited number of features. We might have left out some important features of C++.
- Our dataset is balanced over the regions that we categorized. We collected data from Asia and Europe. Computer programs were collected from 19 countries in Asia and 26 countries in Europe, but the dataset is not balanced over all of the countries. This could be another potential threat to validity.
- For each programmer, 50 programs were stored in the data repository. We took eight programs randomly among those 50 in our dataset. There might be some programs from expert programmers that were written during their beginner days. Still, those codes were considered as expert-written, which is a potential threat.
- There is a possibility that someone may not write code and borrowed it from some other programmer to submit in the Codeforces programming contest. This act is a violation of ethics. Still, if it happens, there is a possibility that an expert-written program might be labeled as beginner-written in the source, which is a threat to validity.

# Chapter 5

## Analysis of Features

In this chapter we discuss the features and their importance in determining the expert-written or beginner-written programs. We analyze the feature reduction methods and how the reduced number of features impact the models. Moreover, we describe a statistical approach to determine the most relevant features to categorize programs based on the author's experience.

### 5.1 Feature Reduction

As described in Chapter 3, we selected fifteen features based on [44]. Experiments 1 and 4 were carried out using 15 features to categorize computer programs based on the author's programming experience. Then the feature set was reduced to seven features by selecting seven features using the infogain. This feature selection method provides the rank of features based on the class. We used the top seven features in experiments 2 and 5. These features were:

- (i) percentage of source lines of code (SLOC%),
- (ii) percentage of blank lines of code (BLOC%),
- (iii) percentage of comment lines of code (CLOC%),
- (iv) average lines of code per functions (FLOC AVERAGE),
- (v) lines of code (LOC),

- (vi) physical executable lines of code (SLOC-P), and
- (vii) source lines of code (SLOC).

Following this, we used a univariate selection method, `selectkbest` (using  $k=4$ ), to select the four best features among all fifteen features. `Select4best` returned the following features:

- (i) lines of code (LOC),
- (ii) physical executable lines of code (SLOC-P),
- (iii) logical executable lines of code (SLOC-L), and
- (iv) source lines of code (SLOC).

Experiments 3 and 6 were conducted using the above four features. The machine learning models were evaluated using a five-fold cross-validation method in experiments 1 to 3. The holdout method was used to evaluate the models in experiments 4 to 6.

We compared the six models' performance to determine the significance of reduced feature sets. Table 5.1 shows the comparative performance of six models with fifteen features, seven features and four features using a five-fold cross-validation method.

Table 5.1: Performance evaluation of six models based on five-fold cross-validation technique.

<b>Model</b>	<b>Accuracy based on 15 features</b>	<b>Accuracy based on 7 features</b>	<b>Accuracy based on 4 features</b>
Decision Tree	64%	61%	60%
Naïve Bayes	50%	58%	<b>63%</b>
Random Forest	64%	63%	<b>63%</b>
K Nearest Neighbor	64%	62%	60%
Logistic Regression	65%	<b>65%</b>	<b>63%</b>
Bagging Classifier	<b>67%</b>	63%	61%

As described in Chapter 4, bagging classifiers yielded a better result (accuracy=67%) than other models when using fifteen features. Logistic regression performed well compared to other models using only the seven features returned by the `infogain`. It correctly

predicted 65% of the programs. Along with logistic regression, naïve Bayes and random forest performed well using the reduced set of four features returned by select4best. These three models had an accuracy of 63% using four features.

The performance of all the machine learning models decreased or remained the same as the number of features reduced except with the naïve Bayes model. For example, the accuracy values of the decision tree model were 64%, 61% and 60%, respectively for fifteen, seven and four features. The performance of naïve Bayes model improved as the number of features was reduced. The accuracy values of the naïve Bayes model were 50%, 58% and 63% respectively by using fifteen features, seven features and four features.

The bagging classifier performed better than other models using 15 features in experiment 1. Logistic regression was the leading model in experiments 2 and 3 (using seven features and four features). Tables 4.7, 4.14 and 4.21 show the confusion matrices of the logistic regression model for three feature sets (respectively for fifteen, seven and four features). Logistic regression correctly predicted the most (2931 out of 3808) programs as beginner-written using seven features. It was the best predictor for beginner-written programs among all six models and using all three feature sets. Logistic regression worked well using fifteen features to correctly classify most number (2874 among 3808) of programs as beginner-written. However, the naïve Bayes model predicted the most (3752 among 3808) expert-written programs among all six models using fifteen features.

Among all machine learning models, the bagging classifier showed the highest accuracy using cross-validation method, which was 67% in experiment 1. However, the accuracy of 67% may not be good enough for industrial use. Nevertheless, there is a scope for improvement. Using a larger set of data and features may increase the accuracy rate.

## 5.2 Statistical Approach

In this research, we categorized programs into two classes: expert and beginner. We built six machine learning models to categorize programs based on the author's program-

ming experience. These models were built using up to fifteen features. Our goal was to determine which features are most significant to classify programs in these two classes. These features might refer to the different programming styles of experts and beginners. We wanted to determine whether the difference was realistic or it occurred randomly. We performed the statistical test T-test to determine whether the differences are significant or not.

The T-test compares two groups based on their means. This method is known as a bivariate statistical test [9]. The T-test computes the difference between the two groups by calculating their means. In addition, it represents the significance of the differences.

