Module Performance Prediction: Java programming flaws

Final Project Report

BSc Computer Science

Author: Nour Rabih

Supervisor: Dr. Raphael Jeffery

Student ID: 1863453

April 17, 2022

Abstract

Student data is vastly available for educational institutions, whether they are schools or universities. This data can be used in various ways to provide students with predictions that are high in accuracy and could massively facilitate their educational journey. Computer science students need to receive personalised feedback in order to be able to overcome the difficulties that they might experience during their studies. However, it is a highly complex mission for the academic staff to be able to go through each student's work and provide detailed feedback. Therefore, the involvement of machine learning would be a great way to facilitate the process for both students and teachers. Data mining is a practical machine learning tool that finds patterns in given data and provides a prediction of a particular outcome based on the total of the gathered data. This paper tests various data mining methods using computer science students' data – java static analysis reports – to predict whether a student is more likely to pass or fail a programming course. A neural Network, Decision Tree, Random Forest, Naïve Bayes models are created to classify students as "failing" or "passing". The models derived, however, were unable to accurately predict students' performance. Despite not having achieved the goal of identifying students that are bound to fail, we have gained insights into the Java programming practices of the students, which may help us identify students who need additional support to succeed with the completion of the course.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Nour Rabih

April 17, 2022

Acknowledgements

I would like to thank my helpful supervisor, Professor Dr. Jeffery Raphael. Having his supervision, support and learning from his astonishing experience in the field throughout this project made it much easier to move forward smoothly. My family was also very supportive throughout the project, and I am thankful for their encouragement.

Contents

Chapter 1

Introduction

Learning a programming language is a challenging requirement for students [1]. Around one-third of computer science students in the United Kingdom and the United States do not comprehend programming basics. Meanwhile, 35% of students fail their first programming course [2]. Teaching Computer Science students has been distinguished as a critical issue that has pulled in much attention. Giving personalised feedback to students on their programming assignments is a common concern. Students need detailed feedback in a timely manner. The feedback is needed for students to try and enhance their performance before the end of the semester in order to be able to achieve their academic goals. However, the large number of students in programming classes makes it difficult for the course staff to provide such feedback promptly [3].

According to the list of top ten programming languages in the world that was published in 2021 by IEEE, Java continues to dominate at the top of the rankings, both on its own merits and because of the huge existing base of code written in it [4]. In fact, living in a technology-driven world where nearly every field needs computerised frameworks and applications, there are a lot of career pathways and job openings available for Java Programmers. Therefore, supporting students in their learning journey and improving their completion and graduation rate is crucial to ensure that students continue to learn Java efficiently and get the necessary support to provide them with the bedrock for their careers.

Due to online learning and the digitised form of in-class activities and learning, universities collect large volumes of educational data from their students. Universities are rich with data; however, they lack information which results in undependable decision making. The greatest challenge is the successful change of vast volumes of data into knowledge to move forward the

quality of administrative choices [5].

Students submit programming assignments regularly in most universities. Whether it is an assignment or an in-class task, these submissions carry valuable information that could be helpful for students and teachers. The information is collected from students through 'Learning Management System (LMS)', Moodle, Blackboard, or GitHub. This information includes programming flaws in student code after submission, the date and time of the submissions, as well as code style and documentation. Some institutions ask for permission from students to access their IDEs through extensions to access the data of their programs. In addition to the information types mentioned above, these institutions can also access information about how students respond to different error types and how they progress over time.

In this paper, the data was collected through Keats (Moodle) from a first-year programming module that teaches Object-oriented Programming. Some of the goals of this module are to be able to critically assess the quality of (small) software systems, obtain sound knowledge of programming principles, as well as to be able to implement a small software system in Java. Four data mining techniques, Naïve Bayes, Decision Tree, Random Forest, and Neural network, will be examined. These techniques analyse the students' code that was collected from Keats identifying students bound to fail an introductory programming module.

1.1 Report Structure

In this report, the structure is as follows: first, background information about the topic is provided. A brief but clear explanation is given about Java standards, Static Analysis, and Data mining techniques, where the descriptions of decision tree, Random forest, Naïve Bayes, and neural network are introduced. The understanding of these topics is the bedrock of this project. Basic statistics about the data used in this paper are then presented. The background section is followed by the Existing Literature chapter, which provides insights of researchers that have already published papers about similar topics. The specification and implementation of this study are then presented, explaining the pre-processing, post-processing, and the classification methods implementations. The results are then evaluated, concluding that the data set is not informative to perform accurate predictions.The data set is then analysed, providing insights about the most common coding warning among students.

Chapter 2

Background

2.1 Java standards

In computing, a programming language has a specification or standard set of rules that target best practices, methods, and programming styles [6]. The primary aim of Java standards is to reduce the probability of run-time errors in programmers' code. The standards also cover readability, maintainability, and documentation [7].

Various computer scientists and institutions have made standards guidelines documents for Java [8]. Google has its own style document that includes all the features of the Java programming language as well as the standards. The document covers aesthetic issues of formatting, as well as conventions and programming practices. Jet Propulsion Laboratory, JPL, has also published a Java standards document that categorises them into: Critical, Important, and Advisory [7]. Some important and critical standards include concurrency, duplicate code, and logic errors.

2.1.1 Concurrency

Concurrency errors flag issues when implementing a multi-threaded execution system. Concurrency is the capability of running several parts of a program or various programs simultaneously [9]. The asynchronous performance of tasks improves the throughput and interactivity. 'Multithreading' is a term that is covered by Java concurrency. Multithreading is where the program is divided into multiple threads and runs concurrently [10]. A thread is a path of execution within a process. The process's resources like memory and open files are shared with the threads within a process. However, every thread has its own call stack [11]. All Object-Oriented pro-

gramming languages have a threading concept. Multithreading is important as it provides better utilisation of a single CP or multiple CPUs, and better user experience with regards to Responsiveness and fairness [12]. A simple example of issues when dealing with multiple threads of execution is shown in 2.1. “Explicitly calling Thread.run() method will execute in the caller’s thread of control. Instead, call Thread.start() for the intended behavior” [13].

Listing 2.1: Multithreading error [13]

```
Thread t = new Thread();
t.run();                // use t.start() instead
new Thread().run(); // same violation
```

2.1.2 Logic Errors

Logic errors are “situations where the programmer’s code compiles successfully and executes but fails to generate the expected output for all possible inputs” [14]. Logic errors are among the hardest for students to find due to their lack of debugging knowledge. Nevertheless, once found, they are generally easy to fix. A simple example of a logic error would be using a variable before assigning a value to it, as shown below. The output of the below code snippet would be a random number, as x was not initially assigned a value.

Listing 2.2: Logic error example

```
int x;
x = x + 4;
System.out.println("X=" + x);
```

The solution to the above logic error is quite simple, as shown in Listing 2.3, where x is simply initialised to zero. However, a simple error like the one above could easily take hours to spot by a beginner programmer.

Listing 2.3: Logic error Solution

```
int x = 0;
x = x + 1;
System.out.println("X=" + x);
```

In Java, primitive variables must be given a default value to avoid random numbers being stored in that variable. This form of error is common in object-oriented languages [15]. Logic errors might seem easy to avoid. However, they are tricky, and programmers tend to fall prey to them quite often. Other common logic errors that programmers encounter are incorrect operator precedence, defining the wrong count, assuming a condition is true when it is not, and relying on integer values to measure values [16].

2.1.3 Duplicate code

Duplicate code is "when a snippet of code appears multiple times throughout a codebase" [17]. Having duplicate code in a programme makes it harder to maintain the consistency of the code [18]. Meaning that if there is a bug in a duplicate code, it must be fixed in all other corresponding parts of the code that have a duplicate. Poor design is one of the reasons duplicate code is introduced to a program. An example of duplicate code is the repetition of methods that perform tasks that are almost identical. For instance, as shown in Listing 2.4, all four methods perform a movement but in a different direction.

This code needs to be refactored to the method in the form of listing 2.5. One way to avoid duplicate code is by dividing the code and its logic into smaller snippets that can then be reused throughout the codebase [19].

Maintaining these standards, along with many more, helps programmers avoid run-time errors. These standards can be checked in any piece of code using static analysis tools.

Listing 2.4: Duplicate code example [17]

```
public Position WalkNorth()
{
var player = GetPlayer();
player.Move("N");
return player.NewPosition;
}

public Position WalkSouth()
{
var player = GetPlayer();
player.Move("S");
return player.NewPosition;
}

public Position WalkEast()
{
var player = GetPlayer();
player.Move("E");
return player.NewPosition;
}

public Position WalkWest()
{
var player = GetPlayer();
player.Move("W");
return player.NewPosition;
}
```

Listing 2.5: Duplicate code Solution source[17]

```
public Position Walk(string direction)
{
    var player = GetPlayer();
    player.Move(direction);
    return player.NewPosition;
}
```

2.2 Static analysis

Static analysis tools are used by software system developers on a daily basis, and they are accredited as essential means of software quality control [20]. These tools assess the static structure and the software in abstract without executing it. Static analysis can be performed either automatically while developing the software, before committing the work to a version control system, or at the developer's request at any given time [21]. Rather than fixing syntax errors, they look for violations or issues with recommended coding practices or programming language standards [22]. There are various tools for different programming languages. FindBugs, Checkstyle, and PMD are common static analysers for Java. Although these tools were designed for software professionals, they can also be used for educational purposes [23]. In this paper, PMD is used to analyse the Java code of programming course students.

2.2.1 Checkstyle

The Checkstyle static analysis tool is an open-source tool designed to identify coding issues in Java code when it does not conform to specified coding standards or has incorrect formatting or layout [6]. It also detects potential coding problems, code that is hard to read, as well as code patterns that facilitate the introduction of bugs as the code is maintained and evolved [22]. Errors generated fall into several categories: Annotations, Block Checks, Class Design, Coding, Headers, Imports, Javadoc Comments, Metrics, Miscellaneous, Modifiers, Naming Conventions, Regexp, Size Violations, and Whitespace [24]. New checks can be written in Java to extend Checkstyle's functionality.

Sample output of running a checkstyle static analysis tool on sample code looks like this:

Listing 2.6: checkstyle sample output [25]

```
$ checkstyle -c checkstyle.xml Deck.java
```

```
Starting audit...
/Users/checkstyle/my/project/Blah.java:0: File does not end with
a newline.
/Users/checkstyle/my/project/Deck.java:23: Line has trailing spaces.
/Users/checkstyle/my/project/Deck.java:70: Line has trailing spaces.
Audit done.
Checkstyle ends with 3 errors.
```

2.2.2 FindBugs

FindBugs [26] searches for bugs in Java bytecode. This tool utilises a bug pattern-based approach. Bugs are given a rank between 1 and 20 and grouped into the categories: scariest (rank 1-4), scary (rank 5-9), troubling (rank 10-14), and of concern (rank 15-20). In comparison with static analysis tools that focus more on coding style, FindBugs tends to issue fewer warnings, but these warnings identify real problems [27]. The errors generated are categorised into several categories: bad practice, correctness, malicious code vulnerability, multithreaded correctness, performance, security, and dodgy code.

