Predicting student performance using data from an Auto-grading system

by

Huanyi Chen

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2018

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

As online auto-grading systems appear, information obtained from those systems can potentially enable the researchers to create predictive models to predict student behavior and performances. In University of Waterloo, the ECE150 (Introductory Programming) Instructional Team wants insights into how to best allocate their limited teaching resources, especially individual tutoring, to achieve improved educational outcomes. However, currently, the Instructional Team allocates tutoring time in a reactive basis. They help students "as-requested". This approach serves those students with the wherewithal to request help, but many of the students who are struggling do not reach out for assistance.

In ECE150 of year 2016, the Instructional Team had a hypothesis that the assignment grades may not be an accurate predictor of students' performance. Instead, they had another hypothesis that a behaviour analysis of student performance might be able to identify students for proactive intervention. Therefore, we, as the Research Team, want to explore what can be inferred from the students' behaviour, such as how frequently they submit, how early they submit for the first time, from the auto-grading data that can potentially allow us to identify students who need help. However, given the changing nature of the setup of auto-grading systems (for example, assignment content might be different from year to year), it is more important for us to explore the data and get insights, rather than trying to create a precise predictive model.

In this thesis, we applied Exploratory Data Analysis from NIST and conducted several experiments with decision-tree and linear-regression algorithms using information from the Marmoset auto-grading system aiming to investigate the following research themes.

1. If we put students into categories according to their final exam and midterm performances, can we create a model over the auto-grading data to understand the students' behaviour and predict those categories? More importantly, to predict the students who need help and identify them as early as possible.

2. Can we predict students' raw numerical midterm grades and raw final exam grades from the students' behaviour?

3. Can we find any interesting relations between the features generated (reflecting students' behaviour) from auto-grading system information, grades and student categories?

In our experiments, we generated different type of features based on the raw data we collected from the Marmoset of 428 first-year students in ECE150 of year 2016, such as the

passing rate for each programming task, the testcase outcomes, the number of submissions, the lab attendance and the time interval of submissions. The experiments for those features are our first step for exploring the auto-grading data. However, we mentioned more features which are reasonable for conducting experiments in the thesis and future experiments will be conducted for them. We applied a decision-tree algorithm to all above features and a linear regression algorithm to the time intervals feature to predict the students' grades on their midterm and final exam. In all experiments, we split the data into training set and testing set. The training set was balanced by applying Synthetic Minority Oversampling Technique (SMOTE).

For regression, we used the time interval between the student's first reasonable submission and the deadline as the feature, and applied linear regression algorithm to predict the exam grades. The results showed that for the midterm, the mean of difference between predicted midterm grades and actual midterm grades (maximum is 110 points) is -5.76 points and the standard deviation is 16.44 points. For the final exam, the mean of difference between predicted final exam grades and actual final exam grades (maximum is 120 points) is 0.92 points and the standard deviation is 17.12 points. In order to stabilize the residual variance, power transformation was applied.

For classification, students were divided into three categories according to their midterm and final exam grades: good-performance students, satisfactory-performance students, and poor-performance students, and we used C4.5 decision tree algorithm to classify students. In order to take the regression model into comparison, we used the predicted midterm and final exam grades to create predicted categories for regression method. The results showed that for both midterm and final exam, the regression model using the time interval between the student's first reasonable submission and the deadline gave us the best Precision and F-measure for predicting which students would perform poorly on the exams.

During the experiments, we found for predicting raw midterm grades or raw final exam grades, the time interval information from the assignment assigned right before the midterm exam or the final exam was most correlated with the midterm grades or final exam grades; however, if we considered midterm grades for the final exam, we found the correlation of the midterm grades was greater than the correlation of all assignments.

The experiment results show that the linear regression model using submission time interval performs better than other models and further researching on this might be the best next step. However, since this is only a preliminary auto-grading data exploratory study, we can only get limited insight from the data and features. Future work will include performing additional experiments on combining different features to explore the data and as we collect more data, we can reach more definitive conclusions.

Acknowledgements

I would like to thank Prof. Paul A.S Ward for his great supervision and continuous support for my Master study and research. His idea is like a train, taking me to a boundless and colorful research world.

I would like to thank Dr. Carol Hulls and Dr. Derek Rayside for giving me priceless feedback for my thesis. It is like fuel, which turns the train into an express train.

Last but not the least, I would like to thank my family, my wife, my friends and my colleges for supporting me in writing the thesis and my life.

Dedication

This thesis is dedicated to my wife, Hua.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In programming courses, if students can get feedback in a short time period after they submit their code, they might be able to improve quickly. However, it is not realistic for instructors or teaching assistants (TAs) to mark students' code repeatedly and generate feedback quickly if students can submit their code multiple times. The most important reason is students may potentially have large amount of submissions, it is a huge load and it may take several days before giving feedback. An auto-grading system is needed in this case. A good auto-grading system can not only give out the testcase outcomes, but also generate useful feedback for the students so that they can go back to the material and then improve their code according to the feedback. Also, since the system can keep running in the background, the feedback can be generated quickly whenever students submit their code (within a few minutes, typically).

Automatic grading systems for assessing student programming assignments have been designed and used for over 50 years [16] and it seems research on auto-grading systems never stops. Forsythe and Wirth [20] used two ALGOL grader programs to evaluate student ALGOL programs in 1961 for connecting introductory programming and numerical analysis courses at Stanford University. Hext and Winings [32] presented an automatic grading scheme for simple programming exercises in 1969. Matt [49] presented an automactic grading system for grading assignments in scientific computing in 1998. Douce et al. [16] reviewed a significant amount of influential automatic assessment systems and explored several directions automated assessment systems may take. Spacco et al. [65] developed Marmoset, a project snapshot and submission system, to gain insight into students' programming habits. This is also the system that we use in our study.

Marmoset allows students to submit code multiple times for an assignment to a central

server, which automatically tests it and records the testcase outcomes into a MySQL database [65]. Marmoset also collects fine-grained code snapshots as students work on assignments: each time a student saves her work, it is automatically committed to a CVS repository. However, because no one uses CVS anymore, the CVS repository was not set up when Marmoset system was set up in our university, so we have no data for that.

Because the ECE150 (Introductory Programming) Instructional Team allow students to submit their code multiple times for a given project, they can keep modifying their code until they get an acceptable score, or until the project deadline. Therefore, assignment performance or the testcase outcome may not be necessarily a valuable metric for them to judge if a student has learned the material well or not. However, their behaviour, such as how frequently they submit, how early they submit for the first time, etc, might give them valuable prediction ability.

Researchers in different area, such as social science or engineering education, might have different ways to explore the auto-grading data. However, in our study, we are focusing on applying exploratory data analysis (EDA [51]) from NIST to the MySQL database from the Marmoset auto-grading system with the goals of investigating the following research themes.

1. If we put students into categories according to their final exam and midterm performances, can we create a model over the auto-grading data to understand the students' behaviour and predict those categories? More importantly, to predict the students who need help and identify them as early as possible.

2. Can we predict students' raw numerical midterm grades and raw final exam grades from the students' behaviour?

3. Can we find any interesting relations between the features generated (reflecting students' behaviour) from auto-grading system information, grades and student categories?

1.1 Thesis Contributions

This thesis is the summary of our primary study of using the data from the auto-grading system to predict student performance. In this thesis we demonstrate the raw data we can find from the auto-grading system, the methods to turn a raw data into useful features for prediction, and the prediction results by using the C4.5 decision tree algorithm and the

linear regression algorithm. We explored 7 possible models and found the regression model using the time interval information between the student's first reasonable submission and the deadline gave us the best Precision and F-measure for predicting which students would perform poorly on the exams.

1.2 Thesis Organization

The organization of this thesis is as following: Chapter 1 introduces the automactic grading systems and raises the questions that we aim to answer in the research. Chapter 2 provides the background of the study. Chapter 3 demonstrates the modeling techniques that we used in the study. Chapter 4 gives the related work of the study. Chapter 5 demonstrates what the data looks like. Chapter 6 demonstrates the experiments and the evaluation of them. Chapter 7 describes the conclusion and future work of the study.

Chapter 2

Background

This chapter gives the background of our study, giving out the student-performance definition, the student categories we use, why early prediction is important and the usage of auto-grading system in programming courses.

2.1 Student-Performance Definition

Student performance has many definitions; however, graduation from the institution is the most used criteria to measure student success in most prior studies that do data mining over student information [63]. In general, most courses in University of Waterloo use the final course grades to represent students performance.

In the University of Waterloo, student performance for one course is usually calculated based on a mixture of assignment marks, project marks, raw midterm, and raw final exam grades. In terms of the Introductory Programming course in Electrical and Computer Engineering, ECE150, the raw midterm grades and raw final exam grades contribute from 60% to 90% to the final course grades. In other words, student performance is highly based on the raw midterm grades and raw final exam grades.

2.2 Student Categories

We categorize students into three: good-performance students, satisfactory-performance students and poor-performance students.

We categorize students according to their raw midterm grades and raw final grades. If grade is below T_1, we deem those students as poor performance students for that exam. If grade is above T_2, we deem them in as good performance students. Otherwise, if grade is between T_1 and T_2, we deem them as satisfactory-performance students.

2.3 Early Prediction

From the instructors' perspectives, one important thing instructors want to learn is how to predict student performance as early as possible. In this case, if a student perform poorly before the midterm exam or final exam, the instructor can help the student by arranging TAs to help them or by other methods. Therefore, we might want to predict the student categories or their midterm and final exam grades using the data from early stages of the term.

2.4 Auto-Grading System in Programming Courses

Programming courses involves skill acquisition rather than knowledge acquisition. A student cannot write bug-free, easy-to-extend code even if they have a very good understanding of the programming language but without much experience in using it.

Programming is not a one-time construction. No one can guarantee that their code can always work perfectly without causing any issue. Therefore, for any programming courses, students are supposed not only to learn the basic programming skill, but also the skill to design, test and debug their code. Students can learn those skills only by practice.

In auto-grading systems such as Marmoset, we cannot make an accurate prediction of the students' midterm and final performances purely based on the best testcase outcomes because students can modify their code and re-submit to test against the testcases multiple times. When we look at the testcase outcomes, we find that most of the students can get almost full marks for the testcases for a given task.

However, there is actually large amount of data of students that we can look into to see if it is useful in predicting student performance since the auto-grading system stores the time information of each of the students' submissions, the submitted code, the testcase outcomes of each submission, and the feedback of each submission, etc.

2.5 Marmoset

Marmoset [64] is an auto-grading system for marking programming project submissions. It was built at, and used for, computer science courses in the University of Maryland and was integrated as the auto-grading tool for ECE150 (Fundamentals of Programming) and ECE356 (Database systems) in University of Waterloo in 2015.

The architecture of it contains three major components: A J2EE webserver, an SQL database, and one or more buildservers. The webserver provides a website as the interface for students to submit their code. The SQL database stores all the Marmoset registration information of the students, the project information, and the code submissions. The buildservers are software servers that compile and run the student code. Typically, there are many build servers per physical server.

To use Marmoset the instructor needs to set up project startup files first, which is the canonical solution to the assignment. After that the instructor needs to set up the testcases. The testcases are the criteria to judge if the students' code is good or if it contains errors in it. It can be thought of as a fine-grain marking scheme. There are three types of testcases: public testcases, release testcases, and secret testcases [65]. Public testcases are those testcases the results of which for submitted projects are always visible to students. Release testcases are the testcases whose results are limited to students. The outcomes are only displayed if the students have passed all public testcases. A student will consume one release token whenever the student try to test their code against release testcases. For each project, students can only have a certain amount of release token in every 12 hours. The tokens get regenerated every 12 hours. The test results of release testcases will display the number of release testcases passed and failed, but only the feedback of the first two (These are dependent parameters and subject to change) testcases are displayed. The name of a testcases can be descriptive enough to give students some information about where they should improve their code. For secret testcases, students will not get any feedback; they cannot even know how many secret testcases there are until after the project deadline.

When the student submits his/her code, the submit server assigns it to one of the buildservers. Each of the buildservers will handle the assigned code in a queue, test it with the testcases provided by the instructor, and give the test results or the error message as the feedback back to the submit server. The submit server will then show that feedback on the website. Students can learn how to improve their code and also see if they have passed all the visible testcases. Marmoset changes the pressure from marking the students' code by hand to making comprehensive testcases.

When giving out an assignment, it usually comes with multiple tasks. Each task

contains several testcases. The naming convention for year 2016 of ECE150 is "course number"-"assignment number"-"task name" for grading tasks. Figure 2.1 shows the graded tasks for assignment 1 of ECE150. Notice that they have same deadlines. Figure 2.2 shows us the student view of test results in the Marmoset. For year 2016, the Instructional Team only used public testcases (shown in the right part of the figure) but no release testcases nor secret testcases. Green color means a submission passed a given testcase while red color means it failed that testcase. There is also grey color meaning the submission failed to compile. Marks for testcases can be different. Each testcase may not always worth one mark. Therefore, the top submission in the figure 2.2 gets a result as "6/0/0/0" because testcases worth different marks. Figure 2.3 shows us the student view of detailed test results and feedback for the second submission in figure 2.2. Students can pick any task to start and they can switch between tasks even if they have not pass all testcases.

Figure 2.1: Marmoset graded tasks

150-1-netPriceCalc	view	submit	26 Sep, 11:59 PM	Assignment 1, Task 5
150-1-bitValue	view	submit	26 Sep, 11:59 PM	Assignment 1, Task 4
150-1-floatPtNum	view	submit	26 Sep, 11:59 PM	Assignment 1, Task 3
150-1-basicAdd	view	submit	26 Sep, 11:59 PM	Assignment 1, Task 2
150-1-ASCII	view	submit	26 Sep, 11:59 PM	Assignment 1, Task 1

Figure 2.2: Student view of a task in Marmoset

Project 150-0-basicgraphics: Assignment 0, Graded Task 2

Deadline: Mon, 19 Sep at 11:59 PM (Late: Tue, 20 Sep at 01:59 AM)

Current Extension: 0

language: c

Submissions

#	date submitted	Results	release tested	# inconsistent background retests	view source	Download	Public			
							0	1	2	3
2	16 Sep 2016 11:42 AM	6 / 0 / 0 / 0			view	download				
1	16 Sep 2016 11:39 AM	5 / 0 / 0 / 0			view	download				

Figure 2.3: Test results of a task in Marmoset

Project 150-0-basicgraphics: Assignment 0, Graded Task 2

Deadline: Mon, 19 Sep at 11:59 PM (Late: Tue, 20 Sep at 01:59 AM)

Source | Download | Student view | Grant extension | Retest | mark broken

Test Results

Note: For test outcomes, *failed* means *wrong* and *error* means *crashed*.

type	test #	outcome	source coverage	points	name	short result	long result
public	0	failed	source	1	TopBottomWall.py	Test TopBottomWall.py failed	`failed`
public	1	passed	source	2	RowsWithSnakeTail.py	Test RowsWithSnakeTail.py passed	passed
public	2	passed	source	2	RowWithSnakeHead.py	Test RowWithSnakeHead.py passed	passed
public	3	passed	source	1	OtherSideWalls.py	Test OtherSideWalls.py passed	passed

5/6 points for public test cases.

0/0 points for release test cases.

0/0 points for secret test cases.

See other test results for this submission

Chapter 3

Modeling Techniques

This chapter describes the introduction of exploratory data analysis and the modeling techniques used in predicting student performance. We selected decision tree and linear regression in our experiments as our first step. However, future work will try to involve other techniques.

3.1 Exploratory Data Analysis

In Exploratory Data Analysis (EDA) [51] handbook from NIST, the EDA is said as:

> EDA is an approach/philosophy for data analysis that employs a variety of techniques (mostly graphical) to maximize insight into a data set; uncover underlying structure; extract important variables; detect outliers and anomalies; test underlying assumptions; develop parsimonious models; and determine optimal factor settings.
>
> Similar to classical data analysis approach, they both start with a general science/engineering problem and all yield science/engineering conclusions. However, the major difference is the sequence and focus of the intermediate steps [51, Section 1.1.2].
>
> For classical analysis, the sequence is:
>
> $$Problem \rightarrow Data \rightarrow Model \rightarrow Analysis \rightarrow Conclusions$$

For EDA, the sequence is:

$$Problem \rightarrow Data \rightarrow Analysis \rightarrow Model \rightarrow Conclusions$$

In addition, there are more differences in terms of the following aspect.

1. models: the classical approach imposes models (both deterministic and probabilistic) on the data while EDA approach does not;
2. focus: for classical analysis, the focus is on the model–estimating parameters of the model and generating predicted values from the model while for EDA, the focus is on the data–its structure, outliers, and models suggested by the data.
3. techniques: classical techniques are generally quantitative in nature while EDA techniques are generally graphical.
4. rigor: classical techniques serve as the probabilistic foundation of science and engineering; the most important characteristic of classical techniques is that they are rigorous, formal, and "objective". EDA techniques do not share in that rigor or formality.
5. data treatment: classical estimation techniques have the characteristic of taking all of the data and mapping the data into a few numbers ("estimates") while for EDA approach, on the other hand, often makes use of (and shows) all of the available data
6. assumptions: many EDA techniques make little or no assumptions–they present and show the data–all of the data–as is, with fewer encumbering assumptions while classical approach does not.