There are two hypotheses in a T-test: the null hypothesis and the alternative hypothesis. The null hypothesis is denoted by  $H_0$ , and  $H_a$  represents the alternative hypothesis. The null hypothesis,  $H_0$ , is tested in contrast to the alternative hypothesis,  $H_a$ . The two hypotheses are as follows:

- the null hypothesis ( $H_0$ ) defines that, there is no difference between two classes ( $\mu_1 = \mu_2$ ) as it might occur by chance, and
- the alternative hypothesis ( $H_a$ ) states that there are differences between two classes ( $\mu_1 \neq \mu_2$ ).

The outcome of the T-test may accept or reject the null hypothesis. The alternative hypothesis is accepted when the null hypothesis is rejected. The  $p$  value is computed in a T-test, which shows whether the difference occurs by chance or is real [9]. A threshold value, 0.05, is used to illustrate the statistical significance between two classes. The relationship between the  $p$  value and the hypotheses is as follows:

- the null hypothesis ( $H_0$ ) is rejected when  $p < 0.05$ ; it means that the statistical difference between the two classes is real, and
- the null hypothesis ( $H_0$ ) gets accepted when  $p > 0.05$ ; which means that the statistical difference occurs by chance.

Table 5.2: T-test ( $\rho$ ) values of features.

Features	$\rho$ value
Lines of code (LOC)	$2.3089 \times 10^{-57}$
Source lines of code (SLOC)	$1.8661 \times 10^{-61}$
Percentage of source lines of code (SLOC%)	0.9932
Blank lines of code (BLOC)	$5.6244 \times 10^{-39}$
Percentage of blank lines of code (BLOC%)	0.0213
Comment lines of code (CLOC)	0.6110
Percentage of comment lines of code (CLOC%)	0.0077
Mixed lines of code with both source and comments (C_SLOC)	0.0002
Percentage of mixed lines (C_SLOC%)	0.1192
Total words in all comments (CWORD)	0.0625
Physical executable lines of code (SLOC_P)	$1.8661 \times 10^{-61}$
Logical executable lines of code (SLOC_L)	$2.0891 \times 10^{-60}$
The number of functions defined (Functions)	$1.4839 \times 10^{-26}$
Lines of code in functions (FLOC)	$1.2852 \times 10^{-38}$
Average lines of code per functions (FLOC average)	0.0447

We conducted a T-test using Microsoft Excel© to determine the significance of our features for expert-written and beginner-written programs. We carried out a two-tailed T-test to determine the differences between the two classes and identify whether the difference is statistically significant. The features were treated as dependent variables, and the two classes, experts and beginners, are considered as independent variables in the T-test. Table 5.2 shows the  $\rho$  values returned by the T-test.

From Table 5.2, we can see that eleven features have  $\rho$  values less than 0.05. These features are:

- (i) lines of code (LOC),

- (ii) source lines of code (SLOC),
- (iii) blank lines of code (BLOC),
- (iv) percentage of blank lines of code (BLOC%),
- (v) percentage of comment lines of code (CLOC%),
- (vi) mixed lines of code with both source and comments (C\_SLOC),
- (vii) physical executable lines of code (SLOC\_P),
- (viii) logical executable lines of code (SLOC\_L),
- (ix) the number of functions defined (Functions),
- (x) lines of code in functions (FLOC), and
- (xi) average lines of code per functions (FLOC average).

The above features are said to be statistically significant as their  $p$  values are less than 0.05. The null hypothesis is rejected for these eleven features. For the remaining features, which have  $p$  values greater than zero, we cannot reject the null hypothesis. These features (SLOC%, CLOC, C\_SLOC%, CWORD) are therefore not considered as statistically significant to classify expert-written and beginner-written programs.

A comparative analysis was conducted on eleven statistically significant features. Table 5.3 demonstrates the usage of statistically significant features in both expert-written and beginner-written programs.

Beginner programmers used more blank lines in the program compared to expert programmers. 66% of the total blank lines were present in the beginner-written programs, whereas 34% of the total blank lines were present in the expert-written programs. Therefore, blank lines of code (BLOC) is a significant feature that can differentiate between beginner-written and expert-written programs. Another feature, average lines of code per

Table 5.3: Comparison of feature usage in expert-written and beginner-written programs.

<b>Features</b>	<b>Expert-Written Programs</b>	<b>Beginner-Written Programs</b>
Lines of code (LOC)	55%	45%
Source lines of code (SLOC)	59%	41%
Blank lines of code (BLOC)	34%	66%
Percentage of blank lines of code (BLOC%)	52%	48%
Percentage of comment lines of code (CLOC%)	57%	43%
Mixed lines of code with both source and comments (C_SLOC)	57%	43%
Physical executable lines of code (SLOC_p)	59%	41%
Logical executable lines of code (SLOC_L)	60%	40%
The number of functions defined (Functions)	61%	39%
Lines of code in functions (FLOC)	57%	43%
Average lines of code per functions (FLOC average)	49%	51%

function (FLOC average), had a higher values of usage in beginner-written programs at 51%.

Apart from BLOC and FLOC average, the other nine statistically significant features were more used in the expert-written programs than in beginner-written programs. Expert programmers used more functions than beginner programmers. Expert programmers defined 61% of total functions, while beginner programmers defined 39% of total functions. The programs written by expert programmers could be described as more structured as they tended to utilize more user-defined functions than beginner programmers. Moreover, the number of executable statements (SLOC\_L) was higher in expert-written programs. 60% of SLOC\_L were present in the expert-written program. Also, the usage of LOC, SLOC, BLOC%, CLOC%, C\_SLOC, SLOC\_P, and FLOC were higher in expert-written programs than beginner-written programs.