A Sample output of running a FindBugs static analysis tool on sample code looks like this:

Listing 2.7: FindBugs sample output [25]

```
\$ findbugs -textui .
M P UuF: Unused field: java.deck.Deck.classVar2  In Deck.java
M P UuF: Unused field: java.deck.Deck.instanceVar3  In Deck.java
M D UuF: Unused public or protected field: java.deck.Deck.instanceVar2
In Deck.java
M D UuF: Unused public or protected field: java.deck.Deck.classVar1
In Deck.java
M D UuF: Unused public or protected field: java.deck.Deck.instanceVar1
In Deck.java
Warnings generated: 5
```

2.2.3 PMD

Rather than focusing on style, PMD emphasises programming flaws and complex code [13]. PMD offers a wide range of rules that fall into a variety of categories. These categories are:

Code Style, Design, Documentation, Error Prone, Multithreading, Performance, and security. PMD can also be extended by adding new checks [22]. It finds these common programming flaws in Java, JavaScript, Salesforce.com Apex and Visualforce. Additionally, it includes a copy-paste-detector that finds duplicate code in Java, C, C++, C#, Groovy, PHP, Ruby, Fortran, JavaScript, and various other programming languages. PMD stands for nothing directly, but it goes by several unofficial names, the most appropriate of which is Programming Mistake Detector [6].

Here's an example of PMD running through some code:

Listing 2.8: PMD sample output [25]

```
$ pmd pmd -R java-basic,java-unusedcode -d Deck.java
/Users/pmd/my/project/Deck.java:35: Avoid unused private fields
such as 'classVar2'.
/Users/pmd/my/project/Deck.java:47: Avoid unused private fields
such as 'instanceVar3'.
```

2.3 Data mining

Data mining is the discovery of new observations from existing databases. It is the science where statistics, machine learning, data management and databases, pattern recognition, and artificial intelligence intersect [28]. The process of extracting new correlations, patterns, and trends is done by using pattern recognition technologies, as well as statistical and mathematical techniques on the given data sets [29]. Data mining requires very large data sets. The production of huge databases has been possible due to the advancement of digital data acquisition and storage technology [28]. Data mining is a "secondary" type of data analysis.. That is because its objective has no role in the data gathering approach. The data used in data mining analysis has typically been collected for other purposes [28].

Various data mining techniques such as classifications, clustering, associations, statistical techniques, text mining techniques, and mathematical models have been acknowledged by Researchers [30]. The most common data mining techniques are those used for description and prediction [30]. Classification is a data mining technique used to predict the class of an instance given the values of other variables. It is a major technique in data mining and is widely used in various fields [31]. Various examples of classification methods are Decision tree, Random forest, Naive Bayes, and Neural network.

2.3.1 Decision Tree

Extracting patterns and discovering features in large data sets that are of great significance for predictive modelling can be done with decision trees [32]. Decision trees are important predictive modelling tools because of their simplicity [33]. They have become a popular choice for many researchers to create classification or regression models, as they are easy to understand and create [33]. "A decision tree is a flowchart-like tree structure, where each internal node (nonleaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node" [34]. An individual is classified into a class; in this case it classifies whether a student is going to "pass" or "fail". The results obtained by a decision tree model are more comprehensible than "black box models" like neural networks (which will be explained in section 2.3.3).

As S. B. Kotsiantis explained in his paper [35], a collection of previously acquired labelled data is required for a decision tree. The algorithm finds patterns in the data by recursively determining the best division of instances into separate classes, forming a tree. The best division is partitioned based on the 'split'. There are various splitting criteria that can be applied, including information gain, gain ratio, and Gini index. In this study, two criteria, namely, information gain, and Gini index have been used:

1. **Gini index**: is the measure of impurity in a node. Its formula is:

$$Gini(D) = 1 - \sum_{i=1}^{J} (p_i)^2 \tag{2.1}$$

   Where p is the distribution of the classes in the node and J is the number of classes in the node.

2. **Information gain**: minimises the needed information for the classification and reflects the least randomness [34]. The variable that has the largest Information gain, or the least 'entropy' is chosen to split the node. Entropy is the measure of Randomness in the system. A pure node has an entropy of 0. If a node is pure, its entropy is 0. Bigger entropy value means less purity of the node. The entropy of a node is calculated for all attributes. The formula for Entropy is:

$$entropy = -\sum_{i=1}^{c} (p_i) \log_2 (p_i) \tag{2.2}$$

   Where C is the number of classes present in the node and p is the distribution of the class

in the node [34].

2.3.2 Random Forest

A Random Forest (RF) "is an ensemble of tree-structured classifiers" [36]. It is similar to a decision tree in that the RF uses tree predictors to combine trees based on a random vector [37]. Each tree of the forest casts a vote, appointing each input to the class label with the most votes. This method is fast, robust to noise and is a successful ensemble method for detecting nonlinear patterns in data. It is capable of handling both numerical and categorical data. Random Forest has the advantage of not over fitting, even when more trees are added [36].

In order to produce the random forest, we need two parameters: Ntree and Mtree. Ntree is "the number of decision trees to be generated", and Mtree is " the number of variables to be selected and tested for the best split when growing the trees". In light of RF classifier's high computational efficiency and lack of overfitting, Ntree can be as large as possible [38]. A typical value for Mtry is the square root of the number of input variables [39]. All the calculated class assignment probabilities of all the trees are averaged (using the arithmetic mean) to arrive at the final classification decision. A new unlabelled data input is evaluated against each tree in the ensemble to determine its class membership. The class that receives the most votes is selected as the final membership class. In Ghosh et al. [40, 41], Mtry is set to the total number of variables available to the algorithm, but this approach increases computation time since the algorithm has to calculate the information gain from all variables used to split the nodes.

2.3.3 Naïve Bayes

Naïve bayes is a Bayesian classifier which is a statistical classifier. Statistical classifiers are used to calculate the probabilities of the class membership of given records [5]; in this paper, it is the probability of either passing or failing. Bayesian classification is based on Bayes' theorem, shown below as equation 2.3.

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)} \tag{2.3}$$

In the above equation, P(H|X) is the posterior probability of H (the target) conditioned on X (the predictor) and P(H) is the prior probability of H. As a result of Bayes' theorem, we can calculate the posterior probability, P(H|X), based on P(H), P(X|H), and P(X).

Studies have found naive Bayesian classifier to be comparable in performance with decision

tree and selected neural network classifiers. When applied to large databases, they have shown high accuracy and speed. It is assumed in Naïve Bayesian classifiers that features are independent in a given class. This assumption greatly simplifies the computations and is called class conditional independence. It simplifies calculations involved and, in this sense, is considered naive [34, 42].

In Naïve bayes $P(C_i|X)$ $P(C_i)$ is maximised, as identified in eq. 2.4. where C is the class and X is a tuple of the attributes.

$$P(C_i|X) = \frac{P(X|C_i)P(c_i)}{P(X)} \tag{2.4}$$

Given a large number of attributes, it would be extremely computationally expensive to compute $P(X|C_i)P(C_i)$. Therefore, to reduce computation, the naive assumption of class conditional independence is assumed, which implies that there are no dependencies among X's attributes. Thus, it is calculated as shown below in eq. 2.5.

$$P(X|C_i) = \prod_{k=1}^{n} P(x_1|C_i) * P(x_2|C_i) * P(x_3|C_i) * ... * P(x_n|c_i) \tag{2.5}$$

In this paper, the values of the tuple X are continuous. It is generally assumed that continuous-valued attributes have a Gaussian distribution of mean and standard deviation. If the data is continuous-valued, we must do a bit more math, but the calculation is straightforward. The calculation is shown in eq ...

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

So that,

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}) \tag{2.6}$$

Although Bayesian classifiers, in theory, have the lowest error rate compared to all other classifiers, they are not always true in practice due to inaccurate assumptions about conditional independence [34].

2.3.4 Neural Network

Neural Network is a more sophisticated classification algorithm. The concept of neural networks is based on biology, specifically on how neurons are connected in the brain [34]. Scientists and neurobiologists developed and tested computational analogues of neurons to initiate the field

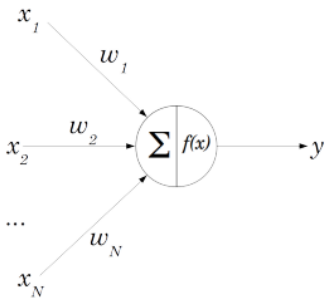

Figure 2.1: Perceptron

of neural networks. Neuronal networks consist of connected input/output units with weights assigned to each connection. The network adjusts its weights during the learning phase in order to predict the correct class label of the input tuples.

Due to their popular characterisation as "black boxes", neural networks have been criticised for their interpret-ability. Humans find it difficult to understand the symbolic meaning behind learned weights and "hidden units" in networks, for example. Therefore, neural networks were less attractive for data mining in the beginning. However, neural networks benefit from being able to tolerate noisy data as well as classify patterns without prior training. It is useful when you are not familiar with the relationship between attributes and classes. They are appropriate for inputs and outputs with continuous values.

Layers are made up of units/neurons. Inputs are simultaneously fed into units that constitute the input layer. The learning process consists of two stages: forwards propagation and backward propagation.

1. **Forward Propagation:** The inputs pass through the input layer and are weighted before being fed simultaneously to the second layer called a hidden layer. Figure 2.1 illustrates an example of a hidden layer or output layer unit. A hidden or output layer's net input is computed as the linear combination of its weighted inputs. As each unit in the hidden and output layers receives its net input, it applies an activation function to the neuron it represents. The activation function maps the large input domain onto the range of 0 to 1. Sigmoid, equation 2.7, is an example of an activation function. Where I is the calculated net input to the unit j and O is the output of the unit j.

$$O_j = \frac{1}{1 + e^{-I_j}} \tag{2.7}$$

2. **Backward Propagation:** In backpropagation, the weights are modified for each training tuple in order to minimise the mean squared error between the network's prediction and the actual result. Those changes are made in a backwards fashion, from the output layer, down through each hidden layer until the first hidden layer is reached. The error of the network's prediction is propagated backward by updating the weights and biases. Following are the equations that are used to update weights, where $\Delta w_{i,j}$ is the change in weight $w_i,_j^1$ and l is the learning rate:

$$\delta w_{ij} = (l) Err_{ij} \tag{2.8}$$

$$w_{ij} = w_{ij} + \Delta w_{ij} \tag{2.9}$$

2.3.5 Data sets Basic Statistics

Two data sets are used in this paper. The statistics of each data set is presented in this section. Each data set consists of the report results of the PMD analysis on a coding assignment. Where the header contains the warning/error types, and each student is associated with the number of instances of each warning type. Information about the individuals, like age, level, and gender, are not considered in this study. The target feature that the classification algorithms will predict is the 'grade' (pass or fail). The two data sets are not related, and the submission dates and times are not considered.