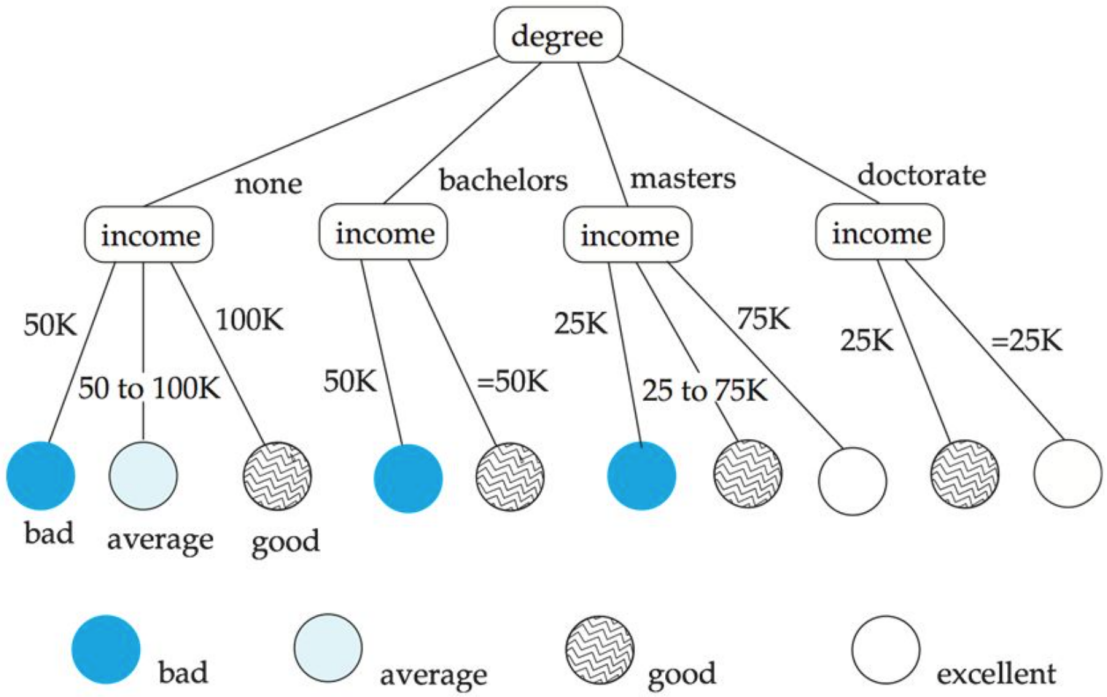
3.2 Decision Tree

Decision tree is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree. Learned trees can also be re-represented as sets of if-then rules to improve human readability. These learning methods are among the most popular of inductive inference algorithems and have been successfully applied to a broad range of tasks from learning to diagnose medical cases to learning to assess credit risk of loan applicants.

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. Each node in the tree specifies a test

of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node. An example of this is shown in Figure 3.1.

Figure 3.1: Decision tree example [44]



For example, if the degree level of a person is masters, and the person's income is 40K, starting from the root we follow the edge labeled "masters" and from there the edge labeled "25K to 75K", to reach a leaf. The class at the leaf is "good", so the decision tree algorithm predicts that the credit risk of that person is good.

3.2.1 Best Splits

The question then is how to build a decision-tree classifier given a set of training instances. The most common way of doing so is to use a greedy algorithm which works recursively, starting at the root and building the tree downward. Initially there is only one node, the root, and all training instances are associated with that node.

At each node, if all, or "almost all", of the training instances associated with the node belong to the same class, then the node becomes a leaf node associated with that class. Otherwise, a partitioning attribute and partitioning conditions must be selected to create child nodes. The data associated with each child node is the set of training instances that satisfy the partitioning condition for that child node. In our example, the attribute "degree" is chosen, and four children, one for each value of degree, are created.

Intuitively, by choosing a sequence of partitioning attributes we start with the set of all training instances, which is "impure" in the sense that it contains a mixture of different types of instances and ends up with leaves which are "pure" in the sense that at each leaf all or most training instances belong to only one class.

To judge the benefit of picking a particular attribute and condition for partitioning of the data at a node, we measure the "purity" of the data at the children that would result from partitioning by that attribute. The attribute and condition that result in the maximum purity are chosen.

Gini Measure For a set S containing k classes, let the fraction of instances in class i of the instances in S is p_i, the Gini measure of purity is defined as:

$$\text{Gini}(S) = 1 - \sum_{i=1}^{k} p_i^2 \tag{3.1}$$

When all instances are in a single class, the Gini value is 0, while it reaches its maximum (of $1 - 1/k$) if each class has the same number of instances.

Entropy Measure For a set S containing k classes, let the fraction of instances in class i of the instances in S is p_i, the Entropy measure of purity is defined as:

$$\text{Entropy}(S) = -\sum_{i=1}^{k} p_i \log_2 p_i \tag{3.2}$$

When all instances are in a single class, the entropy value is 0, and it reaches its maximum if each class has the same number of instances. The entropy measure derives from information theory.

When a set S is split into multiple sets S_i, $i = 1, 2, \ldots, r$, we can measure the purity of the resultant set of sets as:

$$\text{Purity}(S_1, S_2, \ldots, S_r) = \sum_{i=1}^{r} \frac{|S_i|}{|S|} purity(S_i) \tag{3.3}$$

That is, the purity is the weighted average of the purity of the sets S_i. The above formula can be used with both the Gini measure and the entropy measure of purity.

Information gain The information gain due to a particular split of S into $S_i, i = 1, 2, \ldots, r$ is then:

$$\text{Information\_gain}(S, S_1, S_2, \ldots, S_r) = \text{purity}(S) - \text{purity}(S_1, S_2, \ldots, S_r) \tag{3.4}$$

In this case, splits into fewer sets are preferable to splits into many sets, since they lead to simpler and more meaningful decision trees. The number of elements in each of the sets S_i may also be taken into account; otherwise, whether a set S_i has 0 element or 1 element would make a big difference in the number of sets, although the split is the same for almost all the elements. The information content of a particular split can be defined in terms of entropy as:

$$\text{Information\_content}(S_1, S_2, \ldots, S_r) = -\sum_{i=1}^{r} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|} \tag{3.5}$$

The information-gain ratio of a split is then defined as:

$$\text{Information\_gain\_ratio}(S_1, S_2, \ldots, S_r) = \frac{\text{Information\_gain}(S_1, S_2, \ldots, S_r)}{\text{Information\_content}(S_1, S_2, \ldots, S_r)} \tag{3.6}$$

Classifiers usually choose the split that gives the maximum information gain ratio. However, for binary splits, many classifiers choose the split that gives the maximum information gain, instead of the maximum information gain ratio. This is because the number

of partitions is fixed, and moreover if a split results in most values being in one partition, its information content is very low, which can result in a high information gain ratio even with a small information gain.

C4.5 Decision Tree [57] C4.5 builds decision trees from a set of training data in the same way as ID3 (Iterative Dichotomiser 3 [56]), using the concept of information entropy. It has several improvements from ID3 algorithm. In Weka [73], the class implements this is called J48. Many researchers use this technique for exploring their data [52][14][6][13] and this is the decision tree algorithm used in our experiments.

3.3 Linear Regression

Linear regression assumes that the relationship beween the dependent variable and the independent variable can be described in a linear manner. We try to find the best estimation of the value for the coefficient of the equation by using the values of the independent variable.

For example, if we have a set of value pairs as the observation $\{< x_i, y_i >: i = 0..n\}$ and we want to build a linear-regression model based on that, first, we need to consider $\hat{y}_i$ such that:

$$\hat{y}_i = \beta_0 + \beta_1 x_i \tag{3.7}$$

and

$$y_i = \hat{y}_i + \epsilon_i \tag{3.8}$$

Here, x and y are assumed to be correlated, and β_0 and β_1 are the model parameters. The ϵ_i represents the error between y_i and its theoretic value $\hat{y}_i$.

The error can be caused by measurement error, system error, etc. It can be thought of as the composite of a number of minor influences or errors. As the number of these minor influences gets larger, the distribution of the error tends to approach the normal distribution (Central Limit Theorem [46]). Therefore, the error follows:

$$\epsilon \sim N(0, \sigma^2)$$

It should be equal variance and normally distributed.

3.3.1 Multiple Linear Regression

The difference between linear regression and multiple linear regression is that in linear regression, we have only one independent variable; however, in multiple linear regression, we can have multiple independent variables. For example, a multiple linear regression model can be written as:

$$\hat{y}_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_k x_{ik} \tag{3.9}$$

and

$$y_i = \hat{y}_i + \epsilon_i \tag{3.10}$$

given the observation $\{< x_i, y_i >: i = 0..n, x_i = (x_{i1}, x_{i2}, ..., x_{ik})\}$ where k is the number of independent variables and $\beta = \beta_0, \beta_1, ..., \beta_k$ are the model parameters.

3.3.2 Estimation of the Model Parameters

A standard approach in building a linear regression model is applying the least-squares method. The goal of the least-squares method is that we want to find the β pair $\hat{\beta}$ that minimizes the sum of the squared residuals $\sum (y_i - \hat{y}_i)^2$, which means:

$$\sum \epsilon_i^2 = \sum (y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_i)^2 \tag{3.11}$$

Taking the derivatives with respect to the model's parameters β_0, ..., β_k and making them equal to zero, then our estimated parameters $\hat{\beta}_0$, $\hat{\beta}_1$, ..., $\hat{\beta}_k$ should fulfill:

$$\begin{array}{rrrcr} n\hat{\beta}_0 + & \hat{\beta}_1 \sum x_{i1} + & \hat{\beta}_2 \sum x_{i2} + & ... + & \hat{\beta}_k \sum x_{ik} = \sum y_i \\ \hat{\beta}_0 \sum x_{i1} + & \hat{\beta}_1 \sum x_{i1}^2 + & \hat{\beta}_2 \sum x_{i1}x_{i2} + & ... + & \hat{\beta}_k \sum x_{i1}x_{ik} = \sum x_{i1}y_i \\ \vdots & \vdots & \vdots & & \vdots = \vdots \\ \hat{\beta}_0 \sum x_{ik} + & \hat{\beta}_1 \sum x_{ik}x_{i1} + & \hat{\beta}_2 \sum x_{ik}x_{i2} + & ... + & \hat{\beta}_k \sum x_{ik}^2 = \sum x_{ik}y_i \end{array} \tag{3.12}$$

In Linear Algebra, we can re-write the equation to:

$$y = X\beta + \epsilon \tag{3.13}$$

where

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1k} \\ 1 & x_{21} & x_{22} & \dots & x_{2k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nk} \end{bmatrix}, \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix}, \epsilon = \begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

because

$$\sum \epsilon_i^2 = \epsilon'\epsilon = (y - X\beta)'(y - X\beta) \tag{3.14}$$

In some sense, the best β should be able to make the sum of squared residuals to be zero if there is a perfect fit (i.e., zero measurement error), which means:

$$\hat{y} = X\hat{\beta} \tag{3.15}$$

Since $y - \hat{y}$ is orthogonal to the columns of X, the best $\hat{\beta}$ should fulfill:

$$\begin{aligned} \epsilon'\epsilon =& (y - X\beta)'(y - X\beta) = X'(y - X\beta') = 0 \\ \Leftrightarrow& X'y - X'X\hat{\beta} = 0 \\ \Leftrightarrow& X'X\hat{\beta} = X'y \end{aligned}$$

so

$$\hat{\beta} = (X'X)^{-1}X'y$$

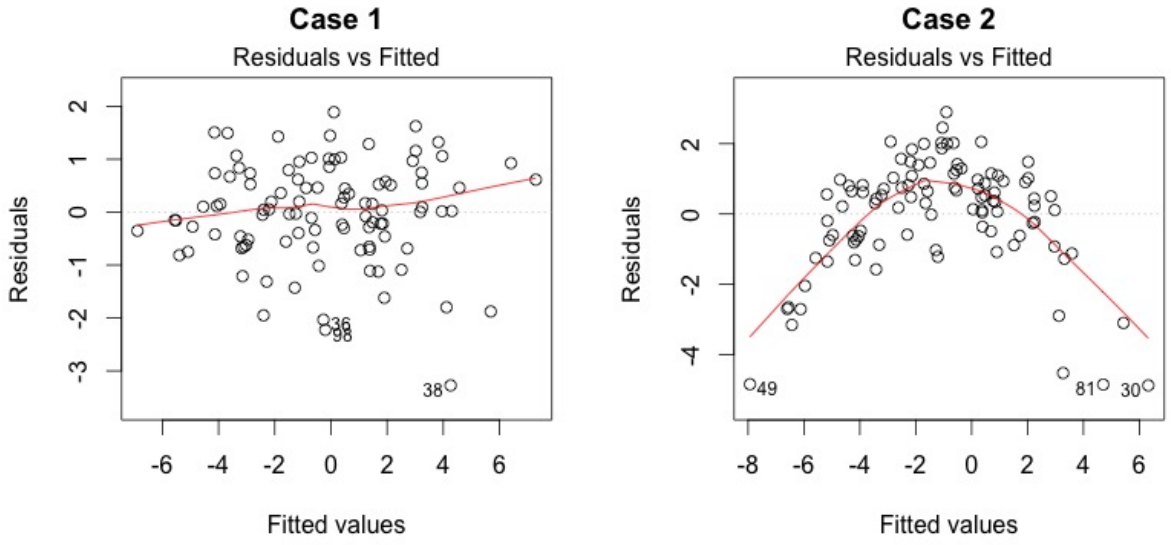
3.3.3 The Correlation Coefficient

The correlation coefficient measures the degree to which two variables are related in linear relationship when the p-value of the given model is less than 0.05. A positive relationship between two variables means if there is a positive increase in one variable, there is also a positive increase in the second variable. A value of exactly 1.0 means a perfect positive relationship. In comparison, a value of exactly -1.0 means a perfect negative relationship, which means the two variables move in opposite directions. However, if the correlation is 0, it means no relationship exists between the two variables. The value of the correlation coefficient indicates the strength of the relationship. For example, a value of 0.3 indicates a very weak positive relationship.

3.3.4 Plots

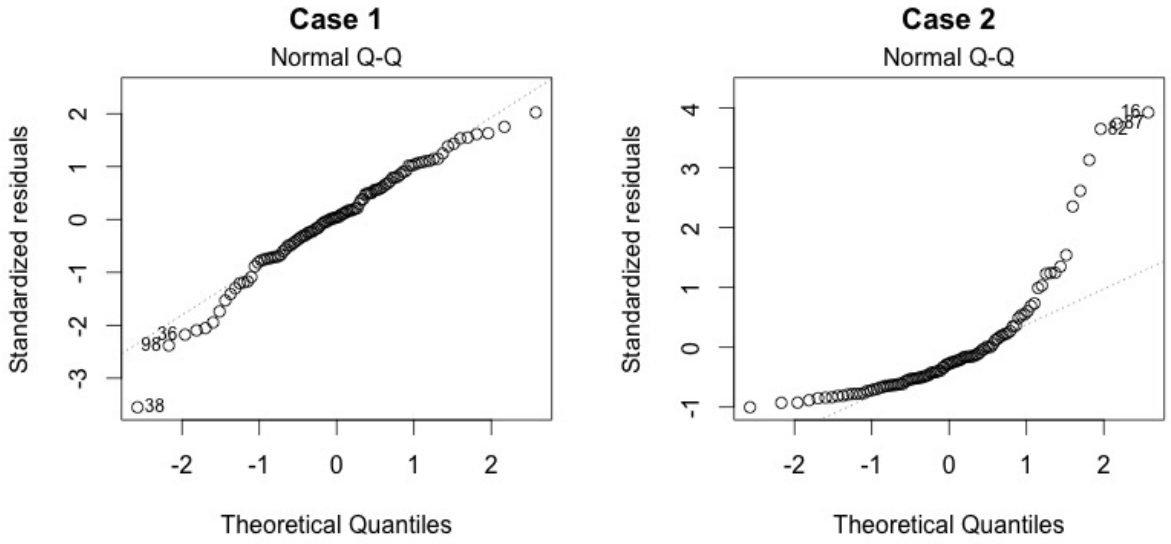
Residuals vs Fitted A residuals vs fitted plot is used for visually oberving non-linearity, unequal error variances, and outliers [53]. Equally spread residuals around a horizontal line without distinct patterns indicates there is no non-linear relationships and they have equal error variance. An example of a Residuals vs Fitted plot is shown in Figure 3.2. It is a R plot. By default, R labels the indices of three points with most extreme residuals, but it does not necessarily mean there is any outlier.

Figure 3.2: Residuals vs Fitted



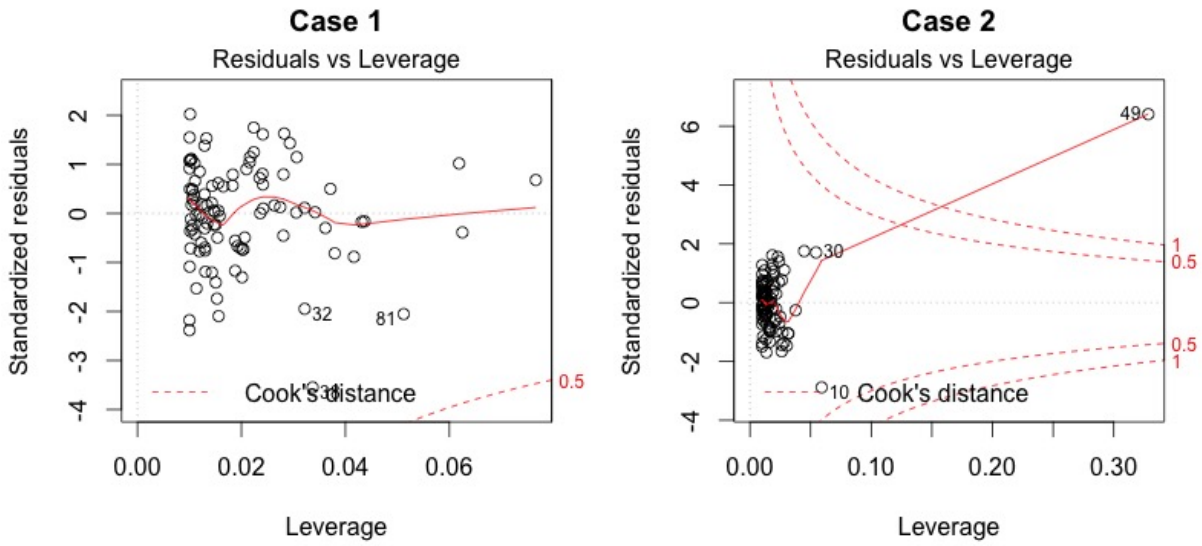
Normal Q-Q The Q-Q plot, or quantile-quantile plot, is a plot to illustrates the distribution of a dataset [8]. It is a scatterplot created by plotting two sets of quantiles against one another. If the points form a roughly straight line, it indicates both sets of quantiles are from the same distribution. A normal Q-Q plot is the plot to estimate if a dataset is from theoretical normal distribution [19]. An example of a Normal Q-Q plot is shown in Figure 3.3.