### 5.3 Visual Analysis of Programs

In this section, we visually explored the programs to discover the differences between expert-written and beginner-written programs. We analyzed the eleven statistically significant features by investigating four randomly-selected programs. Among these programs, two were the expert-written programs, and two were the beginner-written programs. The comparison took place among expert and beginner written programs having an almost similar length of codes.

For the first visual analysis, we compared a beginner-written program that was 450 lines long with an expert-written program which was 447 lines long. As our second comparison, we studied another pair of programs written by a beginner and an expert that were both 100 lines long.

Table 5.4 shows the first comparison between beginner and expert-written program.

Table 5.4: First comparison between expert-written and beginner-written programs.

<b>Features</b>	<b>Expert-Written Program</b>	<b>Beginner-Written Program</b>
Lines of code (LOC)	447	450
Source lines of code (SLOC)	416	393
Blank lines of code (BLOC)	5	55
Percentage of Blank lines of code (BLOC%)	1.12%	12.22%
Percentage of comment lines of code (CLOC%)	5.82%	0.44%
Mixed lines of code with both source and comments (C.SLOC)	0	0
Physical executable lines of code (SLOC.P)	416	393
Logical executable lines of code (SLOC.L)	296	286
The number of functions defined (Functions)	21	2
Lines of code in functions (FLOC)	269	7
Average lines of code per functions (FLOC average)	12.81	3.5

Table 5.4 shows that the values of SLOC, SLOC.P, SLOC.L features were greater in the expert-written programs than beginner-written programs. But the use of the BLOC and BLOC% in the expert-written program were significantly less compared to the beginner-written program. Conversely, the feature values of CLOC%, functions, FLOC and FLOC average were higher in the expert-written program than in the beginner-written program.

Then we visually compared two programs written by expert and beginner individually having the same length of code. The second comparison is represented in Table 5.5.

Table 5.5: Second comparison between expert-written and beginner-written programs.

<b>Features</b>	<b>Expert-Written Program</b>	<b>Beginner-Written Program</b>
Lines of code (LOC)	100	100
Source lines of code (SLOC)	75	90
Blank lines of code (BLOC)	3	10
Percentage of blank lines of code (BLOC%)	3%	10%
Percentage of comment lines of code (CLOC%)	22%	0%
Mixed lines of code with both source and comments (C.SLOC)	0	0
Physical executable lines of code (SLOC.P)	75	90
Logical executable lines of code (SLOC.L)	46	72
The number of functions defined (Functions)	12	1
Lines of code in functions (FLOC)	34	88
Average lines of code per functions (FLOC average)	2.83	88

From Table 5.5, we can see that the value of the feature SLOC, BLOC, SLOC.P and SLOC.L was higher in the beginner-written program. However, in the expert-written program, the feature value of comment lines and functions was higher.

From both comparisons, it can be speculated that the expert-written programmer used more functions and comments, and less blank lines than the beginner-written programmer.

We can see that expert-written programmers were more structured as they used more functions than beginner-written programmers. Also, expert-written programmers used more comments in the program, which enhances the readability of the program.

## 5.4 Relationship Between Features

In this section, we investigate the relationship between pairs of features in order to find out the strength of the relationship between features. We are interested in finding out the linear relationship between features for both beginner-written programs and expert-written programs. The  $\gamma$  value was calculated to discover whether there was a relationship between a pair of features or not. There can be three kinds of relationships between a pair of features based on the  $\gamma$  value [56]:

- (i) A pair of features is positively correlated when  $\gamma > 0$ . A pair of features is strongly related when the  $\gamma$  value is higher.
- (ii) A pair of features is negatively correlated when  $\gamma < 0$ . It indicates that when a feature from a pair occurs, the other feature may not occur.
- (iii) There is no correlation between features when  $\gamma = 0$ . This means that the features are not dependent on each other.

### 5.4.1 Beginner-written Programs

Figure 5.1 shows the correlation matrix of beginner-written programs based on original feature frequencies. There are 63 positively correlated pairs of features in the beginner-written programs (correlation,  $\gamma > 0$ ). Among them, 13 pairs are strongly related as their correlation ( $\gamma$ ) values are greater than 0.5. The strongly connected features of beginner-written programs are shown in Table 5.6.

There are five pairs of features of beginner-written programs which are not dependent on each other ( $\gamma = 0$ ). These pairs are:

Table 5.6: Strongly connected features of beginner-written programs.

Pair No.	Feature1	Feature2
1	Lines of code (LOC)	Blank lines of code (BLOC)
2	Source lines of code (SLOC)	Physical executable lines of code (SLOC_P)
3	Source lines of code (SLOC)	Logical executable lines of code (SLOC_L)
4	Source lines of code (SLOC)	Lines of code in functions (FLOC)
5	Comment lines of code (CLOC)	Percentage of comment lines of code (CLOC%)
6	Comment lines of code (CLOC)	Total words in all comments (CWORD)
7	Percentage of comment lines of code (CLOC%)	Total words in all comments (CWORD)
8	Mixed lines of code with both source and comments (C_SLOC)	Percentage of mixed lines (C_SLOC%)
9	Physical executable lines of code (SLOC_P)	Logical executable lines of code (SLOC_L)
10	Physical executable lines of code (SLOC_P)	Lines of code in functions (FLOC)
11	Logical executable lines of code (SLOC_L)	The number of functions defined (Function)
12	Logical executable lines of code (SLOC_L)	Lines of code in functions (FLOC)
13	Lines of code in functions (FLOC)	Average lines of code per functions (FLOC average)

- Percentage of source lines of code (SLOC%) and the number of functions defined (Function).
- Blank lines of code (BLOC) with comment lines of code (CLOC), mixed lines of code with both source and comments (C\_SLOC), total words in all comments (CWORD) and the number of functions defined (Function).