Data set one

There are 414 students in this data set and 33 warning/error types (the feature names). The error types include UnnecessaryImport, NoPackage, and UselessParentheses. The mean of the students' grades is 0.458937 with a standard deviation of 0.498914. As demonstrated by figure 2.2, the percentage of the passing students is 54.11%, with the remaining 45.89% of the students failing at the end of the academic year. The figure suggests that the data set is balanced; the number of passing students is close to the number of the failing students.

Data set two

There are 404 students in this data set, and 59 warning/error types (the feature names). The error types include the ones in data set one as well as additional ones like FinalField-

CouldBeStatic, DoubleBraceInitialization, and UselessOverridingMethod. The mean grade of the students' grades is 0. 0.465347 with a standard deviation of 0.499416. Figure 2.3 shows the percentage of failing and passing students. 53.47% of the students passed, while 46.53% failed. There is very minor difference between the class counts, making the data set balanced.

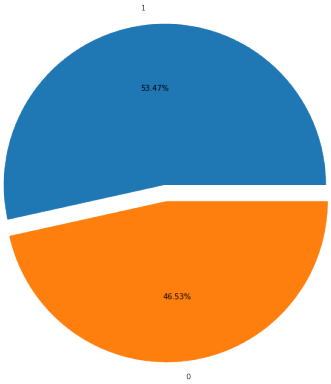

Figure 2.2: Data set 1 pass/fail percentage

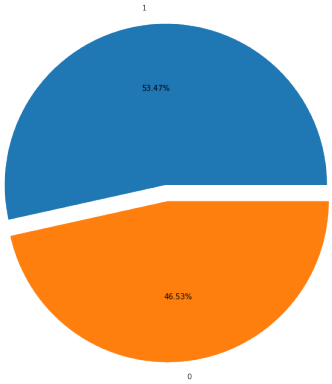

Figure 2.3: Data set 2 pass/fail percentage

The blue part of the charts represents the value 1, which is pass, and the orange part represents 0, which is fail.

Both data sets are extracted by the PMD tool from students' code. As stated in section 2.2.3, PMD offers a wide range of rules that fall into a variety of categories. Table 2.1 lists all the warnings that are included in data sets one and two by their category. The warnings are highlighted in yellow, green, and orange representing being present in data set one, two, or both correspondingly.

Code style	Best practices	Design	Error prone	Documentation	Multithreading	Performance
Unnecessary Import	UnusedFormalParameter	SingularField	AssignmentInOperand	Uncommented EmptyMethodBody	Unsynchronized StaticFormatter	Optimizable ToArrayCall
NoPackage	UnusedPrivate Field	LogicInversion	CloseResource	Uncommented EmptyConstructor	NonThread SafeSingleton	
LocalVariable NamingConventions	LooseCoupling	SimplifyBoolean Returns	EmptyStatement NotInLoop			
UselessParentheses	ForLoopCan BeForeach	UseUtilityClass	EmptyIfStmt			
ControlStatement Braces	UnusedLocal Variable	FinalFieldCould BeStatic	Idempotent Operations			
UnnecessaryFully QualifiedName	UnusedPrivate Method	UselessOverriding Method	AssignmentTo NonFinalStatic			
FormalParameter NamingConventions	MissingOverride	SimplifyConditional	UseLocaleWith CaseConversions			
MethodNaming Conventions	UseCollection IsEmpty	ClassWithOnly PrivateConstructors ShouldBeFinal	EmptyCatchBlock			
Unnecessary LocalBeforeReturn	OneDeclaration PerLine		AvoidBranching StatementAsLastInLoop			

Table 2.1 continued from previous page

Code style	Best practices	Design	Error prone	Documentation	Multithreading	Performance
Unnecessary Constructor	LiteralsFirstIn Comparisons		EmptyStatement NotInLoop			
Unnecessary Return	SwitchStmts ShouldHaveDefault		UseLocaleWith CaseConversions			
ClassNaming Conventions	DoubleBrace Initialization		CompareObjects WithEquals			
IdenticalCatch Branches	DefaultLabelNot LastInSwitchStmt		UseEqualsTo CompareStrings			
Unnecessary Modifier			OverrideBoth EqualsAndHashcode			
			MissingBreakInSwitch			
			AvoidCatching Throwable			
			EqualsNull			
			AvoidUsingOctalValues			
			ReturnEmptyArray RatherThanNull			

Table 2.1 continued from previous page

Code style	Best practices	Design	Error prone	Documentation	Multithreading	Performance
			SuspiciousEquals MethodName			
			BrokenNullCheck			

Table 2.1: warnings categories of data sets one and two

Chapter 3

Literature Review

3.1 Java standards and Static Analysis

In a study conducted by Ashfaq et al. [43], static analysis tools were categorised into 7 major categories according to the type of rules enforced by a tool. The categories are style, concurrency, exceptions, quality, security, dependency, compatibility and general methods of static code analysis. The study analyses a number of coding standard analysers including, PMD, Checkstyle, and FindBugs. The result of the comparison is portrayed in table 3.1. Though PMD detects issues that are similar to those detected by Checkstyle and FindBugs, it places less emphasis on formatting conventions and more emphasis on programming guidelines, design recommendations, and recommended programming practices.

Tool	**Input**	**Rules**	**Availability**	**Extensibility**	**errors**	**output**
Checkstyle	Source code	General, Style	Open source	possible	Naming conventions, length, line, white spaces errors	List, HTML , XML file
FindBugs	Source code	General, Style	Open source	possible	Potential bugs and performance issues	List, XML
PMD	Source code	General, Style	Open source	possible	Potential problems, Dead code, possible bugs, duplicate code, and over-complicated expressions	List

Table 3.1: Analysis of Tools [43]

In 2017, A. Abdallah and M. Al-rifaee [19] developed a comparative study of the most common Java standards. In this study, it is determined that there are numerous rules and guidelines for writing Java code. Nevertheless, each covers the main features of Java in a

similar manner. Additionally, Static analysis tools were presented, and it was concluded that most static analysis tools are based on the best two Java standards by SUN and Google. However, Along with Sun and Google standards, PMD uses rules that are approved by other developers. This makes it a tool with plenty of rules to apply.

3.2 Classifiers

Many studies have been conducted to predict student performance. In a study conducted by Digna S. Evale et al. in Japan [44], classification rules were used to capture the pattern of data. Three classification methods were used: Tree Classifiers, Bayes Classifiers, and Rule classifiers. The data included 8 attributes: the users' age, gender, course, section, schedule, and grades in Programming courses. The accuracy levels of the three methods were compared with the Tree Classifiers, Bayes Classifiers, and Rule classifiers, having accuracy rates of 94.74%, 92.84%, and 94.21% respectively. Various tree classifier types were tested, and it was deducted that J48, which is an improved implementation of C4.5 Decision tree, has the best precision rate. Among the various tree classification algorithms that can be used for performance prediction, J48 is considered one of the most widely-used today. That is due to its high accuracy of prediction, optimised diagram, as well as the easy interpretation of rule sets.

The work in "Annete: An Intelligent Tutoring Companion Embedded into the Eclipse IDE" [45] implemented a Neural Network Embedded Tutor for Eclipse. Annete is an acronym for A Neural Network Embedded Tutor for Eclipse. The neural network with a supervised learning algorithm is used to give student feedback. Where the Eclipse plugin reads the student code and returns personalised advice/feedback on what they should do while they work on Java programming coursework based on the trained neural network.

Annete's brain was built using a multi-layer perceptron neural network with a back propagation learning algorithm. The back propagation uses A sigmoid transfer function. Sixteen nodes with the data collected from students are inputted into the Neural network. The hidden layer has 7 nodes, and the output layer represents the feedback the student will receive. The input given to the neural network is of type "double". Before the input data enters "Anette's Brain", it goes through various steps before it can enter the Neural network. The data is pre-processed, normalised, and then sent to the neural network. The output is also normalised, therefore, when the neural network outputs a number, it must be scaled to match the set of possible outputs. The outputs are given as "codes" of multiples of 10, so for instance the code 20 represents a "Problem with recursion identified".

The project uses Java, using several existing libraries. The Neuroph library was selected, which is an open-source Java library designed to simplify the implementation of neural networks. Bayesian networks and linear regression were also considered for the implementation of "Anette"; however, neither of these well-suited their use case.

Other studies [5, 46, 47] have implemented the Naive Bayesian algorithm and have concluded that it gives greater accuracy than other methods such as Regression, Decision Tree, Neural networks etc. The accuracy level of the Naive Bayesian algorithm was over 80% in all three studies. However, the result of this classifier is not actionable as one cannot know what the factor behind the success or failure of students is. It is not easy to interpret, therefore the educational sector cannot take accurate action after retrieving the performance prediction [5].

A study conducted in 2018 [37], compares several methods for prediction including Decision Tree, Random Forest, Naïve Bayes, Neural Network, and k-Nearest Neighbour (kNN).A total of 7 semesters' worth of data for Fall 2012 through Fall 2017 were analysed. A total of 383 students representing 18 class sections from software engineering classes at San Francisco State University were examined. The final grades were treated as the target variable, while all the other attributes were considered features. In the study, pre-processing techniques were used with feature selection, and the experiment was focused on milestones 1 to 5 and their influence on final grades. Comparing the models is done by analysing the area under the curve (AUC), classification accuracy (CA), F-measure, and Log Loss. Precision and recall metrics are then used to compute F1.The results of the study are shown in table 3.2. It suffices to conclude that the Neural Network outperforms the other model based on the scoring methods, especially in terms of model prediction efficiency and classification accuracy.

Method	**AUC**	**CA**	**F1**	**Precision**	**Recall**	**Log loss**
kNN	0.641	0.593	0.596	0.602	0.593	2.193
Random Forest	0.596	0.553	0.546	0.542	0.553	0.933
Neural Network	0.659	0.587	0.591	0.608	0.589	1.280
Naive Bayes	0.591	0.553	0.555	0.593	0.553	3.747

Table 3.2: Performance metrics [37]

Another study [2] derived a model to predict at-risk beginner Java programmers. The linear regression model derived was not accurate, however, insights concerning students' compilation behaviour were attained. It was found that encountering errors at an early stage can negatively affect the average and "atRisk" students' midterm scores. On the other hand, scores of high-performing students were not affected by the errors.