Figure 3.3: Normal Q-Q



Residuals vs Leverage A residuals vs leverage plot illustrates influential points. Not all outliers are influential in linear-regression analysis. Even though data have extreme values, they might not be influential in determining a regression line. However, some points could be very influential even if they appear to be within a reasonable range of the values. They could be extreme points against a regression line and can alter the results if we exclude them from analysis. In the plot, the points at the upper right corner or at the lower right corner needs more attention. Those points are influential against a regression line. The dashed line indicates Cook's distance (Cook's distance measures the effect of deleting a given observation). When points are outside of the Cook's distance [12] (meaning they have high Cook's distance scores), the points are influential to the regression results and we need to further look into those points to decide what action to do for them. For example, we might want to look at those cases to see if there was any measurement error or other types of errors and decide whether to removed them from the data set. An example of a Residuals vs Leverage plot is shown in Figure 3.4.

Figure 3.4: Residuals vs Leverage



3.4 Other Techniques

This section describes the techniques mentioned in related work.

M5 Model Tree [58][71] M5 builds regression tree-based models with multivariate linear models at the leaves. In Weka, the class implements this is called M5P.

Bayesian Network [36][25] A Bayesian network is a graphical model that encodes probabilistic relationships among variables of interest. In Weka, the class implements this is called BayesNet.

Naive Bayes [47] Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

CART [4] The CART builds regression trees. Different from decision tree, it outputs values rather than classes at the leaves. In Weka, the class implements this is called SimpleCart.

SimpleLogistic [45][67] It is a classifier for building linear logistic regression models. In Weka, the class implements this is called SimpleLogistic.

JRip A Weka class implements a propositional rule learner, Repeated Incremental Pruning to Produce Error Reduction (RIPPER), which was proposed as an optimized version of IREP [11].

Random Forests [3] Random forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. In Weka, the class implements this is called RandomForest.

OneR [35] OneR, short for "One Rule", is a classification algorithm that generates one rule for each predictor in the data, then selects the rule with the smallest total error as its "one rule". In Weka, the class implements this is called OneR.

NNge [48][60] A class in Weka implements:Nearest-neighbor-like algorithm using non-nested generalized exemplars (which are hyperrectangles that can be viewed as if-then rules).

Support Vector Machine (SVM) [31] A Supervised learning model with associated learning algorithms that analyze data used for classification and regression analysis. In Weka, the SMO class implements the algorithm with optimization [55][40][29].

Part [22] A Weka class builds a partial C4.5 decision tree in each iteration and makes the "best" leaf into a rule.

Instance-based learning algorithms [1] Nearest-neighbour classifier (IB1 in Weka) or K-nearest neighbours (kNN) classifier (IBk in Weka). Uses normalized Euclidean distance to find the training instance closest to the given test instance, and predicts the same class as this training instance.

RepTree [38] Fast decision tree learner. A class in Weka that builds a decision/regression tree using information gain/variance and prunes it using reduced-error pruning (with back-fitting).

Additive Regression [24] Meta classifier that enhances the performance of a regression base classifier. In Weka, the class implements this is called AdditiveRegression.

AdaBoost [23] AdaBoost, short for Adaptive Boosting, is a machine learning meta-algorithm for classifiers in conjunction with many other types of learning algorithms to improve performance. AdaBoost.R2 is AdaBoost for regression problems [17].

Ordinary Least Squares (OLS) [26] OLS is a method for estimating the unknown parameters in a linear regression model.

Neural Network [30] Artificial Neural Networks (ANNs) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains [70]. Such systems learn tasks by examples without task-specific programming.

Cross Validation [42] Cross Validation is a model validation technique for assessing the results of a given model.

Stratified Sampling In stratified sampling, the population is partitioned into regions or strata, and a sample is selected by some design within each stratum [68].

3.5 Evaluation Parameters

True Positive (TP) [28] It refer to the positive tuples that were correctly labeled by the classifier. Let TP be the number of true positives.

True Negative (TN) [28] It refer to the negative tuples that were correctly labeled by the classifier. Let TN be the number of true negatives.

False Positive (FP) [28] It refer to the negative tuples that were incorrectly labeled as positive. Let FP be the number of false positives.

False Negative (FN) [28] It refer to the positive tuples that were mislabeled as negative. Let FN be the number of false negatives.

Accuracy Calculated by $\frac{(TP+TN)}{(TP+FP+TN+FN)}$.

Precision Calculated by $\frac{TP}{(TP+FP)}$. Denoted by P.

Recall Calculated by $\frac{TP}{(TP+FN)}$. Denoted by R.

F-measure Calculated by $F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P+R}$. F1 score is calculated when β is 1.

Geometric Mean For a set of numbers $x_1, x_2, \ldots, x_n$, the geometric mean is calculated by $\sqrt[n]{x_1 x_2 \cdots x_n}$.

Mean Absolute Error (MAE) MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. For a set of actual versus predicted value pairs $(y_1, \hat{y_1}), (y_2, \hat{y_2}), \ldots, (y_n, \hat{y_n})$, the MAE is calculated by $\frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y_i}|$.

Root Mean Squared Error (RMSE) [7] Its the square root of the average of squared differences between prediction and actual observation. For a set of actual versus predicted value pairs $(y_1, \hat{y_1}), (y_2, \hat{y_2}), \ldots, (y_n, \hat{y_n})$, the RMSE is calculated by $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2}$.

Box Plots [72] A box plot is a method for graphically depicting groups of numerical data through their quartiles.

Violin Plots [33] A violin plot is a method of plotting numeric data. It is similar to box plot with a rotated kernel density plot on each side.

Chapter 4

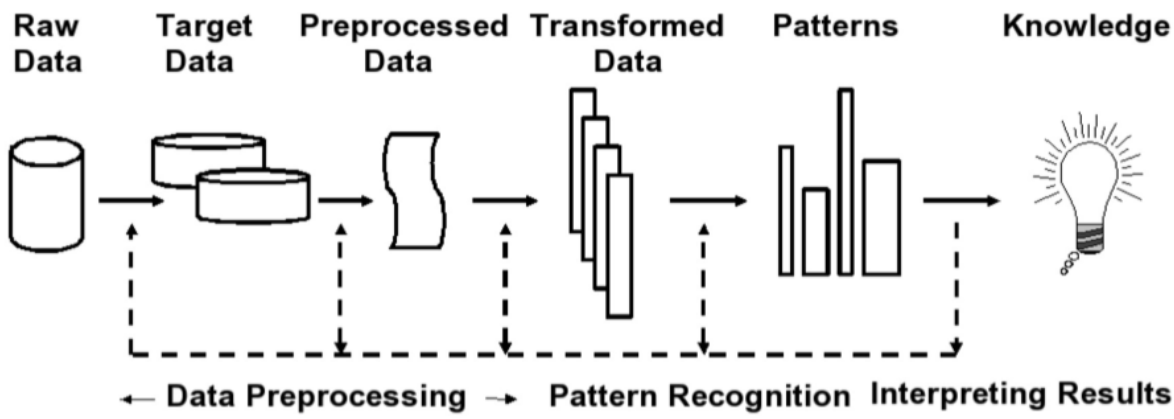
Related Work

This section describes the general data mining process and the related work in predicting student performance using data mining techniques.

4.1 Data Mining Process

The data is shown in Figure 4.1.

Figure 4.1: Data Mining Process [39, Chapter 4]



The original data we acquired from the system are often need some process to filter out the target data. In our work, there are several MySQL tables in the Marmoset containing

the startup file for students to start a project, which are not needed in our experiments. After filter out the target data, we may need to delete invalid data such as "null" fields. Afterwards, we may need to combine necessary data or do the feature extraction step to get the data to be put into classification or regression algorithms. Those steps are done during the data preprocessing. At the end, the results or the patterns need to be validate before we learn any knowledge from it.

4.2 Weka

Weka [73] is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes. Weka is open source software issued under the GNU General Public License. In the study of Nghe et al. [52], they investigate several data-mining tools and filter out Weka as their data mining tool to use. Weka is also used in many other researches [14][13][69][37], so we also choose weka as our tool in the thesis.

4.3 Prediction Target

Some researchers focus on predicting the student's overall performance for the whole semester. Nghe et al. [52] try to predict the student's actual GPA at the end of year using the student's information from the previous year. Dekker and Pechenizkiy [14] describe case studies of electrical engineering students to predict if they will drop out program after the first semester of their studies, or even before they enter the study program. Khobragade and Mahadik [41] conduct several experiments using students data to predict if a student will fail or not at the end of the semester.

Some researchers focus on predicting student performance for certain courses. For example, Bydžovská [6] present approaches to predict student success and final course grade in a particular course. Koprinska et al. [43] try to predict whether students will pass or fail their final exam. However, unlike our work, Bydžovská's work is based on study-related data such as the gender, the year of birth, the year of admission, the number of credits gained from passed courses, or the average grades and Koprinska's work is mainly based on assessment marks.

Some researchers focus on comparing the performance of different data-mining techniques. For example, Strecht et al. [66] evaluate some of the most popular classification and regression algorithms in predicting the success or failure of a student in courses. Costa et al. [13] present a comparative study on the effectiveness of educational data-mining techniques to make early predictions of students likely to fail in introductory programming courses. However, unlike our work, their work are not involving auto-grading data, which has many different aspect from their data. For example, in terms of data collection, the data from the auto-grading system is formatted and easy to collect from other types of data. Also, the auto-grading system provides a same rubric for marking students code.

4.4 Field

Programming course is more like skills acquisition. Therefore, students' code and how students code will provide much information about them.

Some researchers specifically mention that they are involving programming courses in their research. For example, Costa et al. [13] conduct their research specifically in introductory programming courses. Other researchers mention they are dealing with engineering students. However, we cannot know if programming courses are involved or not. Dekker and Pechenizkiy [14] mention they are studying the students in the Electrical Engineering (EE) department of Eindhoven University of Technology. Khobragade and Mahadik [41] use information about students enrolled on Academic Program of the Department of Computer Science and Engineering. Bydžovská [6] studied students of the Faculty of Informatics of Masaryk University.

There are some studies that do not state the department information of the students. Nghe et al. [52] mention that they collect the information of undergraduate and postgraduate students from two very different academic institutes, but none of the course information nor the department information is mentioned. Strecht et al. [66] mention they consider students from 391 programs.

4.5 Input Data

In many studies, the data from auto-grading systems is not considered by the researchers. They are more focused on using the academic data in addition of other data for prediction. For example, Dekker and Pechenizkiy [14] consider three data sets: pre-university data, a

dataset with university grades, and a dataset combining both of these. Khobragade and Mahadik [41] use data collected through surverys, college reports and departmental surveys. Strecht et al. [66] mention their data sets are extracted from the academic database of the university information system, as does the Costa et al. [13] study.

In a few studies, the researchers conduct experiments using data from auto-grading systems. However, they do not consider those data as the only source for the input data. For example, in Koprinska et al. [43] study, they have: three different sources: progression in code writing and diagnostic from an auto-grading system (PASTA), interaction and engagement from online discussion boards and wikis (Piazza), and assessment marks. However, unlike our work, their work is mainly based the assessment marks. Other data in their work only makes small improvements.

There is some studies in which the researchers consider the data from auto-grading system as the only source of input data. However, they are not trying to use it for prediction. For example, McBroom et al. [50] mine data from PASTA and cluster the students according to their submissions. They focus on analysing students' behaviours. Gramoli et al. [27] study the impact of auto-grading and instant feedback using the system called PASTA in various computer science courses. However, they are more focused on exploring the various relations rather than prediction. The only prediction they mention is that the students who start submitting their work early or finish submitting early (meaning that they finished the work) tend to get higher marks, but they also mention they have not fully answered the question.

4.6 Methodology

Nghe et al. [52] collect 20,492 complete records for students admitted from 1995 to 2002 from Can Tho University (CTU), a large national university in Viet Nam; and 936 complete records for students admitted from 2003 to 2005 from the Asian Institute of Technology (AIT), a small international postgraduate institute in Thailand that draws students from 86 different countries. Their target is to predict the student's actual GPA at the end of the 3rd year of undergraduate studies at CTU, and at the end of the 1st year of postgraduate studies at AIT. They construct and evaluate models using Decision Trees (the C4.5 Decision Tree (J48) and the M5 Model Tree) and Bayesian Network algorithms. Besides predicting continuous GPA, they also try to subdivide the range of values into new classes. They find in some cases subdividing the range of values into new classes led to significant improvements in accuracy. To compensate for the dataset's imbalance problem (some algorithms work best when the number of instances of each classes are roughly equal. When

the number of instances of one class far exceeds the other, they may not work), they use the resample function in Weka, which oversamples the minority class and undersamples the majority class to create a more balanced distribution for training the algorithms.

Dekker and Pechenizkiy [14] collect data accross 9 years, from 2000 to 2009, containing information about all students involved in the EE program. They select a target dataset of 648 students who were in their first-year and came either from a pre-university secondary education or from a polytechnical education. They want to predict if a student will drop out of the program after the first semester of study or even before he/she enters the study program. They apply several Weka classifiers which are: two decision-tree algorithms, CART (SimpleCart) and C4.5 (J48), a Bayesian classifier (BayesNet), a logistic model (SimpleLogistic), a rule-based learner (JRip), and the Random Forest (RandomForest) for prediction. The OneR classifier is considered as a baseline and as an indicator of the predictive power of particular attributes. They apply cross validation for estimating general performance. They mainly consider Accuracy and the F-measure as the two important factors for evaluating their models.

Khobragade and Mahadik [41] collect students' data through the surveys, college reports and department surveys and they want to predict if a student will fail or not at the end of the semester. They build a system to store the students' data, applying data-mining techniques, and returning the prediction result to the users. For the data mining techniques, two rule-induction algorithms (NNge, SimpleCart), two decision-tree (OneR and RandomTree) and the Naive Bayes algorithm are used. The students' passing rate (true positive), failure rate (true negative), overall accuracy rate, and geometric mean are the four factors for evaluating the model performance.

Bydžovská [6] use the data stored in the Information System of Masaryk University between the years of 2010 and 2013. They omitted freshmen students because they had no data about them in the system. The data comprised of 3,584 students and 138 courses. They want to predict if a student will succeed or fail in a particular course; predict the student's final course grade of a particular course. The data set was divided into two independent data sets. The training set consisted of the data collected between the years of 2010 and 2012 (37,005 instances) and is used for the identification of the most suitable methods with their settings. The test set consists of the data from the year 2013 (11,026 instances) and is used for validation of the methods on different data. In their study the following grade scale is used: 1 (excellent), 1.5 (very good), 2 (good), 2.5 (satisfactory), 3 (sufficient), 4 (failed or waived). The value 4 represents student failure; the others represent a full completion. They take the mean absolute error (MAE) and sensitivity (also called recall) as the two most important factor for evaluation. For student success or failure prediction, they also utilize F1 score that conveys the balance between precision and recall.

They build a classifier for each investigated course based on the training set and evaluate the results using a 10-fold cross validation. The method that achieved the best results is subsequently validated on the test set. For student success or failure prediction they apply Support Vector Machines (SVM), Random Forests, Rule-based classifier (OneR), Trees (J48), Part, IB1, and Naive Bayes (NB) algorithms. For regression models to predict the final course grade, they apply SVM Reg., Random Forest, IBk, RepTree, Linear Regression, and Additive Regression algorithms. SVM gets the best result for both. In addition, they prove that students' social behavior characteristics can improve the prediction for a quarter of the courses.

Koprinska et al. [43] collect data from three different sources. All of them not only contain student-performance information, but also student activities information. They are: data from PASTA (an automatic marking and instant feedback system) which offers information of all sequences of assessment submissions for each task and student, the tests that were passed and failed (and why), the time stamps and mark obtained; data from Piazza (a sophisticated discussion board) which offers information about the number of questions asked, answered, and viewed, and the time and content of the posts for each student; and the assessment marks and final exam marks. In their study they want to predict students who are at risk of failing from a computer-programming course and they want to investigate if introducing the data from the auto-grading system (PASTA) can improve the result. They try to predict the students whose final exam grade is below 50 and the students equal to or above 50. They use a decision-tree algorithm for prediction and they use the accuracy for evaluation. They achieve 87% accuracy by using the data from Piazza and the assessment marks. They achieve the same accuracy by only using the pre-mid semester data from the same source. They notice some interesting rules during experiments. However, they only improve their result by 1% by integrating the data from PASTA, which means the fairly high accuracy is calculated from none-auto-grading data. In their experiments, the auto-grading data makes little impact.