In beginner-written programs, source lines of code (SLOC) and physical executable lines of code (SLOC\_P) are highly related as  $\gamma = 1$ . Apart from that, lines of code (LOC)

and blank lines of code (BLOC) are strongly connected in beginner-written programs with a  $\gamma$  value of 0.99. Both logical executable lines of code (SLOC\_L) and lines of code in functions (FLOC) appeared in four pairs of related features for beginner-written programs.

### 5.4.2 Expert-written Programs

The  $\gamma$  values of correlation matrix for expert-written programs are represented in Figure 5.2.

There are 81 pairs of features in expert-written programs that are positively correlated ( $\gamma > 0$ ). Twenty four pairs among them are strongly related ( $\gamma > 0.5$ ) which are listed in Tables 5.7 and 5.8.

Table 5.7: Strongly related features of expert-written programs.

Pair No.	Feature1	Feature2
1	Lines of code (LOC)	Source lines of code (SLOC)
2	Lines of code (LOC)	Blank lines of code (BLOC)
3	Lines of code (LOC)	Physical executable lines of code (SLOC_P)
4	Lines of code (LOC)	Logical executable lines of code (SLOC_L)
5	Lines of code (LOC)	The number of functions defined (Function)
6	Lines of code (LOC)	Lines of code in functions (FLOC)
7	Source lines of code (SLOC)	Physical executable lines of code (SLOC_P)
8	Source lines of code (SLOC)	Logical executable lines of code (SLOC_L)
9	Source lines of code (SLOC)	The number of functions defined (Function)
10	Source lines of code (SLOC)	Lines of code in functions (FLOC)
11	Blank lines of code (BLOC)	Percentage of blank lines of code (BLOC%)
12	Blank lines of code (BLOC)	Physical executable lines of code (SLOC_P)

Table 5.8: Strongly related features of expert-written programs (continued).

<b>Pair No.</b>	<b>Feature1</b>	<b>Feature2</b>
13	Blank lines of code (BLOC)	Logical executable lines of code (SLOC_L)
14	Blank lines of code (BLOC)	Lines of code in functions (FLOC)
15	Comment lines of code (CLOC)	Percentage of comment lines of code (CLOC%)
16	Comment lines of code (CLOC)	Total words in all comments (CWORD)
17	Percentage of comment lines of code (CLOC%)	Total words in all comments (CWORD).
18	Mixed lines of code with both source and comments (C_SLOC)	Percentage of mixed lines (C_SLOC%)
19	Physical executable lines of code (SLOC_P)	Logical executable lines of code (SLOC_L)
20	Physical executable lines of code (SLOC_P)	The number of functions defined (Function)
21	Physical executable lines of code (SLOC_P)	Lines of code in functions (FLOC)
22	Logical executable lines of code (SLOC_L)	The number of functions defined (Function)
23	Logical executable lines of code (SLOC_L)	Lines of code in functions (FLOC)
24	The number of functions defined (Function)	Lines of code in functions (FLOC)

In the expert-written programs, there are three pair of features which are independent of each other. These pairs are:

- Percentage of blank lines of code (BLOC%) and mixed lines of code with both source and comments (C\_SLOC).
- Percentage of comment lines of code (CLOC%) and average lines of code per functions (FLOC average).
- Percentage of mixed lines (C\_SLOC%) and lines of code in functions (FLOC).

In both beginner and expert-written programs, source lines of code (SLOC) and physical

executable lines of code (SLOC\_P) are highly related as  $\gamma = 1$ . In expert-written programs, lines of code (LOC) are very strongly related to source lines of code (SLOC) and physical executable lines of code (SLOC\_P) with  $\gamma = 0.97$ . Moreover, lines of code (LOC), physical executable lines of code (SLOC\_P), logical executable lines of code (SLOC\_L) and lines of code in functions (FLOC) appeared in six pairs of related features for expert-written programs.

## 5.4. RELATIONSHIP BETWEEN FEATURES

	LOC	SLOC	SLOC %	BLOC	BLOC %	CLOC	CLOC %	C&S LOC	C&S LOC %	CWORD	SLOC -P	SLOC -L	Functions	FL OC	FLOC_avg
LOC	1														
SLOC	0.12	1													
SLOC %	0.38	0.03	1												
BLOC	0.99	0.01	0.37	1											
BLOC %	0.43	0.08	0.83	0.45	1										
CLOC	0.05	0.16	0.45	0.00	0.03	1									
CLOC %	0.02	0.07	0.54	0.01	0.03	0.84	1								
C&S LOC	0.03	0.23	0.01	0.00	0.02	0.06	0.05	1							
C&S LOC %	0.01	0.11	0.01	0.01	0.02	0.04	0.04	0.92	1						
CWORD	0.05	0.18	0.40	0.00	0.04	0.93	0.77	0.08	0.05	1					
SLOC -P	0.12	0.01	0.03	0.01	0.08	0.16	0.07	0.23	0.11	0.18	1				
SLOC -L	0.11	0.95	0.02	0.01	0.07	0.14	0.06	0.18	0.06	0.17	0.95	1			
Functions	0.07	0.60	0.00	0.00	0.01	0.07	0.01	0.29	0.17	0.09	0.60	0.56	1		
FLOC	0.10	0.84	0.06	0.01	0.03	0.21	0.15	0.12	0.05	0.24	0.84	0.86	0.38	1	
FLOC_avg	0.04	0.37	0.03	0.01	0.04	0.11	0.11	-0.02	-0.02	0.13	0.37	0.40	-0.16	0.65	1

Figure 5.1: Correlation based on raw frequency of features in beginner-written programs.