Recurrent Neural Network (RNN) was implemented for predicting final grades of students

in a study conducted by Okubo et. al [48]. The method was applied on the collected log data from 108 students.The collected data was from the weekly submitted reports, quiz, and logbook. Recurrent Neural Network was particularly used to treat time series data of each week in a course. The results of the RNN were compared with multiple regression analysis. Using log data,it is observed that the accuracy of the RNN is above 90% through the 6th week, but less than 90% by multiple regression through the 9th week. Thus, RNNs can be said to be capable of predicting final grades of students with a high degree of accuracy.

Educational data mining has been broadly used to predict student performance and create strategies to improve student performance. Machine learning has been implemented in most studies for interventions, however, the data being analysed is students' background information as well as grades. The use of data mining in evaluating student performance in learning software, using software-specific data like code errors or java standards, is imprecise.

Chapter 4

Specification & Implementation

4.1 Specifications

This paper examines four data mining techniques- Decision tree, Naïve bayes, Neural network, and random forest - to predict whether students are bound to fail an introductory programming module. The data obtained is anonymous student code from previous years. The data is extracted from the code. The code is analysed using PMD, the static analysis tool, that returns a .txt report containing all the issues found in students' code. This report is then inputted into the data mining algorithm. The performances and the prediction accuracy rates of the four methods are compared to extract the best prediction method.

The language used for creating the models is Python3. Python is one of the most popular programming languages for machine learning [49]. A vast number of useful libraries for machine learning have been developed in its open-source community. In this project, Scikit learn library, version 1.0.1, is used for handling the data mining tasks. Scikit-learn has been increasingly popular for machine learning related applications as it has four features that make it attract attention among machine learning programs [50]. The four features are:

1. Inclusiveness of machine learning methods.
2. Computation efficiency optimisation of machine learning methods' algorithm implementation.
3. Quality documentation, bug tracking, and quality assurance.
4. Unified input/output data convention making the switching from one method to another almost extremely easy.

4.2 Implementation

A data mining operation has two stages apart from the classification modelling process itself which are pre-processing and post-processing [51]. The term ”error” is used in this paper, however it does not refer to actual java errors. It refers to ”coding standard warning” that are returned by the PMD static analysis tool.

4.2.1 Pre-processing

Pre-processing consists of multiple steps:

1. Identify the data mining problem: In this case we know that it is a binary classification problem to predict whether students are bound to fail an introductory programming module.

2. Identify the source of the data: That is university students' Java code from previous years.

3. Generate a data subset of the identified data set: That is done by removing all unnecessary information in the data set (**cleaning stage**) and **transforming** it to an appropriate format.

Data cleaning is where unnecessary data is deleted, and format inconsistencies are fixed. The data file in this project, which is the .txt file that was extracted from the PMD analysis, was cleaned by creating a python program that extracts the student number along with the error types. Format inconsistencies were fixed manually by making sure that the python code can run with no errors to extract all the needed data. Only a few inconsistencies were found in the given data, therefore performing it manually was not a time-consuming task. The final grade was also extracted from another document, disregarding other unwanted data from that document.

Data transforming in this project consisted of two stages, first transforming the data after cleaning into a 'Dictionary' for each student with each error and the number of its occurrences. The second stage of data transforming is transforming the data in a way that the data classification algorithm would be able to analyse as an input. The input of classification algorithms is required to be in a similar format of a spreadsheet, and the input in all columns should be completely numeric [52]. Therefore, the python program wrote to a csv document the data in

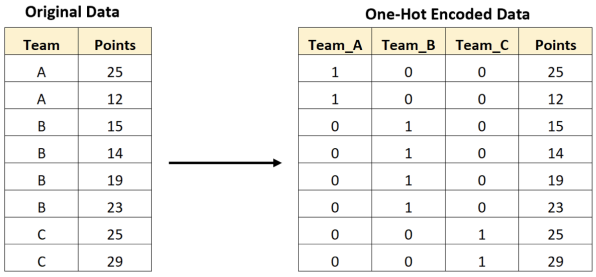

Figure 4.1: One-hot encoding example [53]

a form of a table with all the error names as a header and the number of occurrences of each error for every student along with the grade in a pass or fail format, where 0 represents a grade under 40, and 1 represents pass, a grade greater than or equal to 40.

The above transformation of the input data was inspired by one-hot encoding. The implementation of one-hot encoding was considered however, it was not suitable for the scope of this project due to our data set format (continuous data). One-hot encoding is utilised to transform categorical variables into a format that can be readily used by machine learning algorithms. The concept of one-hot encoding is to give values of 0 and 1 to new variables that represent the original categorical values as shown below in figure 4.1 where one-hot encoding is performed on 'team' variable.

Similarly, the data used in this project was transformed to the format shown below in figure 4.2, where each cell in the table represents the number of occurrences of each error in a student's code. In this manner, the transformed data meets the required format of being completely numeric and is ready to be inputted into the classification algorithm. As the figure suggests, it is not possible to implement one-hot encoding as each cell requires a different value that could be 0 or any other positive number. Another format,which is more like one-hot encoding, was also considered. Where under each variable there's a value of either 0 or 1, which represents either the presence (could be one occurrence or various) of this error type in the student's code or its absence. This is shown in figure 4.3. This format was tested on the classification methods, to check whether it will enhance the accuracy rate. However, it seemed to produce very similar outputs as the first format. Therefore, the transformation format used throughout this project is the one shown in figure 4.2.

grade	UnnecessaryImport	NoPackage	UnusedFormalParameter	LocalVariableNamingConventions	UnusedPrivateField
1	15	4	1	3	4
1	15	5	1	3	2
0	15	4	1	3	5
0	15	4	1	3	4
0	15	4	1	3	6
1	15	4	1	3	4
0	9	4	1	3	1
1	15	4	1	3	3
0	15	4	1	3	4
1	15	4	1	3	4

Figure 4.2: Transformed data

grade	UnnecessaryImport	NoPackage	UnusedFormalParameter	LocalVariableNamingConventions	UnusedPrivateField
1	1	1	1	1	1
1	1	1	1	1	1
0	1	1	1	1	1
0	1	1	1	1	1
0	1	1	1	1	1
1	1	1	1	1	1
0	1	1	1	1	1
1	1	1	1	1	1
0	1	1	1	1	1
1	1	1	1	1	1

Figure 4.3: Alternative format of the transformed data

4.2.2 classification algorithms

As stated above Scikit-learn has a unified input format, so the pre-processing is also unified for all classification methods that are used in this study. The implementation and results of the four classification methods will be explained in the section below.

Decision tree

Scikit-learn implements the CART algorithm in an optimised way, however, it doesn't support categorical variables yet. The variables in our datasets are continuous, therefore the use of Sklearn's CART algorithm is suitable for this project. In CART, binary trees are constructed using the feature and threshold that will yield the greatest information gain at each node [54]. Three parameters of the decision tree were altered seeking for the best outcomes: criterion, splitter, and max_depth.

- **Criterion:** A measure of the quality of splits. The criteria supported are "gini" for the Gini impurity and "entropy" for the information gain..
- **Splitter**: A method of choosing the splits at each node. There are two supported strate-

gies: “best” for selecting the best split, and “random” for selecting the best random split.

- **max_depth**: A tree’s maximum depth.

Trees with depths from 2 to 100 were created to observe the accuracy rate change. Both “criterion” and “splitter” were changed, however, there were no significant improvements in the accuracy rates.

Random Forest

Two parameters of the Random forest were altered seeking for the best outcomes: max_features and n_estimators.

- **n_estimators**: Forest trees count.
- **max_features**: the number of features to consider when looking for the best split.
  - If “auto”, then max_features=sqrt(n_features).
  - If “sqrt”, then max_features=sqrt(n_features) (same as “auto”).
  - If “log2”, then max_features=log2(n_features).
  - If None, then max_features=n_features.

The above definitions are acquired from the random forest user guide from the Scikit learn website [54]. Those parameter values were set to control the complexity and size of the trees to reduce memory consumption. When the parameters for determining the size of the trees are left at their default values, they can result in fully grown and unpruned trees that are very large on some data sets [54].

Random forests with n number of trees, where n is from 100 to 2000 (multiples of 100) were created to provide the mean of the accuracy rates. “max_features” was also changed however, there were no significant improvements in the accuracy rates. Similar to the decision tree results, the accuracy rate yields no promising results in predicting the students’ performance, as it will be discussed in the results section (section 5). The models were also created on both data sets and the results were also similar.

Naïve Bayes

The implementation of Gaussian Naïve Bayes using the Scikit learn library is straightforward. The Gaussian Naïve Bayes was chosen over Multinomial, complement, Bernoulli, and other

Naïve Bayes classifiers, due to the nature of the datasets used in this study, which contain continuous values data. Gaussian Naïve Bayes has two parameters that can be altered, priors and var_smoothing [54]. However, both are remained unchanged as we do not have a prior probability of each class. As it will be discussed in the results section, the accuracy rate is also neither promising nor impressive, as it is 0.517. Given that the dataset target variable is either 0 or 1, having a 50% accuracy rate is similar to a coin toss. Therefore, the predictions of Naïve Bayes were also inaccurate for our test case.

Neural Network

The Class MLPClassifier "implements a multi-layer perceptron (MLP) algorithm that trains using Back pragation" [54]. Tuning a neural network is complex task. Knowing the number of layers to be used as well as the number of nodes to control the architecture or topology of the network is not a straightforward task. There are various other parameters that can be altered when creating a network, those include [54]:

1. **Activation**: Activation function for the hidden layer. Takes 'identity', 'logistic', 'tanh', 'relu' as values.

2. **Solver**: The solver for weight optimisation. Takes 'lbfgs', 'sgd', 'adam' as values.

3. **learning_rate**: Learning rate schedule for weight updates. Used only when solver='sgd'. Takes 'constant', 'invscaling', 'adaptive' as values.

The above definitions are acquired from the Neural network user guide from scikit learn website [54].

There are no clear rules as to what the optimum number of hidden layer units should be. The design of a network is a trial-and-error process and may affect the accuracy of the trained network. Finding reasonable values for all the parameters is best done using an automated techniques like GridSearchCV. In the parameter grid, a number of factors are fed into the program and the best combination of those factors is determined by a scoring metric of your choice (accuracy, f1, etc)[55]. GridSearchCV was implemented to extract possible best hyper parameters for the model. The best extracted Neural Network has 2 layers ,with 20 and 4 neurons, respectively. The activation method for the neural network is 'tanh', with stochastic gradient descent solver.

In addition to the initial pre-processing process that was done in section 4.2.1, In neural networks, normalising the inputs for each attribute measured in the training tuples will provide

a speed boost during the learning phase. A normalised input value typically falls between 0.0 and 1.0. This was done through importing "pre-processing" package from Scikit library and using preprocessing.normalize(X). Where X is the input training data. While it was a expected to boost the accuracy rate, the changes seemed to be unnoticeable.