Strecht et al. [66] extract a total of 5,779 course data sets from 391 programs from the academic database of the university information system. They focus on data from the academic year 2012/2013. Their target is the approval/failure for classification and final course grade for regression. They train a set of models for classification and regression, respectively, for each course using different algorithms. For classification they have used k-Nearest Neighbors (kNN), Random Forest (RF), AdaBoost (AB), Classification and Regression Trees (CART), Support Vector Machines, and Naive Bayes (NB); for regression they used Ordinary Least Squares (OLS), SVM, CART, kNN, Random Forest, and AdaBoost.R2 (AB.R2). Models were evaluated using the k-fold cross-validation method with stratified sampling. For the classification method, the F1 score [10] is the evaluation fac-

tor. For regression methods, the Root Mean Squared Error (RMSE) [7] is the evaluation factor. For analysing performance, they present violin plots with the box plots which are side-by-side relating to F1 score and RMSE, respectively.

Costa et al. [13] analyze two data sources extracted from introductory programming courses performed either on-campus or distance education to make early prediction of students likely to fail in introductory programming courses. The first data source contains information about 262 undergraduate students taking the introductory programming course remotely in their university in 2013 during 10 weeks. The data source contains the following information about the students: age, gender, civil status, city, income, student registration, period, class, semester, campus, access frequency of the students in the system, participation in the discussions forum, amount of received and viewed files, use of the educational tools provided by the system as blog, glossary, quiz, wiki, message, year of enrolling in the course, status on discipline, and performance of the students in the weekly activities and exams. The second data source contains information about 161 students on-campus in their university in 2014 for 16 weeks. The data source contains the following students information: age, gender, civil status, city, income, student registration, period, class, semester, campus, year of enrolling in the course, status on discipline, amount of exercise performed by the student, number of correct exercises, and performance of the students in the weekly activities and exams. They apply the Weka tool for their experiments. They solve the high dimensionality and unbalanced data problem by using Information Gain. They apply four data mining techniques: SVM algorithm, Decision Tree via J48, Neural Network, and Naive Bayes. They use the F-measure for evaluate the effectiveness of the selected techniques. They find after the first week of the courses the data mining techniques are able to identify with at least 50% of effectiveness for the students likely to fail for both data sets and Decision Tree techniques present the highest effectiveness on both data sources.

To summary, some of them is similar to our study in that they also tried to predict the performance of students. However, most of them did not involve auto-grading data and they were not trying to predict the raw grades, which are different from our study.

Chapter 5

Raw Data and Feature Creation

This chapter describes what the raw data is in our experiments and what features we extracted from it.

5.1 Raw Data

This section describes what the raw data is. For example, the course information, the Marmoset information and the grade submission spreadsheet.

5.1.1 Course Information

This section describes the information that we already known for the course.

Student Sections: When admitting a student, the university will ask the student to fill a form indicating what programming experience they have. According to that, students will be put into 2 categories. Students are placed in Section 1 if they have programming experience, while students without are placed in other sections.

Assignment/Project Information: In the course ECE150 during year 2016, the total number of graded tasks (an assignment can contain several tasks) are 31. Among them there are 2 projects (each is a single task in Marmoset), that are specifically designed for the Section 1 students and there is 1 task, the last task, that is optional for all students.

For the other 28 tasks, the first 16 tasks are due before the midterm exam, and the other 12 tasks are provided after midterm exam and due before the final exam.

Lab Time Information: For the ECE150 programming course in Univeristy of Waterloo, it contains three parts: the lectures, the labs and tutorials. Lab sessions are scheduled for students and are assigned several Teaching Assistants (TAs) for each session so that when a student encounter any problem while doing the assignments, he or she can ask for help. Each lab session is usually 2 hours and they are scheduled weekly.

Initially stated due date/time The instructor will post the deadline on the Waterloo LEARN system [15] when giving out an assignment (If an assignment contains several tasks, they all have the same deadline). It is usually the same as the deadline in Marmoset. However, in some circumstances (i.e., when most students are not able to finish in time), the deadline may be changed in Marmoset.

Major Information: The course ECE150 contains students who major in either computer engineering or electrical engineering.

5.1.2 Information from Marmoset

This section describes the information we can directly extract from Marmoset.

Submission Time: Whenever a student makes a new submission, Marmoset will save the time information into the MySQL database.

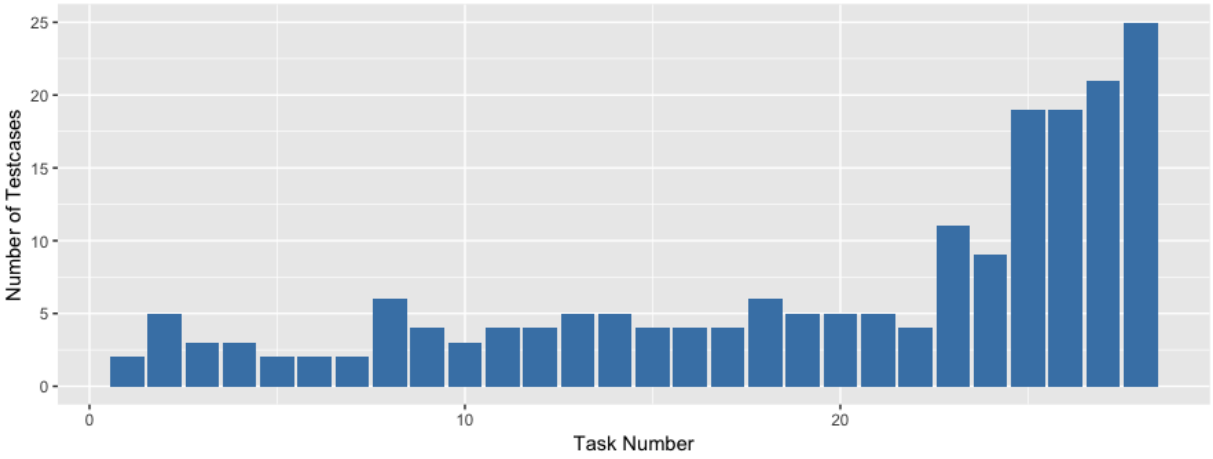
Number of Submissions: Marmoset will count the number of submissions a student made for a given task and store the information in the MySQL database.

Code: Marmoset creates medium blobs in the MySQL database to store the content of every submission of a student. All code information is stored.

Testcases: For each task, we designed a number of different testcases.

Number of Testcases: The number of testcases differ from one task to another, the distribution of testcases is shown in Figure 5.1.

Figure 5.1: Relation between task number and number of testcases



The number of testcases increases as the task number increases, which might also mean the difficulty of the task increases as to what the instructor expected.

Testcase Outcomes: There are usually several testcases for each task. Marmoset stores the testcase outcomes for each submission of each task in MySQL database.

Task Due Date: When seting up a new task in Marmoset, the instructor needs to set up the due date information for it. Those information is also stored in the MySQL database.

5.1.3 Grade Submission Spreadsheet

This section describes the prediction targets in our research.

Outcome data

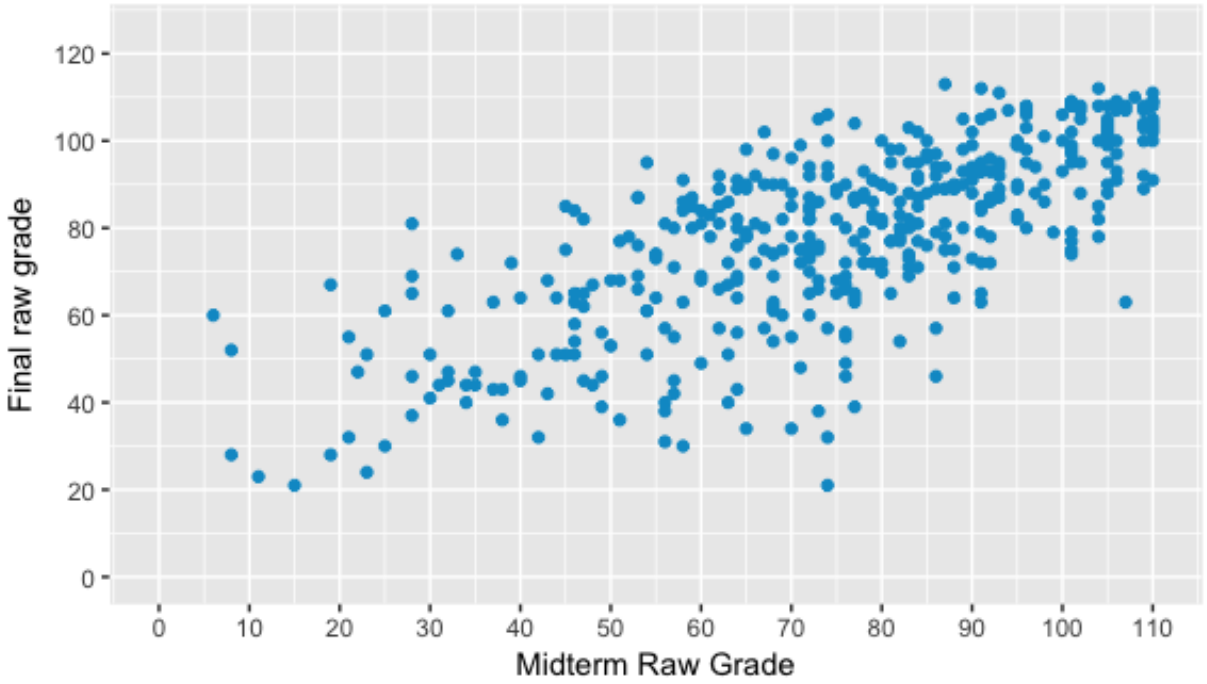
For the course ECE150, the Instructional Team have a spreadsheet that records the raw grades of all students for both the midterm exam and the final exam. This section describes the grades information and the distribution of it.

Midterm Exam: For the midterm exam, the Instructional Team record the total marks for the exam. In addition, the Instructional Team also record the marks for each subquestion.

Final Exam: Similar to the midterm exam, for final exam, the Instructional Team record the total marks for the exam and the Instructional Team also record the marks for each subquestion.

Relation between raw midterm grades and raw final exam grades: For the course ECE150 in 2016, there are bonus points for both the midterm exam and final exam. Therefore, the total mark for midterm is 110 and for final is 120. Figure 5.2 shows us the relation between raw midterm grades and raw final exam grades.

Figure 5.2: Relation between raw midterm grades and raw final exam grades



5.2 Feature Creation

This section describes the feature we think that can be generated.

5.2.1 Passing Rate for each task

This feature reflects the passing rate of the best submission of a student for a given task. However, there are certainly other features that we can further explore, such as the initial submission in terms of the number of testcases passed, the trend between submission in terms of improvement, etc. We plan to explore them in future. If a task has n testcases, and the best submission of a student passes m testcases, then the passing rate is calculated by m/n. We expect this feature to be of limited value because students can keep submitting until they pass all the testcases. We extracted this feature to validate the assumption that using the marks or the grades from the auto-grading system is not able to give us a good prediction model, which motivate us to extract other features for predicion.

Figure 5.3: The relation between passing rate and task number

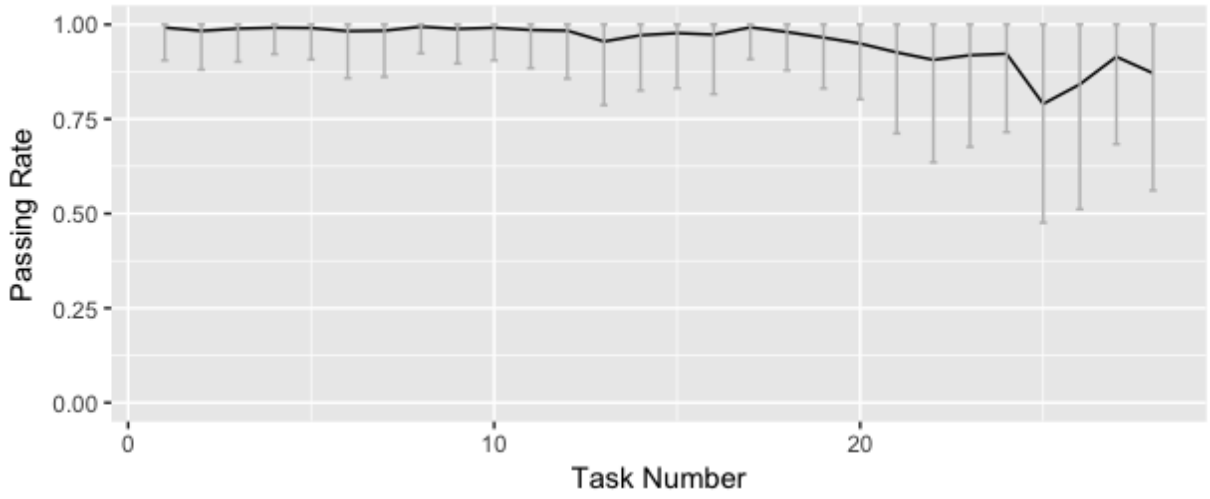


Figure 5.3 shows us the relation between passing rate and task number. The passing rates are close to 1 except towards the end of the term, when students have more work and so are less able to take advantage of unlimited submissions.

5.2.2 Testcase Outcomes

This feature is similar to the passing rate feature, reflecting the testcase outcomes of the best submission of a student for a given task. However, different from the passing rate, this feature emphasizes on individual testcase. It provides us insight of the impact of each testcase and might be able to tell us which testcases are more important.

5.2.3 Submission Time Interval

The time interval between the time of submissions and the task deadline might be considered as a useful feature. In Gramoli et al. [27] study, they observe that students who start submitting early or stop (re)submitting early to the auto-grading system tend to get higher marks in their assignments.

Figure 5.4: Time interval difference between good-performance and poor-performance students

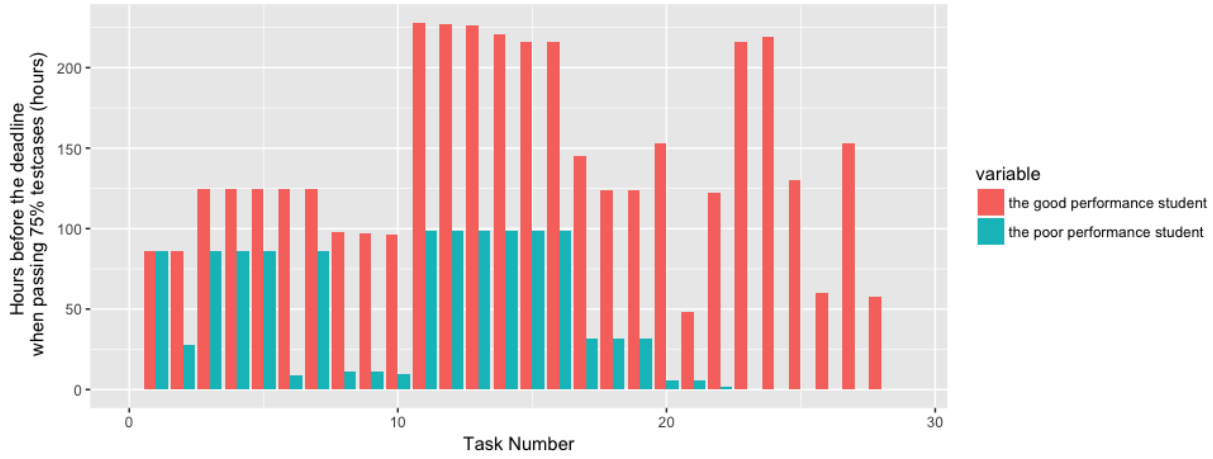


Figure 5.4 shows us an illustration of feature, the time interval difference between a randomly selected poor-performance student (32/110 for midterm and 45/120 for final) and a randomly selected good-performance student (109/110 for midterm and 103/120 for final). Here the Y axis shows us the number of hours before the deadline when the student makes a submission which passes at least 75% of the testcases for a given task. The difference is quite large.

Figure 5.5: Relation between Time intervals and Grades Example

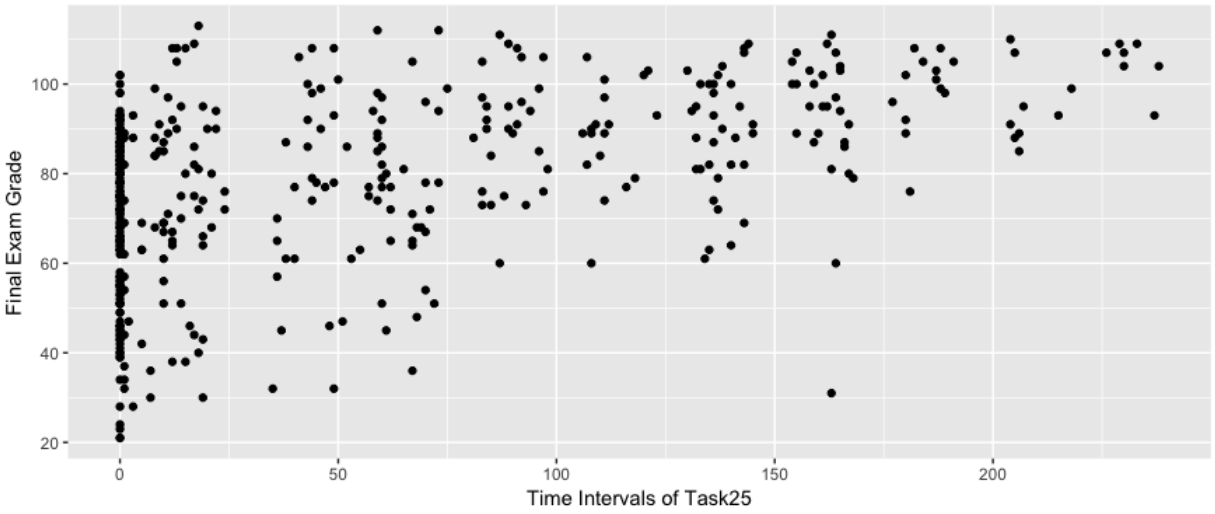


Figure 5.5 shows us an illustration of the relation between time intervals (between the submission passing 75% testcases and the deadline) of a task and raw final exam grades for all students. It shows us that a linear regression model with power transformation might be a good starting point.