## 5.4. RELATIONSHIP BETWEEN FEATURES

	LOC	SLOC	SLOC %	BLOC	BLOC %	CLOC	CLOC %	C&S LOC	C&S LOC %	CW ORD	SLOC -P	SLOC -L	Functions	FL OC	FLOC avg
LOC	1														
SLOC	0.97	1													
SLOC %	0.18	0.02	1												
BLOC	0.67	0.56	0.53	1											
BLOC %	0.02	0.08	0.74	0.61	1										
CLOC	0.42	0.24	0.55	0.21	0.05	1									
CLOC %	0.24	0.07	0.63	0.09	0.05	0.87	1								
C&S LOC	0.14	0.13	0.05	0.10	0.00	0.06	0.06	1							
C&S LOC %	0.03	0.03	0.03	0.02	0.01	0.02	0.06	0.89	1						
CW ORD	0.48	0.32	0.46	0.30	0.01	0.86	0.71	0.21	0.14	1					
SLOC -P	0.97	1	0.02	0.56	0.08	0.24	0.07	0.13	0.03	0.32	1				
SLOC -L	0.93	0.96	0.02	0.56	0.06	0.22	0.04	0.11	0.01	0.32	0.96	1			
Functions	0.67	0.67	0.07	0.48	0.03	0.17	0.07	0.13	0.05	0.21	0.67	0.66	1		
FLOC	0.83	0.85	0.05	0.57	0.01	0.17	0.07	0.08	0.00	0.22	0.85	0.84	0.58	1	
FLOC avg	0.13	0.15	0.04	0.04	0.05	0.01	0.00	0.02	-0.03	0.01	0.15	0.15	-0.34	0.36	1

Figure 5.2: Correlation based on raw frequency of features in expert-Written programs.

# Chapter 6

## Conclusion and Future Work

Argamon demonstrated that sociolinguistic characteristics such as the author's gender, age, and experience have an undeniable effect on natural language use [48], [49]. In [32], Misk-Falkoff stated that a computer program could be analyzed using the techniques of linguistics. A natural language speaker shows individuality through language choice, and in the same way, a computer programmer also represents their individualism through their programming language choice. This research aims to determine the effects of the author's programming experience, a sociolinguistic characteristic, in the development of computer programs.

The focus of our research is to determine how machine learning techniques can be applied to analyze computer programs. We carried out experiments on programs written in C++. C++ programs are text files, and for text categorization, machine learning approaches are widely used [49]. Naz [36] previously used machine learning techniques to classify computer programs based on the author's gender. Rafee [32] improved the learning techniques to categorize the programs based on the author's gender, and also classified programs based on the author's region. In our research, we investigated whether the author is a beginner or an expert by analyzing the program that they have written.

We created a dataset containing 7616 computer programs written in C++. These C++ programs were collected from Codeforces.com. Our dataset was balanced over the author's experience, gender and region: half of the programs were written by beginner programmers, and the remaining half were written by expert programmers. Programs were collected from

two regions: Asia and Europe. Moreover, among 7616 programs, 3808 programs were male-written, and 3808 were female-written. We selected the program's structural attributes such as the total number of lines, the number of blank lines and the number of comments as the features to analyze the programs based on the author's programming experience. Fifteen of these features were selected, which assisted in the determination of the programmer's unique coding style. The programs were converted to numeric representation using the feature values. We used six machine learning techniques including decision tree, naïve Bayes, random forest, k nearest neighbor, logistic regression and bagging classifier to build six models. We performed six experiments using six machine learning models with a different number of features. We used both holdout and a cross-validation method to evaluate the performance of the models.

In experiments 1 and 4, fifteen features were used to classify the programs based on the author's programming experience. Machine learning models were evaluated using five-fold cross-validation technique in experiment 1. 67% programs were correctly classified in experiment 1. In experiment 4, models were evaluated using the holdout method. In the holdout method, the dataset was partitioned into two sets: training set (80% data) and test set (20% data). Models were built by training them using the training set. Then the models were evaluated using the test set. We achieved an accuracy of 69%, which means 69% of the total data were correctly classified.

We used infogain in experiments 2 and 5 to reduce the number of features from fifteen to seven. The top seven features selected using infogain were source lines of code percentage (SLOC%), blank lines of code percentage (BLOC%), comment lines of code percentage (CLOC%), average lines of code per functions (FLOC AVERAGE), lines of code (LOC), physical executable lines of code (SLOC-P), and source lines of code (SLOC). The performance of the models was evaluated using five-fold cross-validation method in experiment 2. We accomplished an accuracy of 65% in experiment 2, meaning 65% of data were correctly classified as beginner-written and expert-written programs. The models in ex-

periment 5 were evaluated using the holdout method. We achieved an accuracy of 67% in this experiment. Later, we performed a T-test which yielded eleven statistically significant features from fifteen features. Six features out of seven used in experiments 2 and 5 were statistically significant. Only the source lines of code percentage (SLOC%) was not statistically significant.