4.2.3 Post-processing

Post-processing is the last stage of data mining, in which the results of the data mining method are further processed. Knowledge filtering, evaluation, and visualisation are used to showcase the knowledge extracted in a meaningful and fathomable manner for users[56].

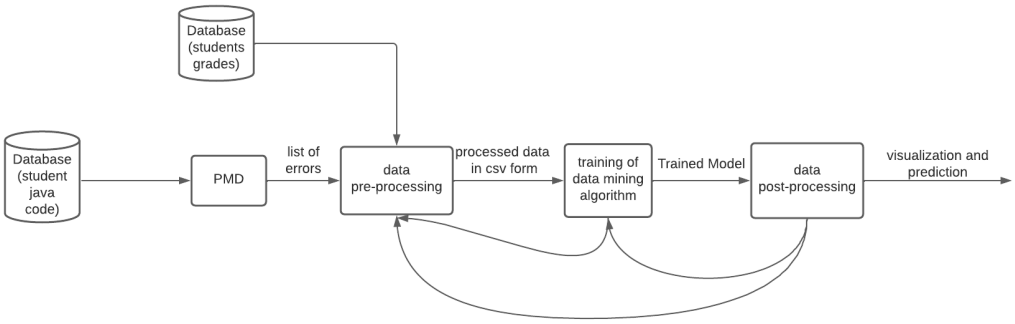

Figure 4.4: The data mining process

Figure 4.4 visualises the whole process of predicting students' performance. As shown at the last stage, post-processing, the process could go back to training or pre-processing. This is because the outcome of the classification method is evaluated in the post-processing stage. Going back to previous steps depends on the acquired results. If the results are not as expected various methods can be performed for an attempt to improve the prediction process. These methods include input normalisation, feature selection, and model modification.

The evaluation of the implemented classification algorithms is done in section 5.2. After the evaluation is done, knowledge filtering is applied to try and improve the accuracy of the models. This is done is section 5.2.1.

Visualisation of Decision Tree

Visualisation is "a way of representing structural information as diagrams of abstract graphs and networks" [57]. In Python, we can create various graphs and plots using different visualisation libraries. The use of modules like seaborn, matplotlib, and bokeh allows for highly interactive, scalable, and visually attractive visualisations. However, we cannot create nodes and edges to

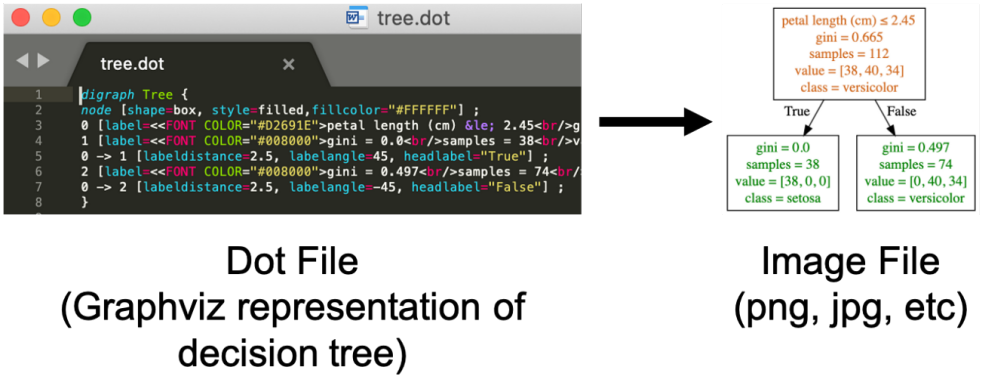

Figure 4.5: Dot file to image file [60]

connect different diagrams, flowcharts, or graphs using these libraries. Graphviz allows us to create graphs and connect them with edges and nodes. It creates source code of the graph in DOT language of the Graphviz software[58]. Decision trees are visualised by Graphviz. First, a DOT file of the decision tree is created. Then, it is converted into an image file as shown in figure 4.5 . The image file is created using PyDotPlus. PyDotPlus provides a Python Interface to Graphviz's Dot language [59].

A sample visualisation of a decision tree with a depth of 5, after training, is shown in figure 4.6. The visual is user friendly, as users can understand the logic behind the predicted classification produced by the algorithm.

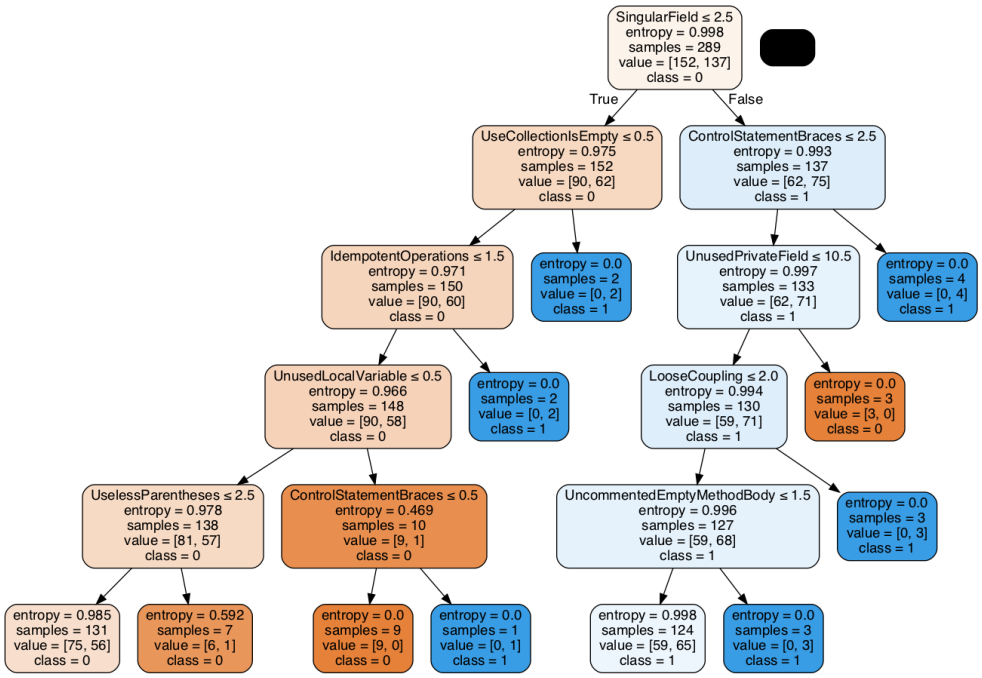

Figure 4.6: Decision Tree Visualisation

Similarly, the visualisation of random forest is also done using graphviz, where each individual tree is visualised separately. The visualisation of a Random forest is an ensemble of decision trees as shown in figure 4.6. In this example, there are 100 to 2000 (multiples of 100) trees in an RF. This would be too big to be shown in this paper.

Chapter 5

Results & Discussion

5.1 Results

The results of running the classification algorithms are summarised as follows. The metrics used to compare the models are classification accuracy, precision, F-measure, and recall. The overall result is not impressing, the numbers produced by all classification techniques were low. The produced classification is not reliable or accurate. Both data sets were tested using the four classification methods. The result of data set one are show in table 5.1 . The difference between the classification accuracies of all methods is very minor, making it hard to decide on which method performs better. Similarly, as shown in table 5.2. the methods poorly performed on the second data set with the rates also being very similar and close to 0.50, which is not a high performing number.

	Accuracy	**Precision**	**Recall**	**F-measure**
Decision Tree Gini- Depth 5	0.508	0.440	0.183	0.259
Decision Tree Entropy- Depth 5	0.513	0.433	0.491	0.460
Random Forest	0.540	0.415	0.340	0.374
Naïve Bayes	0.464	0.348	0.133	0.193
Neural Network	0.544	0.714	0.083	0.149

Table 5.1: Data set 1

5.2 Evaluation

Each data set used in this project is split into two sets: 70% is used as training data and the remaining 30% is testing data. Where the model is trained using the training data and then

	Accuracy	Precision	Recall	F-measure
Decision Tree Gini- Depth 5	0.520	0.468	0.558	0.509
Decision Tree Entropy- Depth 5	0.518	0.577	0.517	0.545
Random Forest	0.503	0.625	0.081	0.143
Naïve Bayes	0.549	0.435	0.192	0.267
Neural Network	0.565	0.500	0.189	0.274

Table 5.2: Data set 2

it is evaluated by passing the testing data and comparing the acquired classification with the actual target value. This evaluation method is called the Hold Out method. It is the "simplest evaluation method and is widely used in Machine Learning projects" [61]. Since the data split occurs at random, it is unknown which data ends up in the test or train set. The results in section 5.1 are based on this evaluation method.

Another method is "K-Fold Cross-Validation", where k sets of almost equal sizes are created from the whole data set. One set is selected to be the test set, and the remaining k-1 sets are the training set. This way each data entry is part of a test set exactly once, and part of the training set k-1 times. Due to the need to run the model k-times, this method is computationally expensive compared with Hold Out [61]. Running K-fold Cross validation on the decision tree with k being 2-39 did not show any significant changes on the accuracy rate. Meaning that the reason behind the low accuracy levels is not the choice of training data. Similarly, k-folds did not show any positive outcomes on any of the models in this project. These findings suggest that the reason behind the low accuracy rate is not the split between the training and testing sets.

5.2.1 Trying to Improve Accuracy

Some variables in the data set could prevent classifiers from achieving good accuracy because they are irrelevant or offer no predictive information. Therefore, choosing a subset of the variables by feature selection could improve the accuracy rates of the classification methods. It has been proved that feature selection is an efficient and effective way to prepare high-dimensional data for various data-mining and machine-learning problems [62]. In addition to building simpler and more understandable models, feature selection aims to improve data-mining performance and prepare clean, understandable data [62].

With the classes in the sklearn.feature_selection module, you can select features from sample sets or reduce their dimensionality to improve estimators' accuracy scores or to improve

their performance on very large datasets [54]. The procedure of univariate feature selection involves selecting the most relevant features based on univariate statistical tests. In univariate tests, each feature is compared with the target variable to determine if there is a statistically significant relationship between them. In the implementation of this project, 5 features that have a statistically significant relationship with the target variable are selected. The results of Feature selection with 5 features on the classification methods are shown in the tables 5.3 and 5.4 below. The five selected features for data set 1 are: 'UnnecessaryImport', 'LocalVariableNamingConventions', 'SingularField', 'LooseCoupling', and 'UnnecessaryLocalBeforeReturn'.