Time interval between first submission and the deadline of each task: Students who start to work on tasks early might also start preparing for exams early, and so they might be able to perform well in the exams.

Time interval between last submission and the deadline of each task: The last submission usually represent the submission that gets the best score among all the submissions made to a task. The students who have large time intervals between the last submission and the deadline might mean they are able to get a high score (usually full points; the Instructional Team find it is very rare that the last submission is not the best submission in the course ECE150) in the task or project fairly early before the deadline, which might mean they are fully understanding the learning materials. Therefore, they might also be able to get a high score in exams.

Time interval between first submission and last submission of each task: The time interval between first submission and last submission reflects how much time a student

is able to spend on a certain task. That might indicate if a student has really tried to figure out the answer by himself or not. A student who is not able to think by himself, is more likely to fail exams.

Time interval between a certain submission and the deadline: Some students might try to further optimize their code even after they have already gotten full points for a task. Also, some student may submit their code to the wrong place where they have already gotten full marks. In that case, they might want to make another submission to get it right. In those cases, using the submission with best score (typically last submission) might not be able to reflect how early the time is when the students finish the task. Therefore, finding out the submission time of the first submission with best score might be more useful. In addition, there are some scenarios when that the submission with full points might not be the best indicator of the student's performance torwards a give task. For example, if a student cannot finish the task by himself, he might just copy the code from others and submit that as his work. In this case, we might need to find out that the time of the submission when he partially passed the testcases and use that for prediction. Therefore, setting up a threshold T and use the time information of the submission that a student gets $T\%$ right for prediction might be better than simply using the time information of the submission with full points.

Time interval between average submission time and the deadline: This feature contains more factors than the feature only contains the first submission or the last submission.

Average time interval between continuous submissions of each task: When we first look at the data, we find some students tend to make several early attempts towards Marmoset system, afterwards there will be a long gap until their next submissions. Therefore, we consider the average time interval between continuous submissions as the feature. It indicates the style of how a student finishes the task to some extent, which might reflect their behaviour when they prepare for the exams.

5.2.4 Number of Submissions

The number of submission feture can be directly generated from the number of submission information stored in the MySQL database of Marmoset. Although in Marmoset, the best

submission may not necessarily be the last submission, typically, the last submission is the best submission. The reason is that it is common that a student stops working on the task if he has already passed all the testcases; if he has not achieved that, he might keep working on it which means the last submission will be the best submission. Therefore, we might be able to use the number of submissions information directly to predict the students' performance. In addition, we might also want to apply some transformation, such as calculating relative number of submissions, to it for better interpretation.

Absolute number of submissions: In this case, we make no further interpretation of the value in Marmoset. We use the value directly. Figure 5.6 gives us a sense of how the median and mean value of number of submissions change according to the tasks. We can clearly see that the median and the mean number of submissions increase as the tasks number increases. This makes sense because the larger number of the task means the closer the time is towards the end of term, the more testcases, and the greater the difficulty of the task. It is reasonable that students might make more submissions.

Figure 5.6: Relation between task number and number of submissions

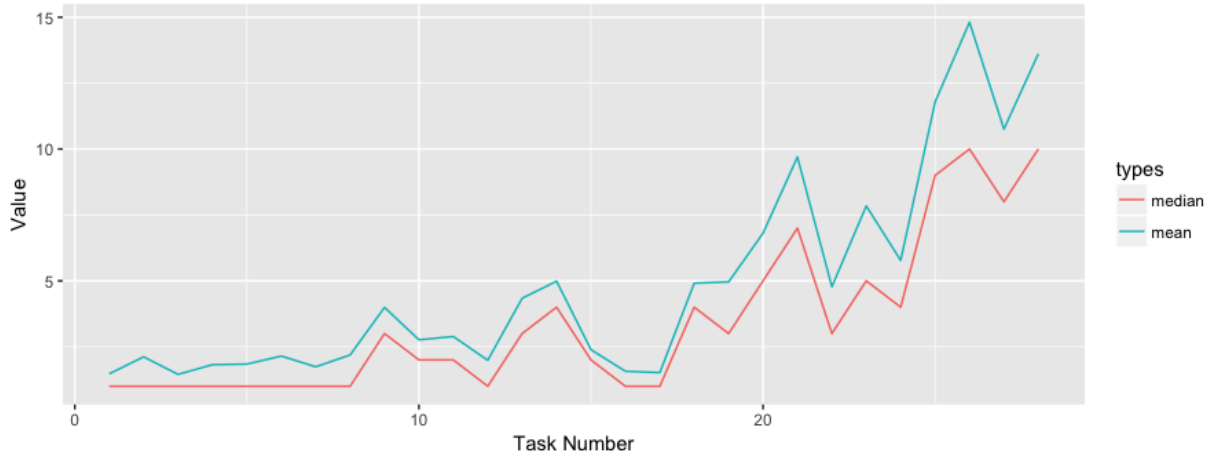
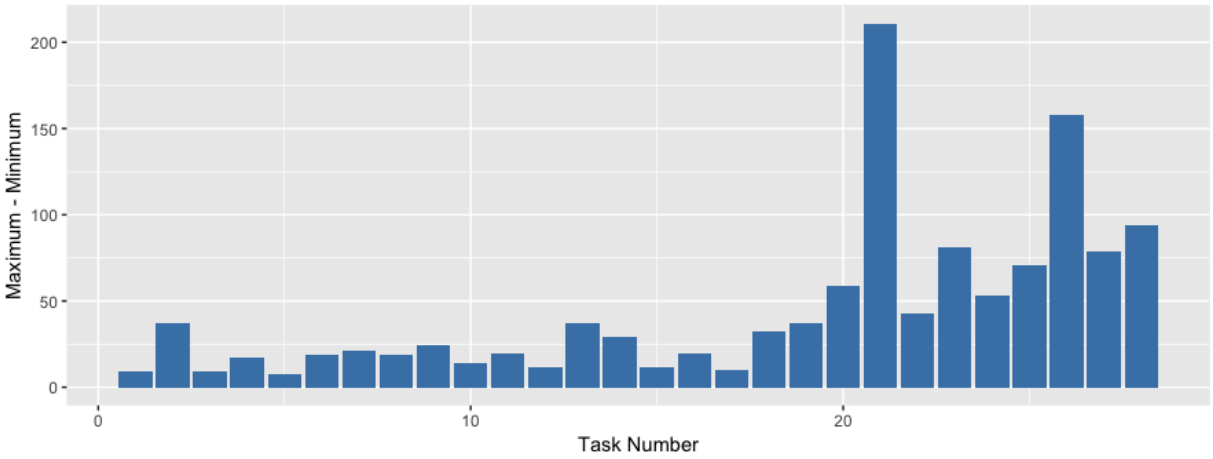


Figure 5.7 gives us the difference between the maximum and minimum number of submissions. We notice that the difference between the maximum number of submissions and minimum can be huge, such as 211 times (from the 21^{st} task). In addition, for the tasks after the midterm exam (the last 12 tasks), the difference become larger than those before the midterm exam (the first 16 tasks).

Figure 5.7: Difference between Maximum and Minimum number of submissions



This can be the first step of using the number of submissions information from Marmoset. We plan to use some other techniques such as percentage of correctness to reduce outliers in future experiments.

Relative number of submissions: Instead of using the absolute number of submissions, we might want to transform it into a relative form. For example, we might want to generate a trinary feature according to the relation between the absolute number of submissions and the median value. If the overall median number of submissions of a task is M, the absolute number of submissions of a student is x, then we might want to generate a feature value according to Table 5.1. Variables t_1 and t_2 represent different thresholds, where $t_1 < t_2$, such as t_1 as 1 and t_2 as 2.

Table 5.1: Generate feature relatively from Number of Submissions

relation	$x \leq t_1 * M$	$t_1 * M < x \leq t_2 * M$	$x > t_2 * M$
feature value	1	2	3

The reason we generate the feature in this way is that if a student makes few submissions, it might mean the performance of the student is either very good or very bad. A good-performance student achieves full mark easily with few submissions. A poor-performance student either gives up, or he starts working on the task too late and ends up with few submissions. A large number of submissions might also mean a student is either

very good or very bad. A good performance student might try several times just to pass the last testcase and achieve full marks. A poor-performance student might make many submissions but still not understand how to solve the task.

5.2.5 Lab Attendance

The lab attendance feature is the feature indicating if a student attended a given lab session or not. It is a boolean value. To generate that, we consider if a student made any submissions during the time of a lab session; if so, the value for the attendance of that lab session will be true; otherwise false. We consider this as a feature because students attending the lab session might have more opportunity to get help from the TA or the lab instructors, which might lead to a better understanding of the learning materials and a better performance.

5.2.6 Submitted Code

Potentially, the coding style of students might be able to reflect their learning style. For example, if the indentations of the C code are all messed up, it might indicate that students do not pay attention to details, and so they might be more likely to miss details when they prepare for exams. For another example, if students can further optimize their code (which might be reflected by the lines of code information) even after they gets full points for tasks, it might indicate that such students are able to think indepently. They can figure out the insuffient parts of their code even though Marmoset cannot tell them. Therefore, mining the students' code might be able to give us the information we need to predict their performances.

Lines of Code: Lines of code (LOC) information is the feature that comes first in this category. There is already some research, such as [2], that tries to use such LOC information of each save event to infer the coding style of a student. For example, if the LOC increases dramatically between previous save and current save, it is likely that the student is copying code from other places. Similarly, in Marmoset, the LOC information of each submission might also be able to provide useful information for predicting how a student will perform during exams. For example, if the LOC between the best submission and the submission before the best submission changes significantly for a student, it might mean that instead of figuring out the solution by himself/herself, the student is using other's code as reference or even he is copying code from others.

Chapter 6

Experiments and Evaluation

This chapter describes the experiments and evaluation of our study as for EDA purpose. We conducted these experiments for seeking the direction for our future studies.

6.1 Methodology

We conduct two types of experiments: classification to predict student categories; regression to predict the raw grade of the midterm exam and the final exam.

The dataset in our experiments contains 428 instances which representing the students who have both the midterm exam grade and the final exam grade in ECE 150 course for the Fall term in 2016. Students who missed either the midterm exam or the final exam are excluded.

We split the dataset into a training set and a testing set. We use the training set to build the model; then we apply the model to the testing set for evaluation. We use Weka to conduct the experitments and the default parameters settings of the algorithms in Weka are used. For the classification method, we select the C4.5 decision-tree algorithm; for the regression method, we select the multiple linear regression algorithm.

6.2 Classification for Predicting Student Categories

This section describes the regression experiments for predicting the raw grades aiming to explore the answer of the first research question.

In this section, we describe our experiments for predicting student categories using the C4.5 decision-tree classification algorithm and their results. We split the students into 3 categories according to their raw exam grades as mentioned before. We select T_1 as 80, T_2 as 50 to split the students into 3 categories: students who get higher than 80 are categorized as good-performance students for an exam; students who get lower than 50 are categorized as poor-performance students; the remaining students are categorized as satisfactory-performance students. Here, the number 80 is selected according to common sense, the number 50 is the term average for not withdrawing the engineering program.

In experiments, the training set to testing set ratio is 8:2. Instances are randomly divided. The distribution of classes is not same in training set and testing set.

However, the original training sets for midterm and final are both highly unbalanced (the majority to minority ratio are 3:1 for midterm, and 5:1 for final) because the raw grades of midterm and final exam are designed to have normal distribution most of the time. It is common that the number of students who perform poorly to be small. Therefore, we apply the Synthetic Minority Over-sampling Technique (SMOTE) [9] to oversample the poor performance students in both training sets for building better predicting models. However, since some of the data are generated, they are not real, so we may encounter some issues when evaluating models. We will mentioned it in later chapter.

After oversampling, for midterm exam, the distribution of training set is: 156 poor-performance students, 143 satisfactory-performance students and 147 good performance students. The distribution of testing set is: 10 poor-performance students, 32 satisfactory-performance students and 44 good-performance students. The information we used are based on first 16 tasks which are due before midterm exam. For final exam, the distribution of training set is: 144 poor-performance students, 124 satisfactory-performance students and 182 good performance students. The distribution of testing set is: 13 poor-performance students, 27 satisfactory-performance students and 46 good-performance students. The information we used are based on all 28 tasks.

6.2.1 Passing Rate as the Feature

For each submission, the Marmoset system tests the code against pre-set testcases, and stores the testcase outcomes. For each testcase, the test outcome will be either 'pass' or 'failed'. The passing rate for a task is calculated by the number of testcases passed versus the total number of testcases for that task. The experiment result for classifying students into 3 categories according to their midterm grades is shown as Table 6.1. The experiment

result for classifying students into 3 categories according to their final exam grades is shown as Table 6.2.

Table 6.1: Confusion Matrix for Passing Rate (midterm)

poor-performance students	satisfactory students	good-performance students	← classified as
0	0	10	poor-performance students
2	3	27	satisfactory students
0	2	42	good-performance students

Table 6.2: Confusion Matrix for Passing Rate (final)

poor-performance students	satisfactory students	good-performance students	← classified as
2	2	9	poor-performance students
3	6	18	satisfactory students
0	9	37	good-performance students

From table 6.1 and table 6.2, we can tell in both models, most of students are classified as good-performance students. In terms of the poor-performance students (which we are more care about), the midterm model makes no correct prediction (true positive) for them. Even worse, all poor-performance students are classified as good-performance students. However, it turns better when it comes to final exam, which classifies some number of poor-performance students correctly. It might indicate the passing rate for tasks after midterm exam are able to reflect some variance among students. The tasks after midterm exam might be more difficult. We will discuss more about the comparison in terms of poor-performance students in later chapter.

6.2.2 Testcase Outcomes as the Feature

Different from the passing-rate feature, the individual testcase outcome from the best submissions might be useful. A poor-performance student might satisfy with the overall passing rate for the task, but not passing all the testcases. Therefore, the individual testcases might help us to build a useful model. Specifically, certain testcase may act as differentiation between students. The experiment result for classifying students into 3

categories according to their midterm grades is shown as Table 6.3. The experiment result for classifying students into 3 categories according to their final exam grades is shown as Table 6.4.

Table 6.3: Confusion Matrix for Testcases Outcomes (midterm)

poor-performance students	satisfactory students	good-performance students	← classified as
0	1	9	poor-performance students
0	5	27	satisfactory students
0	6	40	good-performance students

Table 6.4: Confusion Matrix for Testcases Outcomes (final)

poor-performance students	satisfactory students	good-performance students	← classified as
1	4	8	poor-performance students
2	11	14	satisfactory students
1	8	37	good-performance students

From table 6.3 and table 6.4, similarly in passing rate experiments, both models classify most of students classified as good-performance students. In terms of the poor-performance students, the midterm model makes similar result as passing rate midterm model that it makes no correct prediction for poor-performance students and most of poor-performance students (9 out of 10) are classified as good-performance students. However, it also turns better when it comes to final exam, which classifies some poor-performance students correctly, which, again, might indicate the testcases designed before the midterm are not well designed while the testcases designed after midterm are able to reflect some variance among students.

6.2.3 Number of Submissions as the Feature

We test this using the absolute number of submissions and the relative number of submissions separately.

Absolute Number of Submissions as the Feature

The experiment result for classifying students into 3 categories according to midterm exam grades is shown as Table 6.5. The experiment result for classifying students into 3 categories according to their final grades is shown as Table 6.6.

Table 6.5: Confusion Matrix for Absolute Number of Submissions (midterm)

poor-performance students	satisfactory students	good-performance students	← classified as
3	3	4	poor-performance students
6	13	13	satisfactory students
6	17	21	good-performance students

Table 6.6: Confusion Matrix for Absolute Number of Submissions (final)

poor-performance students	satisfactory students	good-performance students	← classified as
7	5	1	poor-performance students
1	10	16	satisfactory students
6	14	26	good-performance students

From table 6.5 and table 6.6, it seems both models manage to make some predictions of the poor-performance students. However, for midterm exam, most of the predicted poor-performance students are actually satisfactory students and good-performance students while for final exam, most of the predicted poor-performance students are actually poor-performance students and good-performance students. It might indicate before midterm, some satisfactory students and good-performance students behave similarly as poor-performance students. However, for the whole term, most of satisfactory students will not behave as poor-performance students.

Relative Number of Submissions as the Feature

The experiment result for classifying students into 3 categories using relative number of submissions as the feature according to midterm exam grades is shown as Table 6.7. The

experiment result for classifying students into 3 categories according to their final exam grades is shown as Table 6.8.

Table 6.7: Confusion Matrix for Relative Number of Submissions (midterm)

poor-performance students	satisfactory students	good-performance students	← classified as
4	2	4	poor-performance students
6	13	13	satisfactory students
7	20	17	good-performance students

Table 6.8: Confusion Matrix for Relative Number of Submissions (final)

poor-performance students	satisfactory students	good-performance students	← classified as
2	6	5	poor-performance students
1	11	15	satisfactory students
3	15	28	good-performance students

Interestingly, from table 6.7 and table 6.8, we can observing similar results from previous experiment (using absolute number of submissions as the feature), both models also manage to make some predictions of the poor-performance students and for midterm exam, most of the predicted poor-performance students are actually satisfactory students and good-performance students while for final exam, most of the predicted poor-performance students are actually poor-performance students and good-performance students. Again, it might emphasize that before midterm exam, some satisfactory students and good-performance students behave similarly as poor-performance students. However, for the whole term, most of satisfactory students will not behave as poor-performance students.

6.2.4 Lab Attendance as the Feature

The experiment result for classifying students into 3 categories according to midterm exam grades is shown as Table 6.9. The experiment result for classifying students into 3 categories according to their final exam grades is shown as Table 6.10.