The number of features was reduced to four from fifteen using the select4best method in experiments 3 and 6. The four best features were lines of code (LOC), physical executable lines of code (SLOC-P), logical executable lines of code (SLOC-L) and source lines of code (SLOC). All of these features were statistically significant, according to the T-test. The machine learning models were evaluated using five-fold cross-validation technique in experiment 3. We achieved an accuracy of 63% in experiment 3. That means 63% of programs were correctly classified based on the author's programming experience. The same four features were used in experiment 6 to build the models. Instead of the cross-validation technique, the holdout method was used to evaluate the models in experiment 6. 58% of computer programs were correctly classified as expert-written and beginner-written programs in experiment 6.

We performed statistical analysis on the selected fifteen features of the computer programs. T-test was performed on the features based on class labels: expert and beginner. Based on the  $p$  values of the T-test, eleven features were statistically significant. These features were the lines of code (LOC), the source lines of code (SLOC), the blank lines of code (BLOC), the blank lines of code percentage (BLOC%), the comment lines of code percentage (CLOC%), the mixed lines of code with both source and the comments (C\_SLOC), the physical executable lines of code (SLOC\_p), the logical executable lines of code (SLOC\_L), the number of functions defined (Functions), the lines of code in functions (FLOC), and the average lines of code per functions (FLOC average).

We also carried out a visual analysis of some randomly selected programs. We visually compared the statistically significant features of programs based on the author's program-

ming experience. The highlights of the feature analysis are given below:

- Beginner programmers used more blank lines of codes than expert programmers.
- The average lines of code per function were higher in beginner-written programs than expert-written programs.
- Apart from blank lines of codes and average lines of code per function, the rest of the nine statistically significant features were more used in expert-written programs than beginner-written programs.
- From the visual analysis, it appeared that expert programmers were more structured as they used more functions than beginner programmers.
- Moreover, the code readability of expert programmers was higher as they used more comments in the programs than beginner programmers.

### **6.1 Future Research Directions**

This research has opened the path for many future research directions. Some of the future research recommendations are as follows:

- We categorized computer programs based on the author's programming experience. It would be compelling to investigate the effect of another sociolinguistic characteristic, such as the author's native language on computer programs.
- In this research, we used programs written in C++ in the experiments. However, there are other popular programming languages, such as Python and Java. In the future, we would like to conduct experiments on other programming languages.
- There were 7616 programs in our dataset. The machine learning models were built by training them using the programs of our dataset. A larger dataset would train the models more efficiently. Therefore, we plan to extend the dataset.

- We classified the programs in two classes: beginner and expert. There are more categories of programmers apart from beginners and experts. We wish to categorize the programs in more classes such as beginner, intermediate, expert and master.
- We used six supervised machine learning algorithms, including decision tree, naïve Bayes, random forest, k nearest neighbor, logistic regression, and bagging classifier to categorize computer programs based on the author's programming experience. A neural network is a modern supervised learning technique that may achieve higher accuracy to classify computer programs.
- For categorizing programs we applied supervised machine learning approaches. In the future, we would like to use unsupervised machine learning techniques to classify the programs based on the author's programming experience.
- We built our dataset by collecting programs from two regions: Asia and Europe. It would be interesting to collect the programs from other regions such as North America, South America, Africa and Australia and test our models on those programs.
- We used a T-test to find out the statistically significant features of this research. Other statistical analysis techniques, such as principal component analysis, can be used to determine significant features in the future.
- All the programs collected in our dataset were from the Codeforces.com. The source codes collected from Codeforces were the solutions to programming contest problems. There might be differences between the coding style of contest codes and regular codes. We plan to collect computer programs from other sources such as Github, where regular codes can be found.

# Bibliography

- [1] John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug? *28th International Conference on Software Engineering (ICSE)*, pages 361–370, 2006.
- [2] Rajeev K. Bali, Nilmini Wickramasinghe, and Brian Lehaney. *Knowledge Management Primer*. Taylor Francis e-Library, 2 edition, 2010.
- [3] M. Bicego and M. Loog. Weighted k-nearest neighbor revisited. *23rd International Conference on Pattern Recognition (ICPR)*, pages 1642–1647, 2016.
- [4] John Blackwell and Paul Gamble. *Knowledge Management, A State of the Art Guide*. Kogan Page, 2001.
- [5] Leo Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.
- [6] Steven Burrows and S. M. M. Tahaghoghi. Source code authorship attribution using n grams. *In proceedings of 12th Australasian Document Computing Symposium*, pages 32–39, 2007.
- [7] R. P. L. Buse and W. Weimer. Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4):546–558, 2010.
- [8] Naomi R. Ceder. *The Quick Python Book*. Manning, 2 edition, 2010.
- [9] Lynne M Connelly. t-tests. *MedSurg Nursing*, 20(6):341, 2011.
- [10] Steven Deutekom. Collecting source code samples for sociolinguistic research. *Technical Report*, 2019.
- [11] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.
- [12] Alberto Fernández, Salvador García, Mikel Galar, Ronaldo C. Prati, Bartosz Krawczyk, and Francisco Herrera. *Learning from Imbalanced Data Sets*. Springer, 2018.
- [13] Sachin S. Gavankar and Sudhirkumar D. Sawarkar. Eager decision tree. *2nd International Conference for Convergence in Technology (I2CT)*, pages 837–840, 2017.
- [14] Hemant Kumar Gianey and Rishabh Choudhary. Comprehensive review on supervised machine learning algorithms. *International Conference on Machine learning and Data Science*, pages 123–140, 2017.