	Accuracy	**Precision**	**Recall**	**F-measure**
Decision Tree Gini- Depth 5	0.522	0.571	0.067	0.119
Decision Tree Entropy- Depth 5	0.529	0.667	0.138	0.229
Random Forest	0.530	0.529	0.145	0.228
Naïve Bayes	0.512	0.471	0.133	0.208
Neural Network	0.576	0.357	0.102	0.159

Table 5.3: Data set 1 with feature selection

The five selected features for data set 2 are: 'AvoidBranchingStatementAsLastInLoop', 'SwitchStmtsShouldHaveDefault', 'UnusedLocalVariable', 'UnnecessaryImport', and 'LogicInversion'.

No significant improvements were seen using feature selection.

	Accuracy	**Precision**	**Recall**	**F-measure**
Decision Tree Gini- Depth 5	0.533	0.500	0.410	0.450
Decision Tree Entropy- Depth 5	0.528	0.462	0.667	0.545
Random Forest	0.520	0.545	0.453	0.495
Naïve Bayes	0.573	0.500	0.058	0.103
Neural Network	0.483	0.400	0.333	0.364

Table 5.4: Data set 2 with feature selection

5.3 Discussion

The above results are all similar, whether it is with data set one or two, with feature selection or without, and even with tuning the hyper parameters. The consistency of the results signifies that the data might be the factor behind the low accuracy. The models derived could not accurately predict the performance of the students. Although our goal was not attained, we have gained insights regarding the compilation behaviour of the students. In this section, the

data sets will be analysed to deduce the reason behind the poor performance of the classification algorithms, and will analyse the most common errors and their effect on the student performance.

5.3.1 Data set one

Some error types are more common than others. Various error types were seen only in a very small percentage of the students. For instance, "EmptyCatchBlock" and "LogicInversion" (found in figures 5.1 and 5.2) were found in 0.2% of the students, and the remaining 98% had 0 for that feature. Meanwhile, the percentage of distribution of the values of other error types like "SingularField" and "UnnessecaryImport" are more distributed. As shown in figures 5.3 and 5.4, more students encounter these error types with different frequencies. The graphs that display the distributions of all the other features can be found in the appendix A.1.1 .

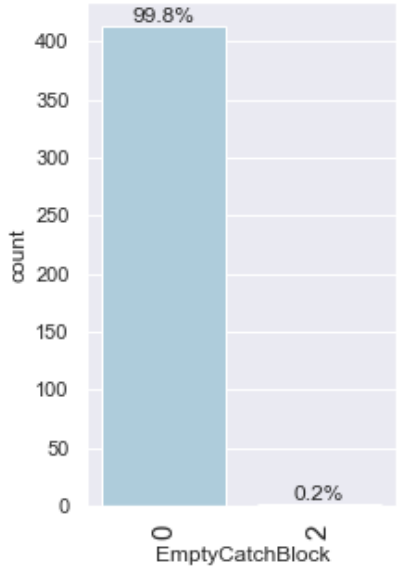

Figure 5.1: EmptyCatchBlock occurrence frequency

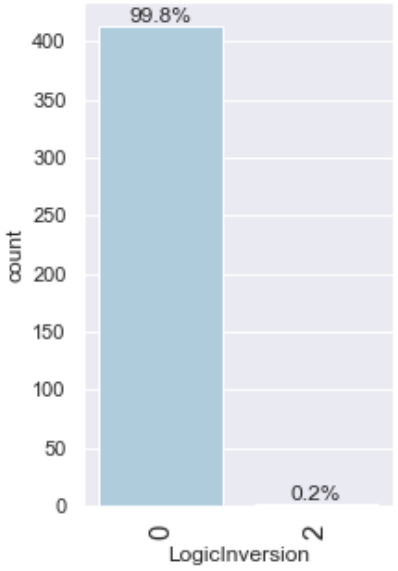

Figure 5.2: LogicInversion occurrence frequency

Most of the error types are seen in a very few number of students. Some errors are more frequent and are found in the majority of the students' submissions. Table 5.5, shows the most frequent warnings with the corresponding percentage of students that had these warnings in their submitted code.

Bi-variate analysis can help to analyse each variable with respect to the class label. The relation between each feature and the target is displayed in the figures below and appendix A.1.2.

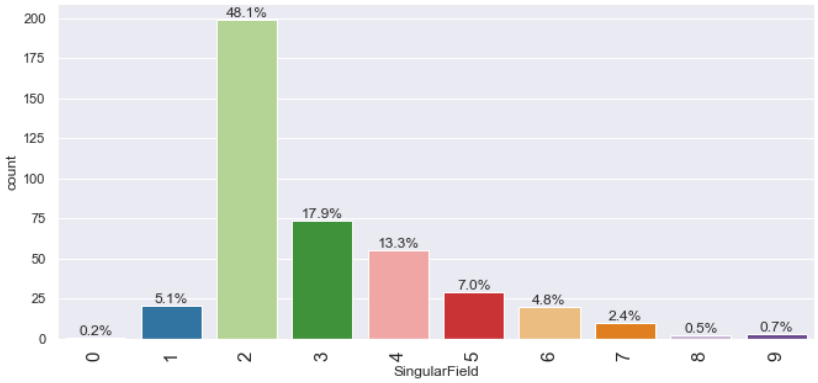

Figure 5.3: SingularField occurrence frequency

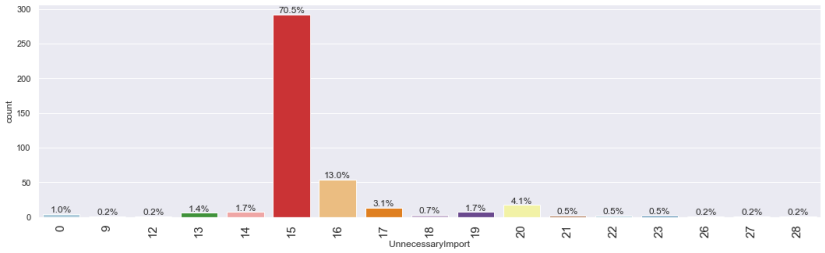

Figure 5.4: UnnessecaryImport occurrence frequency

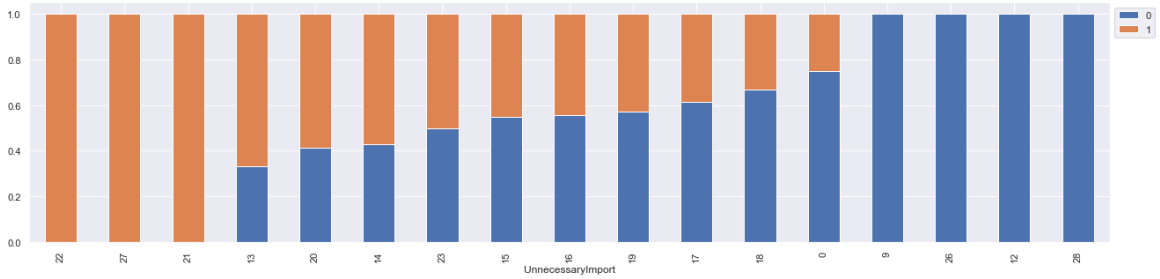

Figure 5.5: UnnecessaryImport analysis with respect to the class label

In the figure above (figure 5.5), where "UnnecessaryImport" is analysed, it can be seen that the majority of the values have both failing and passing students. There is no strict line that defines the result. The values that display a single class label, like 22, 9, and 26, are the ones with the least percentage of students in figure 5.4. Only 0.2% of the students get 26 "UnnessecaryImport" warnings. 0.2% of the students is basically one student only. Therefore, in the testing data if any new student has 26 "UnnessecaryImport" warnings, they will be immediately classified as failing. However, the data set instances of this specific feature are very little for making accurate predictions. It can be seen in table 5.6, that all the instances with a one solid class outcome have a very small number of students. On the other hand, the

Table 5.5: Most frequent errors in data set one

Warning Name	***% of Students***
UnnessecaryImport	98.5%
NoPackage	99.9%
UnusedFormalParamter	99%
LocalVariableNAming Conventions	99.9%
UnusedPRivateField	99.9%
SingularField	99.8%
Unnecessaryfully QualifiedName	97.6%

instances that have many students are divided into both passing and failing, making it hard to classify a student based on the number of instances of a warning type.

Table 5.6: 'UnnessecaryImport' Frequency Table

Grade	***0***	***1***	***All***
UnnecessaryImport			
All	224	190	414
15	160	132	292
16	30	24	54
20	7	10	17
17	8	5	13
13	2	4	6
14	3	4	7
19	4	3	7
21	0	2	2
22	0	2	2
18	2	1	3
23	1	1	2
27	0	1	1
0	3	1	4
9	1	0	1
12	1	0	1
26	1	0	1
28	1	0	1

Similarly, the Bivariate analysis of "SingularField" yields similar results. As shown in figure 5.6, and table 5.7.The values that display a single class label are 0 and 8. However, these two variables represent 3 students only. The remaining values have more students, however, they are almost equally divided between pass and fail.

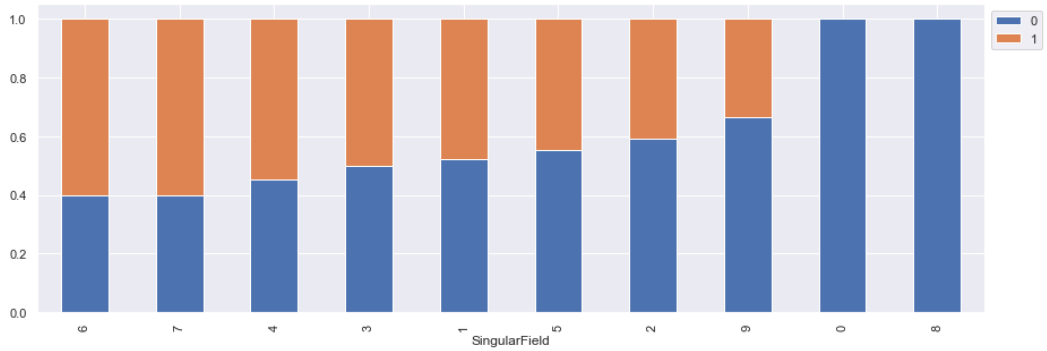

Figure 5.6: SingularField analysis with respect to the class label

Table 5.7: 'SingularField' Frequency Table

Grade	***0***	***1***	***All***
SingularField			
All	224	190	414
0	1	0	1
1	11	10	21
2	118	81	199
3	37	37	74
4	25	30	55
5	16	13	29
6	8	12	20
7	4	6	10
8	2	0	2
9	2	1	3

All thirty three variables in this data set are distributed in a similar manner, and have a similar relation to the class labels (see appendix A.1.2).