Table 6.9: Confusion Matrix for Lab Attendance (midterm)

poor-performance students	satisfactory students	good-performance students	← classified as
2	6	2	poor-performance students
4	22	6	satisfactory students
5	22	17	good-performance students

Table 6.10: Confusion Matrix for Lab Attendance (final)

poor-performance students	satisfactory students	good-performance students	← classified as
1	4	8	poor-performance students
2	11	14	satisfactory students
5	14	27	good-performance students

From table 6.9 and table 6.10, both lab attendance models make some prediction for the poor-performance students. However, most of the poor-performance students behave like satisfactory students before midterm while for final exam, most of them behave like good-performance students. It might indicate even the poor-performance students behave like students in other categories in terms of lab attendance, they still perform poorly in exams.

6.2.5 Time Interval between certain Submission and Deadline as the Feature

Time management skill is a very important factor when considering if a student can achieve high score in the midterm or final exam. From Marmoset data, the time of the students' submissions can be a good feature for us to consider their time management skill.

The feature in our experiment is the time interval between the student's submission time and the deadline of the project. Because using 'minute' or 'second' as the unit might be too fine grain, using day as the unit might be too coarse, so in our experiment we use 'hour' as the unit. The way we do the experiment is that we try to build a decision tree using the submission time to predict the students' categories. We calculate how many hours

before the deadline that the student makes the submission as the feature (a submission time after the deadline, is deemed as 0 hour before the deadline).

Instead of using the submission time of the last submission, we used the submission time of the submission when it gets T_x% correct of all the test cases because a student who have passed 99% of the testcases may keep submitting to pass the remaining testcases, which makes the time interval of those submissions not very useful.

The experiment result for classifying students into 3 categories according to midterm exam grades is shown as Table 6.11. The experiment result for classifying students into 3 categories according to their final exam grades is shown as Table 6.12.

Table 6.11: Confusion Matrix for Submission Time Interval (midterm)

poor-performance students	satisfactory students	good-performance students	← classified as
4	6	0	poor-performance students
5	16	11	satisfactory students
5	14	25	good-performance students

Table 6.12: Confusion Matrix for Submission Time Interval (final)

poor-performance students	satisfactory students	good-performance students	← classified as
5	6	2	poor-performance students
6	8	13	satisfactory students
2	15	29	good-performance students

From table 6.11 and table 6.12, both submission time interval decision tree models make some prediction for the poor-performance students. It might indicate the submission time intervals of the tasks are able to reflect some variance among students. We will discuss more about the comparison in terms of poor-performance students in later chapter.

6.3 Regression for Predicting Raw Grades

This section describes the regression experiments for predicting the raw grades aiming to explore the answer of the second research question.

6.3.1 Time Interval between certain Submission and Deadline as the Feature

The linear regression model is based on the time interval between time of the submission that the student gets 75% percent testcases passed of a given task and Marmoset deadline. The threshold 75% is picked by common sense, we compared the results of picking 70%, 75%, and 80%, they are not much different (However, 75% results look slightly better). We will do more experiments against to other thresholds in future.

Training Regression Model for Predicting Transformed Midterm Grades

Similarly to the tasks we used in classfication method when predicting midterm grades, the tasks we used for regression to predict midterm grades only includes the first 16 tasks (total 28 tasks) that are assigned before the midterm.

The initial model is;

$$grades = \beta_0 + \sum_{1}^{16} \beta_i x_i$$

However, when we validate the underlying assumptions (constant variance and normal distribution) for the residuals [51, section 1.2.1]), we found the model is far from meeting all the assumptions. Therefore, we apply spreadLevelPlot() [21][34] function (It is a function for suggesting the numeric number for power transformation of a linear regression model) to find suggested power transformation in order to stabilize the variance. The model then will be:

$$grades^{0.89} = \beta_0 + \sum_{1}^{16} \beta_i x_i$$

where 0.89 is suggested power transformation from spreadLevelPlot function.

Table 6.13: Summary of the regression model for predicting transformed midterm grades

Task Number	Estimate Coefficients	Task Number	Estimate Coefficients
(Intercept)	18.799832	9	-0.021813
1	0.007167	10	0.084012
2	0.041940	11	0.009478
3	0.052417	12	-0.012414
4	0.062626	13	0.037278
5	-0.039910	14	0.028053
6	0.053718	15	0.035604
7	-0.071680	16	-0.014787
8	-0.005584		

Table 6.13 shows us the intercept and the estimate coefficients of tasks. If we deem the feature value as a row vector F, and the estimate coefficients as a column vector EC, then the model is generated as

$$predicted = intercept + F * EC$$

The p-value for the model is $< 2.2e-16$, which is much less than 0.05 (H_0 hypothesis: no relationship between the time intervals and the final exam grades), which indicates there is a relationship between the time intervals and the midterm grades.

Figure 6.1: Plots of regression model for predicting transformed midterm grades (1/3)

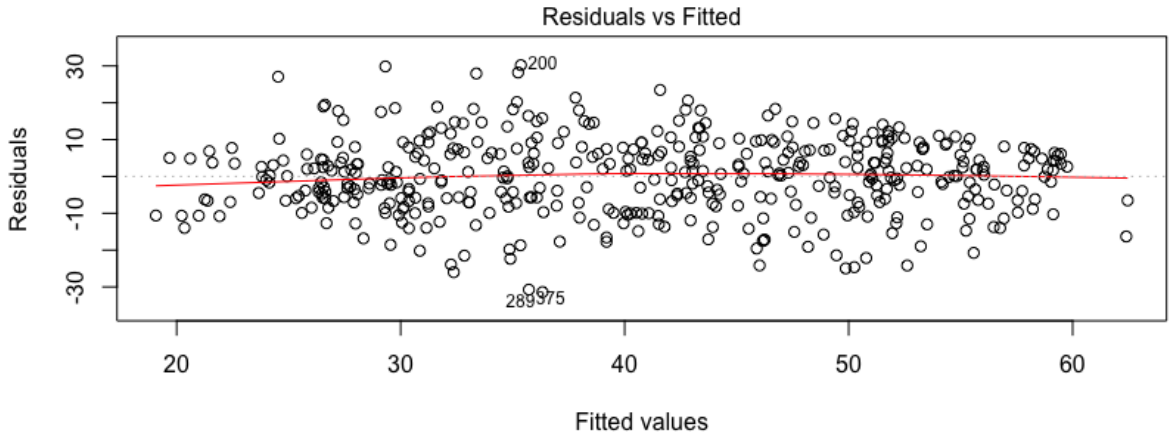


Figure 6.1 shows us the variance of the residuals is random and not a non-linear relation. By default, R labels the indices of three points with most extreme residuals (such as 200

in the graph), but it does not necessarily mean there is any outlier. In addition to using graphic techniques, we also applied Non-Constant Error Variance (ncvTest()) function in R to test if it is constant error variance (H_0 hypothesis). The test accepted our model ($p > 0.05$). The result is $p = 0.44$.

Figure 6.2: Plots of regression model for predicting transformed midterm grades (2/3)

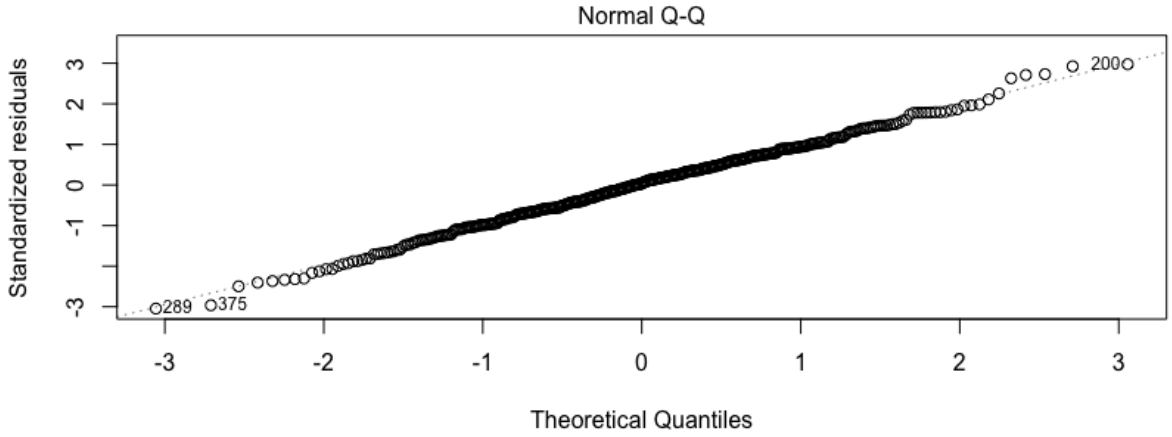


Figure 6.2 shows us the distribution of the residuals is a normal distribution. In addition to, we also applied pearson.test() function (Pearson Chi-Square test [54]) in R to test if the residuals is normally distributed (H_0 hypothesis). The test accepted our model ($p > 0.05$). The result is $p = 0.57$

Figure 6.3: Plots of regression model for predicting transformed midterm grades (3/3)

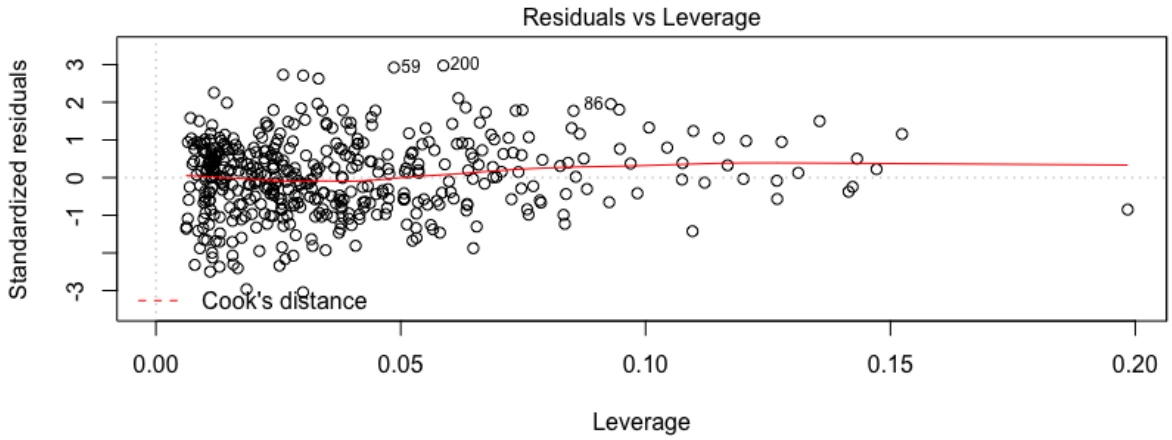


Figure 6.3 shows us there is no influential points in the dataset otherwise there will be some dotted lines to cut out the influential points as shown in the right picture of figure 3.4.

Evaluate on training dataset

Table 6.14: Evaluation of regression model on training dataset for predicting transformed midterm grades

Correlation coefficient	0.72
p-value	$1.80e-58$
Relative absolute error	64.50 %
Root relative squared error	69.25 %
Total Number of Instances	446

Table 6.14 shows us the evaluation of the linear regression model for predicting the transformed midterm grades on training dataset.

Evaluate on testing dataset for predicting transformed midterm grades

Table 6.15: Evaluation of regression model on testing dataset for predicting midterm grades

Correlation coefficient	0.62
Relative absolute error	87.97 %
Root relative squared error	85.36 %
Total Number of Instances	86

Table 6.15 shows us the evaluation of the linear regression model on the testing dataset for predicting the transformed midterm grades.

Figure 6.4: Histogram of Regression Difference between Predicted and Actual Grades

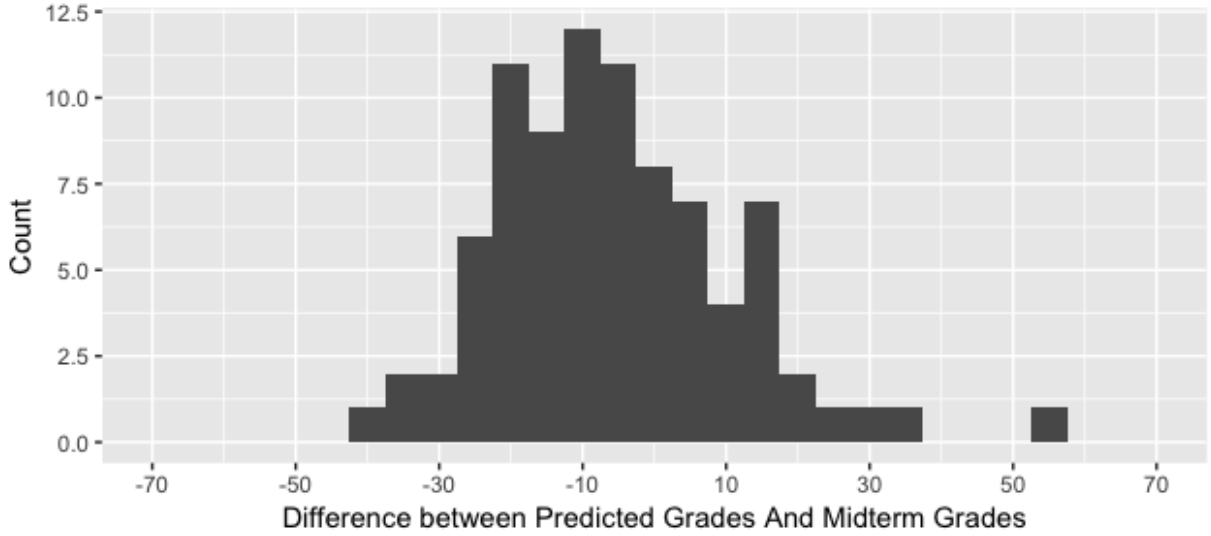


Figure 6.4 shows us the histogram of the difference of the predicted grades and actual grades of testing set. The predicted midterm grades are calculated by applying reversal transformation. the mean of difference between predicted grades and actual midterm grades (maximum is 110 points) is -5.76 points and the standard deviation is 16.44 points.

Training Regression Model for Predicting Transformed Final Exam Grades

The tasks we used for this experiment include all 28 tasks. The midterm grades are not included.

The initial model is:

$$grades = \beta_0 + \sum_{1}^{28} \beta_i x_i$$

However, when we validate the underlying assumptions for the residuals (constant variance and normal distribution), we found the model is not meeting all the requirements. Therefore, similarly to transforming midterm grades, we also apply spreadLevelPlot() function to find suggested power transformation in order to stabilize the variance. The model then will be:

$$grades^{1.32} = \beta_0 + \sum_{1}^{28} \beta_i x_i$$

where 1.32 is suggested power transformation from spreadLevelPlot function.

Table 6.16: Summary of the regression model for predicting transformed final exam grades

Task Number	Estimate Coefficients	Task Number	Estimate Coefficients
(Intercept)	122.434477	15	0.294312
1	0.053899	16	-0.115364
2	0.111702	17	-0.129410
3	0.101179	18	0.012858
4	-0.042122	19	0.376133
5	0.309120	20	0.031075
6	0.211473	21	0.152144
7	-0.296870	22	-0.071589
8	0.085323	23	0.124687
9	-0.004359	24	0.013749
10	0.077508	25	-0.003543
11	-0.111243	26	-0.123731
12	0.062137	27	0.622770
13	0.335927	28	0.501828
14	-0.125272		

Table 6.16 shows us the intercept and the estimate coefficients of tasks. As with the training for the midterm grade model, if we deem the feature value as a row vector F, and

the estimate coefficients as a column vector EC, then the model is generated as

$$predicted = intercept + F * EC$$

The p-value for the model is $< 2.2e-16$, which is much less than 0.05 (H_0 hypothesis: no relationship between the time intervals and the final exam grades), which indicates there is a relationship between the time intervals and the final exam grades.

Figure 6.5: Plots of regression model for predicting transformed final exam grades (1/3)

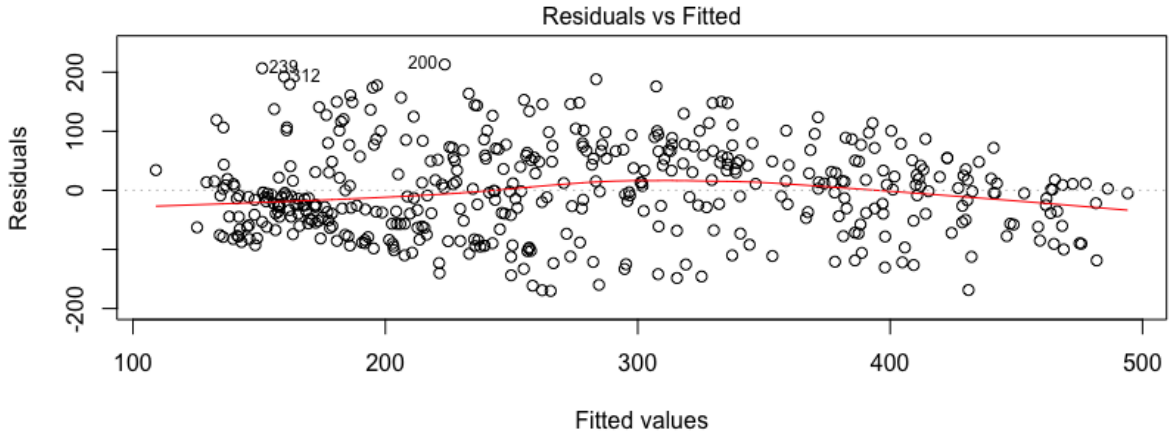


Figure 6.5 shows us the variance of the residuals is random and not a non-linear relation. In addition to using graphic techniques, we also applied ncvTest() function in R to test if it is constant error variance (H_0 hypothesis). The test accepted our model ($p > 0.05$). The result is $p = 0.19$.