- [15] Qiong Gu, Li Zhu, and Zhihua Cai. Evaluation measures of the classification performance of imbalanced data sets. *International Symposium on Intelligence Computation and Applications*, 2009.
- [16] Tanzim Ul Haque, Nudrat Nawal Saber, and Faisal Muhammad Shah. Sentiment analysis on large scale amazon product reviews. *International Conference on Innovative Research and Development (ICIRD)*, 2018.
- [17] Simon O. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1998.
- [18] Geoffrey Holmes, Andrew Donkin, and Ian H. Witten. Weka: A machine learning workbench. *Proceedings of ANZIIS '94 - Australian New Zealand Intelligent Information Systems Conference*, 1994.
- [19] Richard A. Hudson. *Sociolinguistics*. Cambridge University Press, 2 edition, 1996.
- [20] José Antonio Iglesias, Agapito Ledezma, and Araceli Sanchis. An ensemble method based on evolving classifiers: estacking. *2014 IEEE Symposium on Evolving and Autonomous Learning Systems (EALS)*, 2014.
- [21] C. A. Ratanamahatana J. F. Pane and B. A. Myers. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies*, 54(2):237–264, 2001.
- [22] M. Kamber J. Han and J. Pei. *Data Mining Concepts and Techniques*. Elsevier and Morgan Kaufmann Publishers, 3 edition, 2012.
- [23] Veena N. Jokhakar and S. V. Patel. A random forest based machine learning approach for mild steel defect diagnosis. *IEEE International Conference on Computational Intelligence and Computing Research*, pages 123–140, 2016.
- [24] Frank E. Harrell Jr. *Regression Modeling Strategies With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis*. Springer, 2nd edition, 2015.
- [25] Vaibhavi Kalgutkar, Ratinder Kaur, Hugo Gonzalez, Natalia Stakhanova, and Alina Matyukhina. Code authorship attribution: Methods and challenges. *ACM Comput. Surv.*, 52(1), 2019.
- [26] John D. Kelleher, Brian Mac Namee, and Aoife D'Arcy. *Fundamentals of Machine Learning for Predictive Data Analytics*. The MIT Press, 1 edition, 2015.
- [27] I. Krsul and E. Spafford. Authorship analysis: Identifying the author of a program. *Computers and Security*, 16(3):233–248, 1997.
- [28] W. Labov. The linguistic variable as a structural unit. *Washington Linguistics Review*, 3:4–22, 1966.



- [29] Yue Liu, Lingjie Hu, Fei Yan, and Bofeng Zhang. Information gain with weight based decision tree for the employment forecasting of undergraduates. *IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 2210–2213, 2013.
- [30] Mike Mirzayanov. The second revolution of colors and titles. <https://codeforces.com/blog/entry/20638>, Accessed on 2020-15-01.
- [31] Mike Mirzayanov. Codeforces rating system. <https://codeforces.com/blog/entry/102>, Accessed on 2020-31-08.
- [32] L. D. Misek-Falkoff. The new field of software linguistics. *ACM SIGMETRICS Performance Evaluation Review*, 11(2):35–51, 1982.
- [33] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.
- [34] M. Arthur Munson and Rich Caruana. On feature selection, bias-variance, and bagging. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2009.
- [35] M.Narasimha Murty and V.Susheela Devi. *Pattern Recognition: An Algorithmic Approach*. Springer, Universities Press, 2011.
- [36] Fariha Naz. Do sociolinguistic variations exist in programming? *Master's Thesis, University of Lethbridge*, 2015.
- [37] Fariha Naz and Jacqueline E. Rice. Sociolinguistics and programming. *5th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 74–79, 2015.
- [38] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in Neural Information Processing Systems*, 14, 2001.
- [39] Vu Nguyen, Sophia Deeds-Rubin, Thomas Tan, and Barry Boehm. A SLOC counting standard. *Center for Systems and Software Engineering, University of Southern California*, 2007.
- [40] W. O'Grady and J. Archibald. *Contemporary Linguistic Analysis: An Introduction*. Pearson Education Canada, 6th edition, 2008.
- [41] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, pages 2825–2830, 2011.
- [42] Chris Piech, Mehran Sahami, Daphne Koller, Stephen Cooper, and Paulo Blikstein. Modeling how students learn to program. *SIGCSE '12: Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 153–160, 2012.