5.3.2 Data set two

The second data set has 59 warning types (features). However, there are thirteen most common ones (shown in table 5.8). A sample value distribution of a common feature - NoPAckage - is shown in figure 5.7. The remaining 46 warning types are encountered by a very little percentage of the students. "BrokenNullCheck", "ReturnEmptyArray", and "UnnecessaryModifier" are examples of warning types that are not frequent among students. It can be seen in the figures in 5.8, that more than 99% of the students do not get this error type, while the remaining minority, which is one or two students, of the students encounter them just once. The majority of the distributions of the data set features are similar to the ones presented in figure 5.8, with the majority of the student (88% +) having 0 as the number of instances (see appendix A.1.3)

.

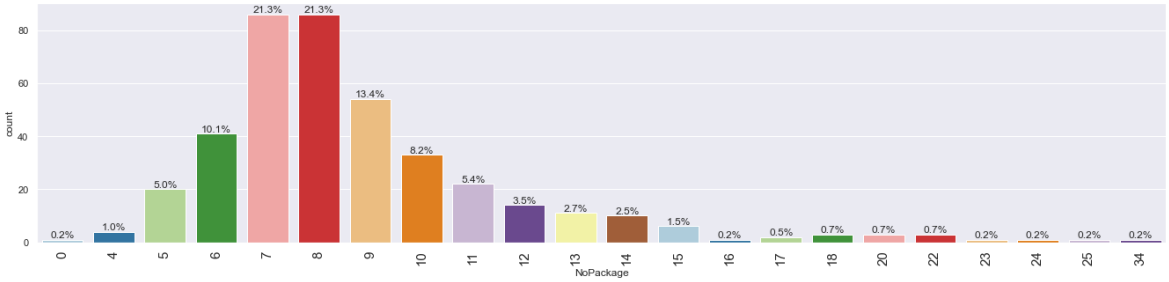

Figure 5.7: NoPackage occurrence frequency

Table 5.8: Most frequent errors in data set two

Warning Name	***% of students***
NoPackage	99.8%
UselessParantheses	98%
UnnecessaryConstructor	73.8%
ForLoopCanBeForeach	74.3%
ControlStatementBraces	74.3%
LooseCoupling	98.5%
OneDeclarationPerLine	89.4%
LiteralsFirstInComparison	87.9%
CloseResources	98%
UnusedPrivateField	56.2%
UnnecessaryImport	65.8%

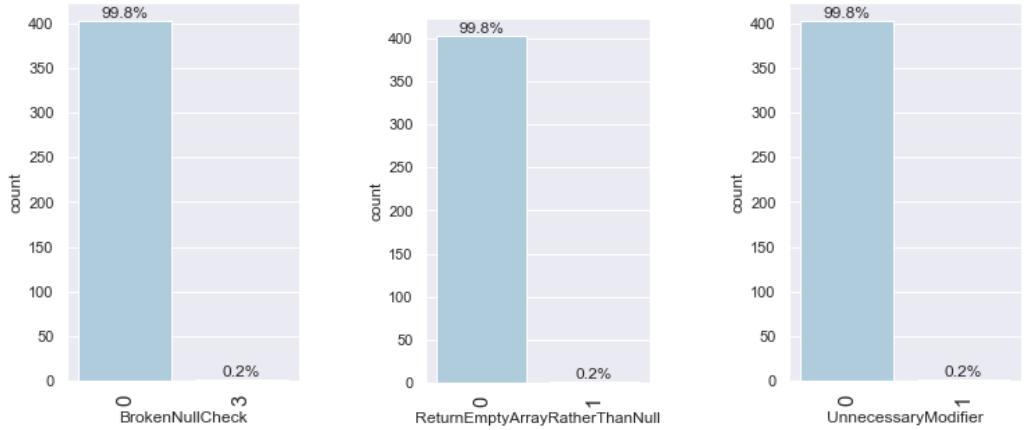

Figure 5.8: Occurrence frequencies

Bi-variate analysis is also used in this data set to help in analysing each variable with respect to the class label. The analysis of the relation between each feature and the target is similar to the analysis of data set one. It can be seen in the figures (see appendix A.1.4) that the values that seem to have only one class label are the values with the minor number of students.

figure 5.9, presents the analysis of students encountering “UselessParantheses”. The number of passing and failing students that are encountering this error are shown in table 5.9 . As seen in previous error types, this error also yields the same analysis result, where the distribution of the passing and failing students per count of an error do not help the model in predicting accurate values.

Table 5.9: 'UselessParantheses' Frequency Table

grade	***0***	***1***	***All***
UselessParentheses			
All	216	188	404
0	5	3	8
1	6	3	9
2	46	36	82
3	77	70	147
4	33	24	57
5	16	16	32
6	11	9	20
7	5	4	9
8	2	4	6
9	1	6	7
10	1	4	5
11	3	0	3
12	4	1	5
13	0	1	1
15	1	0	1
17	2	0	2
19	0	1	1
22	0	1	1
24	0	1	1
29	1	0	1
33	0	1	1
37	1	1	2
45	1	0	1
48	0	1	1
121	0	1	1

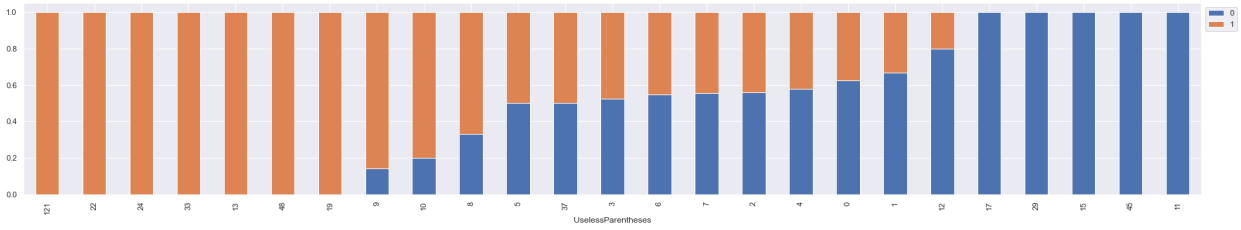

Figure 5.9: UselessParantheses analysis with respect to the class label

The provided student data is not informative enough for the models to predict students' performance. Warnings are present for students that end up passing and failing. Static analysis

tools reports outcomes, do not provide insightful details about how students will perform. The distribution of the variables and the values that these variables are taking does not seem to be consistent. Meaning that passing students do not have significant differences of the number of encountered errors compared to those students that are failing.

The data sets represent students of an introductory Java programming course. Encountering warnings about code style, design, and best practices, does not reflect the students' performance and ability to pass the course. Having different error types in a students submission, does not reflect bad performance. As the errors presented in this study do not prevent the code from executing. Students who have code with high number of warnings do not end up failing.

Chapter 6

Ethical Issues

The British Computer Society (BCS) Code of Conduct and Code of Good Practice have been closely observed during the design and implementation of the project. If a code of conduct is not adhered to, there can be serious legal and ethical consequences. It is considered unethical to use material without the owner's permission. This project involves accessing and analysing fully anonymised pre-existing data. Anonymised data can be accessed and analysed further with no ethical clearance .

All Open-Source code and libraries used in this project have been explicitly stated. The project includes my own work and Open-Source libraries. This has made it possible to speed up the development process significantly and tweak it to meet the project requirements as a result.

Using Artificial Intelligence (AI) to find students in need of additional help can raise various social/ethical issues. For example, in discussions about AI ethics in education, bias and discrimination are critical concerns. Whenever algorithms are used or created, a set of data that represents society's historical and systemic biases is generated, which ultimately translates into algorithmic bias [63]. Despite the main purpose of increased accuracy and objectivity in machine-learning models, an algorithmic bias can be incorporated into a model without explicit intent [64]. Algorithmic bias partially arises from the lack of diversity in the data sets, and the systems are not tested with different populations [65]. Bias information could include gender, race, and other forms of discriminatory assumptions. Misclassifications could occur due to bias data.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Student performance has been predicted using educational data mining and strategies to improve student performance. In most intervention studies, machine learning has been implemented, but the data being analysed is students' background information and grades. Learning software that uses software-specific data like code errors or Java standards to evaluate student performance is not precise. The aim of this study was to predict whether students are passing or failing using the static analysis of their submitted code. Four classification methods, Neural Network, Decision Tree, Random Forest, and Naïve Bayes, were implemented in an attempt of getting accurate results. Students that are bound to fail were however poorly predicted by the models. Nevertheless, insight into students' programming behaviour was gained. PMD analysis reports of student code were analysed. All error types were analysed and it was found that warnings concerning code style are the most common. The most common code style warnings among students are 'UnnecessarImport', 'NoPackage', 'UselessParantheses', and 'LocalVariableNamingConventions'.

It was found that higher frequencies of occurrences of different warning types effect the performance of students that are at risk of failing. However, they do not really effect high performing students. It was also found that the majority of the students have similar error frequencies of the same error types. Meaning that the guidance provided, prior the assignment was given, lacks some information about Java standards. Highlighting some importing coding practices that involve coding style and design could boost the overall student performance.

7.2 Future Work

Two data sets were considered in this study. Considering different data sets of different sources could yield better results. Different static analysis tools give different types of warnings; considering other tools' reports might derive better predictive models. Another possible consideration is using completely different data types. For instance, instead of using Java standards reports, compilation errors might be more informative. Temporal factors could also be highly effective. Meaning that tracking student errors from one assignment to the next one, can show how a student is progressing in his/her studies.

Another aspect to consider is using a different machine learning library, with both the existing data sets as well as new data sets. Apart from Scikit-learn, some Python libraries that are used in Machine Learning are: TensorFlow, Keras, and PyTorch.

A final enhancement that can be done, after acquiring better accuracy results, is having a user friendly interface that will allow educational staff to input student code and then get the predictions. It can also be automated; where the predictive system can be connected either to the submission portal or to the students' IDEs and it will track student coding practices and the provide them with predictions.

Bibliography

[1] P. Perera, G. Tennakoon, S. Ahangama, R. Panditharathna, and B. Chathuranga, "A systematic review of introductory programming languages for novice learners," *IEEE Access*, 2021.

[2] E. S. Tabanao, M. M. T. Rodrigo, and M. C. Jadud, "Predicting at-risk novice java programmers through the analysis of online protocols," in *Proceedings of the seventh international workshop on Computing education research*, 2011, pp. 85–92.

[3] Y. Pisan, A. Sloane, D. Richards, and R. Dale, "Providing timely feedback to large classes," in *International Conference on Computers in Education, 2002. Proceedings.* IEEE, 2002, pp. 413–414.

[4] S. Cass, "Top programming languages 2021," Oct 2021. [Online]. Available: https://spectrum.ieee.org/top-programming-languages-2021

[5] R. Asif, A. Merceron, and M. K. Pathan, "Predicting student academic performance at degree level: a case study," *International Journal of Intelligent Systems and Applications*, vol. 7, no. 1, pp. 49–61, 2014.

[6] M. M. Abdallah and M. M. Al-Rifaee, "Java standards: A comparative study," *International Journal of Computer Science and Software Engineering*, vol. 6, no. 6, p. 146, 2017.