Figure 6.6 shows us the distribution of the residuals is a normal distribution. In addition to, we also applied pearson.test() function in R to test if the residuals is normally distributed (H_0 hypothesis). However, the test did not accept our model ($p > 0.05$). The result is $p = 0.02$. This also can be observed from the plot, the left tail is slightly skewed. However, we applied the same procedure to build a model without balancing the training set, the Pearson Chi-Square test accepted that model with a 0.66 p-value. A possible reason is that the way we balance the training set is not very appropriate. Further research needs to be done in selecting the best balancing algorithm.

Figure 6.7 shows us there is no influential points in the dataset otherwise there will be some dotted lines to cut out the influential points as shown in the right picture of figure 3.4.

Figure 6.6: Plots of regression model for predicting transformed final exam grades (2/3)

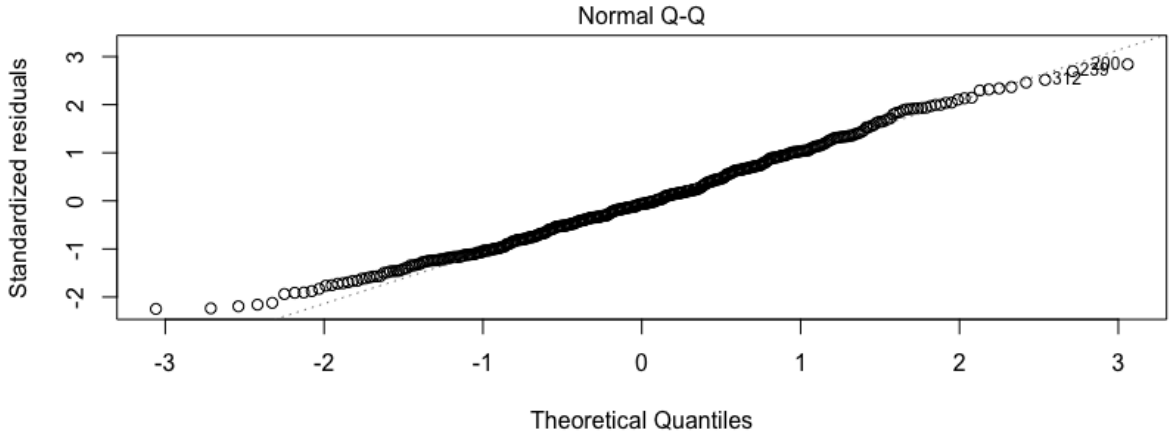
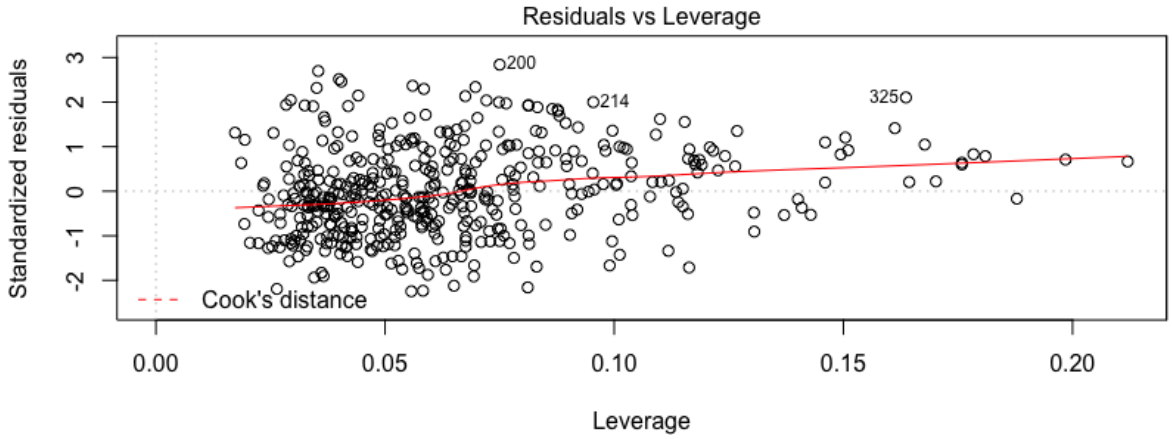


Figure 6.7: Plots of regression model for predicting transformed final exam grades (3/3)



Evaluate on training dataset

Table 6.17: Evaluation of regression model on training dataset for predicting transformed final exam grades

Correlation coefficient	0.79
p-value	$1.42e-73$
Relative absolute error	55.62 %
Root relative squared error	60.84 %
Total Number of Instances	450

Table 6.17 shows us the evaluation of the linear regression model on training dataset for predicting the transformed final exam grades.

Evaluate on testing dataset for predicting transformed final exam grades

Table 6.18: Evaluation of regression model on testing dataset for predicting transformed final exam grades

Correlation coefficient	0.60
Relative absolute error	81.96 %
Root relative squared error	84.51 %
Total Number of Instances	86

Table 6.18 shows us the evaluation of the linear regression model on testing dataset for predicting the transformed final exam grades.

Figure 6.8: Histogram of Regression Difference between Predicted and Actual Grades

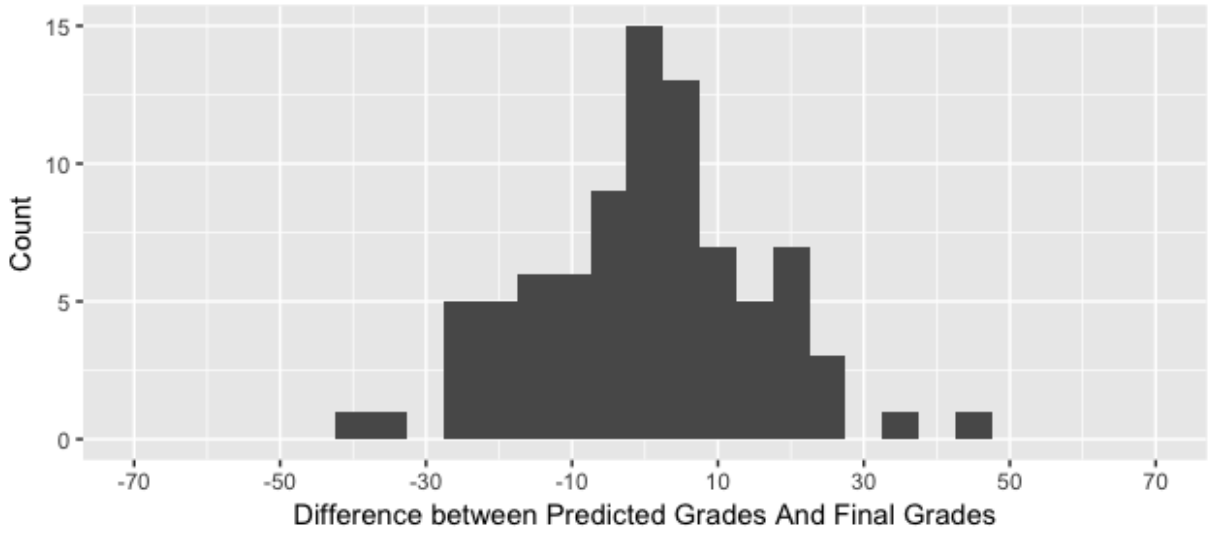


Figure 6.8 shows us the histogram of the difference of the predicted grades and actual grades of testing set. The predicted final exam grades are calculated by applying reversal

transformation. The mean of difference between predicted grades and actual final exam grades (maximum is 120 points) is 0.92 points and the standard deviation is 17.12 points.

In order to compare the performance of the regression model with the classification models, we used the predicted midterm or final exam grades from the regression models to generate the categories and compared it with the actual categories of students. The confusion matrix for midterm exam is shown as table 6.19 and the confusion matrix for final exam is shown as table 6.20.

Table 6.19: Confusion Matrix

poor-performance students	satisfactory students	good-performance students	← predicted as
5	5	0	poor-performance students
5	20	7	satisfactory students
0	19	25	good-performance students

Table 6.20: Confusion Matrix

poor-performance students	satisfactory students	good-performance students	← predicted as
6	4	3	poor-performance students
1	17	9	satisfactory students
1	13	32	good-performance students

In previous classification experiment (section 6.2.5) where we apply decision tree algorithm to submission time intervals, the results of that experiment only show it is possible that the submission time intervals of the tasks are able to reflect some variance among students. However, in this regression experiment, from table 6.19 and table 6.20, we can observe that poor-performance students may have very different behaviour in terms of their submission time intervals. It might indicate that if the way we interpret the feature is different, then we might get different insights. An impressive thing we can see is the left-bottom zero and top-right zero from table 6.19. It indicates the midterm model successfully distinguishes poor-performance students and good-performance students. Students from either category are not classified into the other category.

6.4 Comparison in terms of Poor-Performance Students

We always care more about the poor-performance students comparing to others because they are the students who really needs help urgently. Especially for 1^{st} year engineering students, if they failed out ECE150, they are likely to have much difficulty in passing other courses. Therefore, we will compare the results based on how models behave for poor-performance students.

Table 6.21 shows us the results for poor-performance students on midterm exam and table 6.22 shows us the results for poor-performance students on final exam. Among them, what we really care about is the Precision value for different models. We are targeting to make the least "false alarm", then in this case, if any students are classified as poor-performance students, they are likely to be in trouble. However, by focusing on increasing the precision value, we might put the other poor-performance students who are not classified as poor-performance students aside (low Recall). The reason we let this situation happens is that even if we cannot classify them, the worst case for them is that they remain in a unchanged learning environment where they have to ask for help by themselves. It is acceptable. Same situation towards the poor-performance students who are misclassified into satisfactory-performance students or good-performance students. The importance of the values is $Precision > Fmeasure > Recall > FP\_rate$, F-measure is good way to judge the balance between Precision and Recall, so we put it to the second place. Among them, precision, F-measure and recall are the higher the better while FP_rate is the lower the better.

Table 6.21: Results for Poor-Performance Students on Midterm Exam

	Precision	Recall	F-Measure	FP Rate
Passing Rate	0.00	0.00	0.00	0.03
Testcases Outcomes	0.00	0.00	0.00	0.00
Absolute Number of Submissions	0.20	0.30	0.24	0.16
Relative Number of Submissions	0.24	0.40	0.30	0.17
Lab Attendance	0.18	0.20	0.19	0.12
Submission Time Interval (Decision Tree)	0.29	0.40	0.33	0.13
Submission Time Interval (Linear Regression)	0.50	0.50	0.50	0.07

From table 6.21, we can see for midterm, linear regression with submission time interval

gives us the best result for precision, F-measure and recall.

Table 6.22: Results for Poor-Performance Students on Final Exam

	Precision	Recall	F-Measure	FP Rate
Passing Rate	0.40	0.15	0.22	0.04
Testcases Outcomes	0.25	0.07	0.12	0.04
Absolute Number of Submissions	0.50	0.54	0.52	0.10
Relative Number of Submissions	0.33	0.15	0.21	0.06
Lab Attendance	0.13	0.08	0.10	0.10
Submission Time Interval (Decision Tree)	0.39	0.39	0.39	0.11
Submission Time Interval (Linear Regression)	0.75	0.46	0.57	0.03

From table 6.22, we can see for final exam, linear regression with submission time interval gives us the best result for precision, F-measure, and FP_rate. Only the recall comes the second place.

In addition to those values, we also interested in what the actual grades are for the predicted poor-performance students because although some students are misclassified, their actual grades might be very close to poor-performance students. Figure 6.9 and figure 6.10 are showing the boxplots for the actual grades of predicted poor-performance students.

From figure 6.9, we can see for midterm, the actual grades of predicted poor-performance students from submission time linear regression model are mostly the lowest among all. Also, we have similar result for final exam as shown in figure 6.10.

Therefore, it is reasonable to say that the linear regression model using submission time interval performs better than other models and further researching on this might be the best next step. However, for final exam, it seems the highest point for submission time interval (linear regression) boxplot is an outlier. After we looked into the submission time interval information of that specific student, we found out he/she submit his/her code with an average of 23 hours before deadline while the average hours of other good performance students reside mostly between 100 hours and 140 hours. Therefore, that student is not behaving like a good performance student in terms of the submission time interval feature, but rather a poor performance student whose average hours reside mostly between 10 hours and 60 hours. It makes sense that the student is predicted as poor performance student.

Figure 6.9: Boxplot for Actual Midterm Grades of Predicted Poor Performance Students

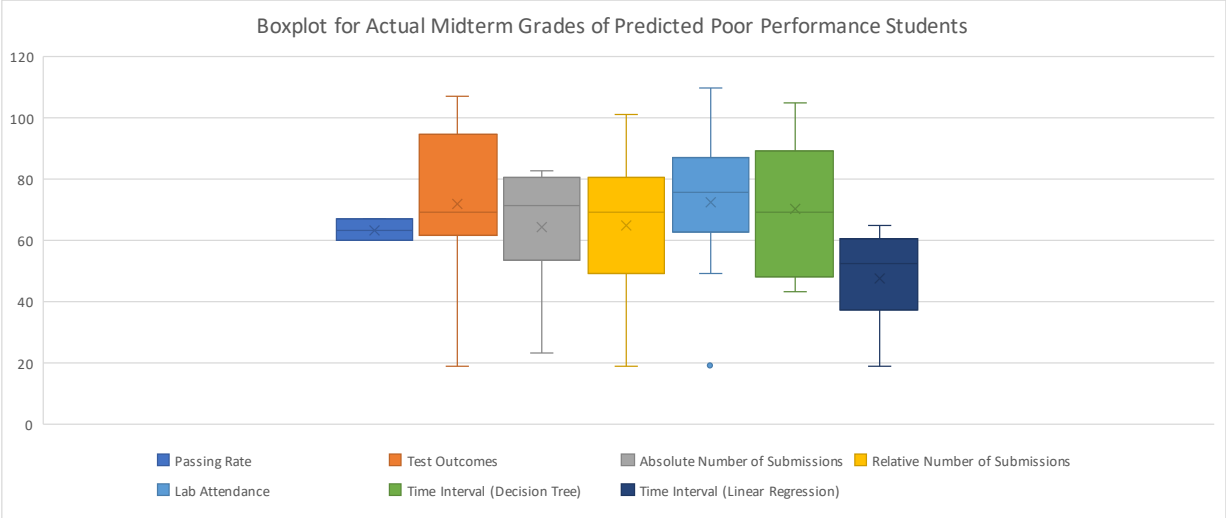
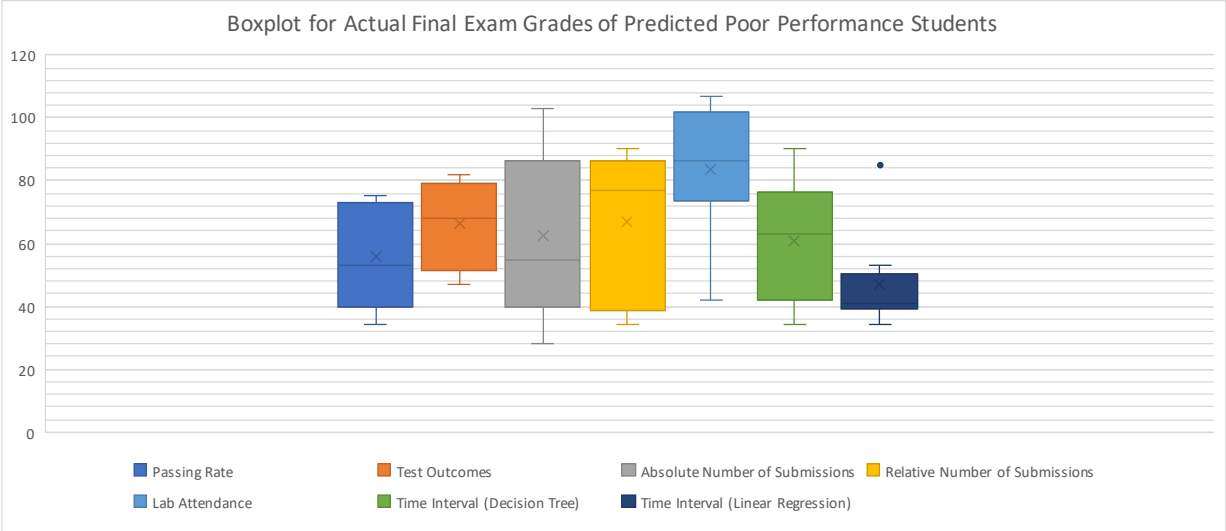


Figure 6.10: Boxplot for Actual Final Exam Grades of Predicted Poor Performance Students



6.5 Correlation between The Time Interval Information of an Assignment and Exam Grades

This section is aiming to answer the third research question. Currently, we have one finding that the time interval of different assignment impacts the prediction on the raw midterm or raw final exam grades differently.

We apply the linear-regression algorithm to different assignment (each assignment contains several tasks with same deadline) and calculate the correlation coefficient and p-value for comparing the effect of the time interval information from each assignment in order to have an insight of the correlation between that and the exam grades. However, in all these experiments, no power transformation is applied because the suggested power might not constant across the experiments. Thus, some underlying assumptions of the residuals of linear regression are not met. The model is as:

$$grades = \beta_0 + \sum \beta_i x_i$$

In the equation,the i is between 1 and the number of tasks of an assignment. For a task, x_i is the time interval between the submission time of a certain submission passing $T\%$ testcases and the deadline. In our experiments, the $T\%$ is set to 75%. The difference from previous experiments is that we combine the time interval information of some tasks instead of all tasks before the midterm exam or final exam. Specifically, we combined the time interval information of tasks of an assignment to do the regression. For example, in figure 2.1, assignment 1 contains 5 tasks. Then in our experiment, we will use the time interval information of those 5 tasks for prediction.