- [43] Hongming Zhu Qin Liu, Xiaolong Li and Hongfei Fan. Acquisition of open source software project maturity based on time series machine learning. *10th International Symposium on Computational Intelligence and Design*, pages 296–299, 2017.
- [44] Md Mahmudul Hasan Rafee. Computer program categorization with machine learning. *Master's Thesis, University of Lethbridge*, 2017.
- [45] Jason D. M. Rennie, Lawrence Shih, Jaime Teevan, and David R. Karger. Tackling the poor assumptions of naive bayes text classifiers. *Twentieth International Conference on Machine Learning*, 2003.
- [46] Willi Richert and Luis Pedro Coelho. *Building Machine Learning Systems with Python*. Packt Publishing, 2013.
- [47] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (3rd ed.)*. Prentice Hall, 2009.
- [48] J. Fine S. Argamon, M. Koppel and A. R. Shimoni. Gender, genre, and writing style in formal written texts. *TEXT*, 23:321–346, 2003.
- [49] R. Horton S. Argamon, J. Goulain and M. Olsen. Vive la difference! text mining gender dfference in french literature. @ONLINE. *Digital Humanities Quarterly*, 3(2), 2009.
- [50] N. Schmit. *An Introduction to Applied Linguistics*. Hodder Education, London, England, 2010.
- [51] Aayushi A. Shah and Keyur Rana. A review on supervised machine learning text categorization approaches. *International Conference on Circuits and Systems in Digital Enterprise Technology (ICCSDET)*, 2018.
- [52] Eugene H. Spaffordl and Stephen A. Weeber. Software forensics: Can we track code to its authors? *Computers and Security*, 12(6):585–595, 1993.
- [53] Richard S. Sutton. *Reinforcement Learning*. Kluwer Academic Publishers, 1 edition, 1992.
- [54] Zengwei Tang, Hong Wang, Xiaobing Li, Xiaohui Li, Wenjie Cai, and Chongyuan Han. An object-based approach for mapping crop coverage using multiscale weighted and machine learning methods. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:1700–1713, 2020.
- [55] Ronald Wardhaugh. *An Introduction to Sociolinguistics*. BLACKWELL, 2 edition, 1995.
- [56] Ian Witten, Eibe Frank, and Mark Hall. *Data Mining Practical Machine Learning Tools and Techniques*. Elsevier and Morgan Kaufmann Publishers, 3 edition, 2011.

# Appendix A

## Python Library

### A.1 Reading Text Data

There are several libraries of Python to read text data. These text data used as input data for various experiments. Some of these libraries are given below:

- “BeautifulSoup” can be used to read data from XML files.
- Data can be imported from a Comma Separated Value (CSV) files using “pandas”.
- Text data can also be stored in json format. Data can be accessed by a Python program using the package “json”.
- Input data can also be stored in Microsoft Excel Worksheet. “xlrd” can import data from Excel files.

### A.2 Natural Language Toolkit

NLTK is the primary toolkit to process natural language [46]. There are various library functions in NLTK.

- Tokenization can be used to tokenize the text data. For instance, It isn’t raining will be tokenized to It, isn’t and raining.
- Stemming converts any word to its basic format. For example, isn’t will be converted to is not.
- There is a list of stopwords in NLTK. Words which do not have much impact in decision making such as am, is, are, a are listed as stop words.
- NLTK also defines several classifiers such as:
  - Decision Tree Classifier
  - Naive Bayes Classifier
  - Conditional Exponential Classifier

### A.3 Scikit-learn

Scikit-learn contains several functions defining the modern supervised and unsupervised machine learning algorithms [41]. This package contains functions defining:

- Supervised machine learning algorithms such as naive Bayes, Support Vector Machine, Ensemble Methods.
- Unsupervised machine learning algorithms like Gaussian Mixture Models, Clustering.
- Model selection and evaluation, such as cross-validation, model evaluation.
- Dataset transformations like feature extraction, pre-processing data.

### A.4 math

“math” module provides access to basic mathematical operations such as absolute value, floor value, factorial, modulus, sum.

### A.5 NumPy

NumPy is a package in Python [46]. It contains an N-dimensional array object, and it is a table of values of the same type. This package also includes useful linear algebra, Fourier transforms, and random number capabilities [46].

### A.6 Merits

The benefits of using Python are listed as follows [8]:

1. Python is an open-source programming language. That means we can download Python free. Also, the updates are always available online, and we can use them without paying anything. Moreover, anyone can modify the source code of Python.
2. Python has a vast built-in library. This feature is the most important advantage of using Python.
3. Python supports both procedural and object-oriented paradigm. To implement a real-world scenario, the object-oriented paradigm is required. A real-world situation cannot be described using one variable. For example, if we need to write a program about students, we need to have several variables such as name, id, GPA and some functions to calculate attendance, GPA. This scenario can be represented by creating a student class. Using Python, we can create a class and its objects.
4. Python is easy to learn in comparison to other programming languages. Computer programs written in Python are easily readable.
5. Python can be integrated with other programming languages such as C, C++, Java.

6. It is efficient to create prototypes using Python as it requires less coding compared to other programming languages. Prototypes may be built to test ideas on specific data.
7. Computer programs written in Python are portable. Python programs written in one machine can be executed in another machine. Other programming languages such as C, C++ do not allow portability.

## A.7 Demerits

Apart from the facilities, there are also some demerits of using Python [8].

1. Python takes more time to execute a program in comparison to other programming languages such as C, C++. Python programs are interpreted. Also, it runs a program line by line. Programs written in Python take more time to execute than programs written in compiled programming languages. Hence Python is not a good choice for a project, where execution time is the primary concern.
2. Implementation of parallel programming is difficult in Python as Python does not support multi-threaded programming. It supports single-threaded programs.
3. C, Java, Perl has more collections in their library in comparison to Python.
4. Python does not check the variable type during compilation. In Python, a variable can store any value such as integer or string or float. Some developers do not like this feature. Because it adds an additional check whether the variable is containing the desired type of value or not. For example, a variable “count” may be declared to store the number of iterations of for loop. It may start as count = 0. However, in the middle, if it has been assigned a value such as count = “Hello”, Python will allow it. If this is the case, before incriminating the value, the programmer has to check whether count stores integer or not.