[7] Mar 2014. [Online]. Available: https://www.havelund.com/Publications/jpl-java-standard.pdf

[8] Mar 2014. [Online]. Available: https://forge.ispras.ru/attachments/download/3200/google-styleguide-googlecode-com.pdf

[9] l. Vogel, "Java concurrency," Jul 2016. [Online]. Available: https://www.vogella.com/tutorials/JavaConcurrency/article.html

[10] R. Singh, “Java concurrency / multithreading basics,” Sep 2020. [Online]. Available: https://www.callicoder.com/java-concurrency-multithreading-basics/

[11] P. Lee, “Concurrency and parallelism in java,” Jul 2020. [Online]. Available: https://medium.com/@peterlee2068/concurrency-and-parallelism-in-java-f625bc9b0ca4

[12] J. Jenkov, “Java concurrency and multithreading tutorial,” Nov 2021. [Online]. Available: https://jenkov.com/tutorials/java-concurrency/index.html

[13] “Pmd documentation index,” Aug 2017. [Online]. Available: https://pmd.github.io/latest/index.html

[14] A. Ettles, A. Luxton-Reilly, and P. Denny, “Common logic errors made by novice programmers,” in *Proceedings of the 20th Australasian Computing Education Conference*, 2018, pp. 83–89.

[15] S. wood, “Common logic errors in java.” [Online]. Available: https://hts.stevenwood.com/ics3u/cos/units/2/activity22/logic_run_err.htm

[16] J. P. Mueller, “Logical errors in java,” Mar 2016. [Online]. Available: https://www.dummies.com/article/technology/programming-web-design/java/logical-errors-in-java-153712/

[17] J. Sonmez, “Types of duplication in code,” Jan 2018. [Online]. Available: https://simpleprogrammer.com/types-of-duplication-in-code/

[18] K. Hotta, Y. Sano, Y. Higo, and S. Kusumoto, “Is duplicate code more frequently modified than non-duplicate code in software evolution? an empirical study on open source software,” in *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, 2010, pp. 73–82.

[19] Daan, “The impact of duplicate code,” Feb 2022. [Online]. Available: https://levelup.gitconnected.com/the-impact-of-duplicate-code-31c0bceab831

[20] A.-J. Molnar, S. Motogna, and C. Vlad, “Using static analysis tools to assist student project evaluation,” in *Proceedings of the 2nd ACM SIGSOFT International Workshop on Education through Advanced Software Engineering and Artificial Intelligence*, 2020, pp. 7–12.

[21] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 672–681.

[22] S. H. Edwards, N. Kandru, and M. B. Rajagopal, "Investigating static analysis errors in student java programs," in *Proceedings of the 2017 ACM Conference on International Computing Education Research*, 2017, pp. 65–73.

[23] I. Albluwi and J. Salter, "Using static analysis tools for analyzing student behavior in an introductory programming course," *Jordanian Journal of Computers and Information Technology (JJCIT)*, vol. 6, no. 3, pp. 215–233, 2020.

[24] "Checkstyle 10.0," Feb 2022. [Online]. Available: https://checkstyle.sourceforge.io/

[25] "Review of java static analysis tools," Mar 2016. [Online]. Available: https://blog.codacy.com/review-of-java-static-analysis-tools/

[26] "Findbugs™ - find bugs in java programs." [Online]. Available: http://findbugs.sourceforge.net/

[27] S. Edwards, J. Spacco, and D. Hovemeyer, "Can industrial-strength static analysis be used to help students who are struggling to complete programming activities?" in *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 2019.

[28] D. J. Hand, "Principles of data mining," *Drug safety*, vol. 30, no. 7, pp. 621–622, 2007.

[29] J. Luan, "Data mining applications in higher education," *SPSS Executive*, vol. 7, 2004.

[30] H.-H. Hsu, *Advanced data mining technologies in bioinformatics*. IGI Global, 2006.

[31] G. Kesavaraj and S. Sukumaran, "A study on classification techniques in data mining," in *2013 fourth international conference on computing, communications and networking technologies (ICCCNT)*. IEEE, 2013, pp. 1–7.

[32] A. J. Myles, R. N. Feudale, Y. Liu, N. A. Woody, and S. D. Brown, "An introduction to decision tree modeling," *Journal of Chemometrics: A Journal of the Chemometrics Society*, vol. 18, no. 6, pp. 275–285, 2004.

[33] R. V. McCarthy, M. M. McCarthy, W. Ceccucci, and L. Halawi, "Predictive models using decision trees," in *Applying Predictive Analytics*. Springer, 2019, pp. 123–144.

[34] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[35] S. B. Kotsiantis, "Decision trees: a recent overview," *Artificial Intelligence Review*, vol. 39, no. 4, pp. 261–283, 2013.

[36] A. Chaudhary, S. Kolhe, and R. Kamal, "An improved random forest classifier for multi-class classification," *Information Processing in Agriculture*, vol. 3, no. 4, pp. 215–222, 2016.

[37] M. Alloghani, D. Al-Jumeily, T. Baker, A. Hussain, J. Mustafina, and A. J. Aljaaf, "Applications of machine learning techniques for software engineering learning and early prediction of students' performance," in *International Conference on Soft Computing in Data Science*. Springer, 2018, pp. 246–258.

[38] H. Guan, J. Li, M. Chapman, F. Deng, Z. Ji, and X. Yang, "Integration of orthoimagery and lidar data for object-based urban thematic mapping using random forests," *International journal of remote sensing*, vol. 34, no. 14, pp. 5166–5186, 2013.

[39] P. O. Gislason, J. A. Benediktsson, and J. R. Sveinsson, "Random forests for land cover classification," *Pattern recognition letters*, vol. 27, no. 4, pp. 294–300, 2006.

[40] A. Ghosh, F. E. Fassnacht, P. K. Joshi, and B. Koch, "A framework for mapping tree species combining hyperspectral and lidar data: Role of selected classifiers and sensor across three spatial scales," *International Journal of Applied Earth Observation and Geoinformation*, vol. 26, pp. 49–63, 2014.

[41] M. Belgiu and L. Drăguţ, "Random forest in remote sensing: A review of applications and future directions," *ISPRS journal of photogrammetry and remote sensing*, vol. 114, pp. 24–31, 2016.

[42] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22, 2001, pp. 41–46.

[43] Q. Ashfaq, R. Khan, and S. Farooq, "A comparative analysis of static code analysis tools that check java code adherence to java coding standards," in *2019 2nd International Conference on Communication, Computing and Digital systems (C-CODE)*. IEEE, 2019, pp. 98–103.

[44] D. S. Evale, M. F. Dumlao, S. Ambat, and M. Ballera, "Prediction model for students' performance in java programming with coursecontent recommendation system," in *Proceedings of 2016 Universal Technology Management Conference (UTMC). Minnesota, United States of America*, vol. 5, 2016.

[45] M. Day, M. R. Penumala, and J. Gonzalez-Sanchez, "Annete: an intelligent tutoring companion embedded into the eclipse ide," in *2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI)*. IEEE, 2019, pp. 71–80.

[46] T. Devasia, T. Vinushree, and V. Hegde, "Prediction of students performance using educational data mining," in *2016 International Conference on Data Mining and Advanced Computing (SAPIENCE)*. IEEE, 2016, pp. 91–95.

[47] A. Mueen, B. Zafar, and U. Manzoor, "Modeling and predicting students' academic performance using data mining techniques." *International journal of modern education & computer science*, vol. 8, no. 11, 2016.

[48] F. Okubo, T. Yamashita, A. Shimada, and H. Ogata, "A neural network approach for students' performance prediction," in *Proceedings of the seventh international learning analytics & knowledge conference*, 2017, pp. 598–599.

[49] S. Raschka and V. Mirjalili, *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing Ltd, 2019.

[50] J. Hao and T. K. Ho, "Machine learning made easy: a review of scikit-learn package in python programming language," *Journal of Educational and Behavioral Statistics*, vol. 44, no. 3, pp. 348–361, 2019.

[51] R. Nayak and T. Qiu, "A data mining application: Analysis of problems occurring during a software project development process," *International Journal of Software Engineering and Knowledge Engineering*, vol. 15, no. 04, pp. 647–663, 2005.

[52] S. M, "Pandas-categorical and continuous values encoding." May 2020. [Online]. Available: https://medium.com/aikiss/pandas-categorical-and-continuous-values-encoding-3d869fbdded0

[53] Zach, "How to perform one-hot encoding in r," Sep 2021. [Online]. Available: https://www.statology.org/one-hot-encoding-in-r/

[54] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[55] S. Okamura, "Gridsearchcv for beginners," Dec 2020. [Online]. Available: https://towardsdatascience.com/gridsearchcv-for-beginners-db48a90114ee

[56] R. Tamilselvi, B. Sivasakthi, and R. Kavitha, "An efficient preprocessing and postprocessing techniques in data mining," *International Journal of Research in Computer Applications and Robotics*, vol. 3, no. 4, pp. 80–85, 2015.

[57] Michael, "Visualizing decision trees with python (scikit-learn, graphviz, matplotlib)," Apr 2020. [Online]. Available: https://www.codementor.io/@mgalarny/visualizing-decision-trees-with-python-scikit-learn-graphviz-matplotlib-154mszcto7

[58] H. SHARMA, "Hands-on guide to graphviz python tool to define and visualize graphs," Oct 2021. [Online]. Available: https://analyticsindiamag.com/hands-on-guide-to-graphviz-python-tool-to-define-and-visualize-graphs/

[59] "Pydotplus." [Online]. Available: https://pypi.org/project/pydotplus/

[60] M. Galarnyk, "Visualizing decision trees with python (scikit-learn, graphviz, matplotlib)," Feb 2021. [Online]. Available: https://towardsdatascience.com/visualizing-decision-trees-with-python-scikit-learn-graphviz-matplotlib-1c50b4aa68dc

[61] "Cross-validation techniques in machine learning for better model," May 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/05/4-ways-to-evaluate-your-machine-learning-model-cross-validation-techniques-with-python-code/

[62] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM computing surveys (CSUR)*, vol. 50, no. 6, pp. 1–45, 2017.

[63] S. Akgun and C. Greenhow, "Artificial intelligence in education: Addressing ethical challenges in k-12 settings," *AI and Ethics*, pp. 1–10, 2021.

[64] B. C. Stahl and D. Wright, "Ethics and privacy in ai and big data: Implementing responsible research and innovation," *IEEE Security & Privacy*, vol. 16, no. 3, pp. 26–33, 2018.

[65] E. Southgate, "Artificial intelligence in schools: An ethical storm is brewing," Aug 2019. [Online]. Available: https://www.aare.edu.au/blog/?p=4325