The dataset in the experiments is not split into a training set and a testing set. Also, to avoid coincidence, cross validation is applied. The values are averaged (The exact values are lost from each splits of the dataset during the cross validation. However, the average value would give us a sense of the relation between each assignment and midterm or final exam grades).

Table 6.23: Correlation between assignments and midterm grades

assignment number	0	1	2	3	0-3
number of sub tasks	2	5	3	6	16
p-value	$9.109e-16$	$< 2.2e-16$	$< 2.2e-16$	$< 2.2e-16$	$< 2.2e-16$
Correlation coefficient	0.37	0.46	0.51	0.61	0.63
Mean absolute error	17.09	16.02	15.53	14.03	13.59
Root mean squared error	20.94	19.99	19.35	17.94	17.34
Relative absolute error	94.49%	88.57%	85.90%	77.58%	75.16%
Root relative squared error	92.86%	88.65%	85.78%	79.52%	76.89%
Total Number of Students	428	428	428	428	428

Table 6.24: Correlation between assignments and final exam grades (1/3)

assignment number	0	1	2	3	0-3
number of sub tasks	2	5	3	6	16
p-value	$1.499e-14$	$1.095e-15$	$< 2.2e-16$	$< 2.2e-16$	$< 2.2e-16$
Correlation coefficient	0.36	0.39	0.47	0.56	0.57
Mean absolute error	15.22	15.09	14.61	13.36	13.31
Root mean squared error	18.93	18.73	17.98	16.81	16.69
Relative absolute error	92.60%	91.82%	88.89%	81.28%	81.02%
Root relative squared error	93.02%	91.05%	88.38%	82.60%	82.04%
Total Number of Students	428	428	428	428	428

Table 6.25: Correlation between assignments and final exam grades (2/3)

assignment number	4	5	6	7	4-7
number of sub tasks	3	3	4	2	12
p-value	$< 2.2e-16$	$< 2.2e-16$	$< 2.2e-16$	$< 2.2e-16$	$< 2.2e-16$
Correlation coefficient	0.54	0.55	0.52	0.57	0.63
Mean absolute error	13.57	13.58	13.78	13.23	12.53
Root mean squared error	17.10	16.97	17.35	16.67	15.80
Relative absolute error	82.54%	82.61%	83.84%	80.48%	76.27%
Root relative squared error	84.03%	83.41%	85.27%	81.91%	77.66%
Total Number of Students	428	428	428	428	428

Table 6.26: Correlation between assignments and final exam grades (3/3)

assignment number	0-7	midterm score	midterm + 0-7
number of sub tasks	28	NA	NA
p-value	$< 2.2e-16$	$< 2.2e-16$	$< 2.2e-16$
Correlation coefficient	0.66	0.72	0.77
Mean absolute error	12.23	11.25	10.27
Root mean squared error	15.35	14.20	13.05
Relative absolute error	74.40%	68.42%	62.49%
Root relative squared error	75.42%	69.77%	64.14%
Total Number of Students	428	428	428

Table 6.23 shows that for the midterm, the assignment immediately before the midterm exam has the greatest correlation with the midterm score. It makes sense that if a student cannot manage their time well or they are not understanding the course material, they are unlikely to get a good score on the assignment right before the midterm exam. In addition,

if they are not getting a good score on the assignment, they may not get well prepared for the midterm exam.

Table 6.24, 6.25 and 6.26 show that for the final exam, it turns out that the midterm exam has the greatest correlation with the final exam score. It makes sense that for engineering courses, it becomes tougher when it comes to the content after midterm. Students will have more difficult labs and harder assignments. Because of that, if they fail getting good scores on midterm exam, which reflects that their time management skill are not good and they are not understanding the pre-midterm course content well, they are likely to fail managing the labs and assignments after midterm and they may not get well prepared for the final exam.

Chapter 7

Conclusion and Future Work

This thesis is a summary of our preliminary study in predicting student performance by using the data from the auto-grading system Marmoset. We carried out two types of experiments: classification and regression, aiming to explore the answer of three research questions and more importantly, we want to know what direction it should be for further research.

1. If we put students into categories according to their final exam and midterm performances, can we create a model over the auto-grading data to understand the students' behaviour and predict those categories? More importantly, to predict the students who need help and identify them as early as possible.

2. Can we predict students' raw numerical midterm grades and raw final exam grades from the students' behaviour?

3. Can we find any interesting relations between the features generated (reflecting students' behaviour) from auto-grading system information, grades and student categories?

For classification, we put students into three categories according to their midterm and final exam grades: good-performance students, satisfactory-performance students and poor-performance students. We apply C4.5 decision-tree algorithm to the passing rate, testcases outcomes, lab attendance, number of submissions and submission time intervals, separately, and the results show that, for predicting midterm performance, the classification models of passing rate and testcases outcomes cannot label any poor-performance

students correctly. For final exam, all the features are able to make some level of prediction. Currently, the earliest prediction is made right before midterm.

For regression, we use the time interval between the student's submission and the deadline as the feature, and apply linear regression algorithm to predict the raw midterm or raw final transformed exam grades of the student. The regression model for predicting transformed midterm grades has a correlation coefficient value as 0.72 on training set and the regression model for predicting transformed final exam grades has a correlation coefficient value as 0.79 on training set. Combining with the p-values, both models have p-values smaller than 0.05 meaning those models can explain some percentage of the variances. However, for the regression model for transformed final exam grades, it did not pass the Pearson Chi-Square test. A possible reason is that the way we balance the training set is not very appropriate. Further research needs to be done in selecting the best balancing algorithm. In terms of comparing the predicted non-transformed exam grades and actual exam grades, the results show that for the midterm, the mean of difference between predicted grades and actual midterm grades (maximum is 110 points) is -5.76 points and the standard deviation is 16.44 points. For the final exam, the mean of difference between predicted grades and actual final exam grades (maximum is 120 points) is 0.92 points and the standard deviation is 17.12 points.

For comparing the results in terms of poor-performance students, we put the students into the categories according to their grades as the predicted categories from the linear regression models. Then we calculate the FP rate, Precision, Recall and F-measure of the poor-performance students for both cases. The importance of the values is $Precision > Fmeasure > Recall > FP\_rate$, because we are more care about the percentage of actual poor-performance students among predicted poor-performance students and the balance between precision and recall. For both midterm and final exam, we find out the regression model using the time interval between the student's first reasonable submission and the deadline gives us the best values for precision and F-measure. In addition, for midterm, if we take a look at the confusion matrix table, it correctly distinguished poor-performance students and good-performance students which might indicate there is distinct difference reside in the submission time interval feature. For students who are misclassified as poor-performance students, we compare the boxplot of their actual grades from each model. We find for both midterm and final exam, the linear regression models using submission time interval give the best result.

To compare the correlation between individual assignment and exam grades, we compared the impact of the time interval of different assignment and it shows for both the midterm exam and the final exam, the assignment assigned right before the exams is the closest correlated. However, if we take midterm grades into consideration for the final

exam, we find the correlation of the midterm grades is greater than the correlation of all assignments.

To summary, it is reasonable to say that the linear regression model using submission time interval performs better than other models in terms of predicting poor-performance students and further researching on this might be the best next step. However, since this is only a preliminary auto-grading data exploratory study for predicting student performance, we can only get limited insight from the data and features. Future work might include performing additional experiments on combining different features to explore the data and as we collect more data, we can reach more definitive conclusions.

References

[1] D. Aha and D. Kibler. Instance-based learning algorithms. Machine Learning, 6:37–66, 1991.

[2] Paulo Blikstein. Using learning analytics to assess students' behavior in open-ended programming tasks. In Proceedings of the 1st International Conference on Learning Analytics and Knowledge, pages 110–116. ACM, 2011.

[3] Leo Breiman. Random forests. Machine Learning, 45(1):5–32, 2001.

[4] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. Classification and Regression Trees. Wadsworth International Group, Belmont, California, 1984.

[5] M. Bremer. Multiple Linear Regression. http://mezeylab.cb.bscb.cornell.edu/labmembers/documents/supplement%205%20-%20multiple%20regression.pdf, Accessed: 2018-06-20.

[6] Hana Bydžovská. A Comparative Analysis of Techniques for Predicting Student Performance. In Proceedings of the 9th International Conference on Educational Data Mining 2016, 2016.

[7] Tianfeng Chai and Roland R Draxler. Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature. Geoscientific Model Development, 7(3):1247–1250, 2014.

[8] John M Chambers. Graphical methods for data analysis. CRC Press, 2018.

[9] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. J. Artif. Int. Res., 16(1):321–357, June 2002.

[10] Nancy Chinchor and Beth Sundheim. MUC-5 evaluation metrics. In Proceedings of the 5th Conference on Message Understanding, pages 69–78. Association for Computational Linguistics, 1993.

[11] William W. Cohen. Fast Effective Rule Induction. In Twelfth International Conference on Machine Learning, pages 115–123. Morgan Kaufmann, 1995.

[12] R Dennis Cook. Detection of influential observation in linear regression. Technometrics, 19(1):15–18, 1977.

[13] Evandro B Costa, Baldoino Fonseca, Marcelo Almeida Santana, Fabrísia Ferreira de Araújo, and Joilson Rego. Evaluating the effectiveness of educational data mining techniques for early prediction of students' academic failure in introductory programming courses. Computers in Human Behavior, 73:247–256, 2017.

[14] Gerben Dekker, Mykola Pechenizkiy, and Jan Vleeshouwers. Predicting students drop out: A case study. In Educational Data Mining 2009, 2009.

[15] Desire2Learn company. Waterloo LEARN. https://uwaterloo.ca/learn-help, Accessed: 2018-06-20.

[16] Christopher Douce, David Livingstone, and James Orwell. Automatic test-based assessment of programming: A review. Journal of Educational Computing Research, 5(3), September 2005.

[17] Harris Drucker. Improving regressors using Boosting techniques. In ICML, volume 97, pages 107–115, 1997.

[18] Nitesh V. Chawla et. al. Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16:321–357, 2002.

[19] Clay Ford. Normal Q-Q. http://data.library.virginia.edu/understanding-q-q-plots/, Accessed: 2018-06-20.

[20] George E Forsythe and Niklaus Wirth. Automatic grading programs. Communications of the ACM, 8(5):275–278, 1965.

[21] John Fox. Applied regression analysis, linear models, and related methods. Sage Publications, Inc, 1997.

[22] Eibe Frank and Ian H. Witten. Generating Accurate Rule Sets Without Global Optimization. In J. Shavlik, editor, Fifteenth International Conference on Machine Learning, pages 144–151. Morgan Kaufmann, 1998.

[23] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In Thirteenth International Conference on Machine Learning, pages 148–156, San Francisco, 1996. Morgan Kaufmann.

[24] J.H. Friedman. Stochastic Gradient Boosting. Technical report, Stanford University, 1999.

[25] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian Network Classifiers. Mach. Learn., 29(2-3):131–163, November 1997.

[26] Francis Galton. Regression towards mediocrity in hereditary stature. The Journal of the Anthropological Institute of Great Britain and Ireland, 15:246–263, 1886.

[27] Vincent Gramoli, Michael Charleston, Bryn Jeffries, Irena Koprinska, Martin McGrane, Alex Radu, Anastasios Viglas, and Kalina Yacef. Mining Autograding Data in Computer Science Education. In Proceedings of the Australasian Computer Science Week Multiconference, ACSW '16, pages 1:1–1:10, New York, NY, USA, 2016. ACM.

[28] Jiawei Han, Jian Pei, and Micheline Kamber. Data mining: Concepts and Techniques. Elsevier, 2011.

[29] Trevor Hastie and Robert Tibshirani. Classification by Pairwise Coupling. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, Advances in Neural Information Processing Systems, volume 10. MIT Press, 1998.

[30] Simon Haykin. Neural Networks: A Comprehensive Foundation. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[31] Marti A. Hearst. Support Vector Machines. IEEE Intelligent Systems, 13(4):18–28, July 1998.

[32] J. B. Hext and J. W. Winings. An automatic grading scheme for simple programming exercises. Communications of the ACM, 12(5):272–275, May 1969.

[33] Jerry L Hintze and Ray D Nelson. Violin plots: a box plot-density trace synergism. The American Statistician, 52(2):181–184, 1998.

[34] David C Hoaglin. John w. tukey and data analysis. Statistical Science, pages 311–318, 2003.

[35] R.C. Holte. Very simple classification rules perform well on most commonly used datasets. Machine Learning, 11:63–91, 1993.

[36] Finn V. Jensen. Introduction to Bayesian Networks. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1996.

[37] Dorina Kabakchieva. Predicting student performance by using data mining methods for classification. Cybernetics and information technologies, 13(1):61–72, 2013.

[38] Sushilkumar Kalmegh. Analysis of WEKA Data Mining Algorithm REPTree, Simple Cart and RandomTree for Classification of Indian News. 2015.

[39] Chandrika Kamath. Scientific Data Mining: a practical perspective, volume 112. Siam, 2009.

[40] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy. Improvements to Platt's SMO Algorithm for SVM Classifier Design. Neural Computation, 13(3):637–649, 2001.

[41] Lumbini P Khobragade and Pravin Mahadik. Students academic failure prediction using data mining. International Journal of Advanced Research in Computer and Communication Engineering, 4, 2015.

[42] Ron Kohavi. A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. In Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[43] Irena Koprinska, Joshua Stretton, and Kalina Yacef. Students at Risk: Detection and Remediation. In Proceedings of the 8th International Conference on Educational Data Mining, pages 512–515, 2015.

[44] H.F. Korth, S. Sudarshan, and P. Abraham Silberschatz. Database System Concepts. McGraw-Hill Education, 2010.

[45] Niels Landwehr, Mark Hall, and Eibe Frank. Logistic Model Trees. 95(1-2):161–205, 2005.

[46] Lucien Le Cam. The central limit theorem around 1935. Statistical science, pages 78–91, 1986.

[47] David D Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In European conference on machine learning, pages 4–15. Springer, 1998.

[48] Brent Martin. Instance-based learning: Nearest Neighbor With Generalization. Master's thesis, University of Waikato, Hamilton, New Zealand, 1995.

[49] Urs von Matt. Kassandra: the automatic grading system. Technical report, 1998.

[50] Jessica McBroom, Bryn Jeffries, Irena Koprinska, and Kalina Yacef. Mining behaviors of students in autograding submission system logs. In Proceedings of the 9th International Conference on Educational Data Mining, pages 159–166, 2016.

[51] Mary Natrella. NIST/SEMATECH E-Handbook of Statistical Methods. http://www.itl.nist.gov/div898/handbook/.

[52] Nguyen Thai Nghe, Paul Janecek, and Peter Haddawy. A comparative analysis of techniques for predicting academic performance. In Frontiers in Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports, 2007., pages T2G–7. IEEE, 2007.

[53] Dr. Iain Pardoe. Residual vs Fitted plot. https://onlinecourses.science.psu.edu/stat501/node/36, Accessed: 2018-06-20.

[54] Karl Pearson. X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 50(302):157–175, 1900.

[55] J. Platt. Fast Training of Support Vector Machines using Sequential Minimal Optimization. In B. Schoelkopf, C. Burges, and A. Smola, editors, Advances in Kernel Methods - Support Vector Learning. MIT Press, 1998.

[56] J. Ross Quinlan. Induction of decision trees. Machine learning, 1(1):81–106, 1986.

[57] Ross Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[58] Ross J. Quinlan. Learning with Continuous Classes. In 5th Australian Joint Conference on Artificial Intelligence, pages 343–348, Singapore, 1992. World Scientific.

[59] R Development Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[60] Sylvain Roy. Nearest Neighbor With Generalization. 2002.

[61] JP Royston. Algorithm as 181: the w test for normality. Journal of the Royal Statistical Society. Series C (Applied Statistics), 31(2):176–180, 1982.

[62] RStudio Team. RStudio: Integrated Development Environment for R. RStudio, Inc., Boston, MA, 2015.

[63] Amirah Mohamed Shahiri, Wahidah Husain, and Nuraini Abdul Rashid. A review on predicting student's performance using data mining techniques. Procedia Computer Science, 72:414–422, 2015.

[64] Jaime Spacco, William Pugh, Nat Ayewah, and David Hovemeyer. The marmoset project: An automated snapshot, submission, and testing system. In Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications, OOPSLA '06, pages 669–670, New York, NY, USA, 2006. ACM.

[65] Jaime Spacco, Jaymie Strecker, David Hovemeyer, and William Pugh. Software repository mining with Marmoset: An automated programming project snapshot and testing system. In Proceedings of the Mining Software Repositories Workshop (MSR 2005), St. Louis, Missouri, USA, May 2005.

[66] Pedro Strecht, Luís Cruz, Carlos Soares, João Mendes-Moreira, and Rui Abreu. A comparative study of classification and regression algorithms for modelling students' academic performance. International Educational Data Mining Society, pages 392–395, 2015.

[67] Marc Sumner, Eibe Frank, and Mark Hall. Speeding up logistic model tree induction. In 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, pages 675–683. Springer, 2005.

[68] S. K. Thompson. Stratified Sampling. in Sampling, John Wiley & Sons, Inc., Hoboken, NJ, USA, third edition, 2012.

[69] Ravi Tiwari and Awadhesh Kumar Sharma. A Data Mining Model to Improve Placement. International Journal of Computer Applications, 120(12):36–38, 2015.

[70] Marcel van Gerven and Sander Bohte. Artificial Neural Networks as Models of Neural Information Processing. Frontiers Media SA, 2018.

[71] Y. Wang and I. H. Witten. Induction of model trees for predicting continuous classes. In Poster papers of the 9th European Conference on Machine Learning. Springer, 1997.

[72] Hadley Wickham and Lisa Stryjewski. 40 years of boxplots. 2011.

[73] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2016.