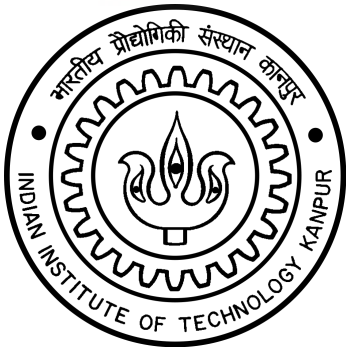
Automated Grading Tool for Introductory Programming

A thesis submitted
in Partial Fulfillment of the Requirements
for the Degree of

Master of Technology

by

Sagar Parihar
Roll Number: **13111055**



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

November, 2015

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

iii

POST GRADUATE OFFICE
THESIS SUBMITTED
ON 27·11·2015
I. I. T., KANPUR - 16

CERTIFICATE

It is certified that the work contained in this thesis entitled
"*Automated Grading Tool for Introductory Programming*",
by *Sagar Parihar (roll number 13111055)*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Arnab Bhattacharya

(Dr. Arnab Bhattacharya)
Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur
Kanpur-208016

Amey Karkare

(Dr. Amey Karkare)
Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur
Kanpur-208016

November, 2015

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

ABSTRACT

Grading, especially for multiple computer programs, for an introductory computer programming course, requires a considerable amount of human time and effort. Consequently, a large number of teaching assistants (TAs) are assigned to manually grade the programs each week. Since the instructions for grading are, by nature, imprecise, there can exist a lot of variability between the grading of two TAs since it is prone to unintentional bias. A more severe problem is the perception of bias among the students whose programs are graded. The main motivation of this thesis is to develop an automated grading tool that will address the above problems. The thesis explores several factors that contribute to the final marks including number of test cases passed, number of successful compilation attempts, etc. A regression model is run to determine the weights of these factors. Results obtained from the automated tool shows that the results are robust and has lesser variability from a carefully graded solution than exhibited between two TAs. The tool is integrated to the backend of the Intelligent Tutoring System (ITS) developed at Indian Institute of Technology, Kanpur for handling introductory programming level courses. The ITS system provides a unified frontend for students to write, compile and check their codes.

Dedicated to my Parents and Family

Acknowledgements

First and foremost of all I would like to express immense gratitude to Prof. Arnab Bhattacharya and Prof. Amey Karkare, my guide and my co-guide for their constant motivation, inspiration, guidance and support. Their expertise & experience in the subject and their timely & accurate guidance are the only reason that I have been able to complete my thesis work.

Only after spending two year in IIT Kanpur I realised how lucky I was to learn from renowned faculties of Computer Science & Engineering department. No amount of gratitude is enough and I will always be in debt to the professors of IITK. Not to mention the world class state-of-art infrastructure provided by the institute and maintained by the staff. I am grateful them.

I am in lifelong debt to my mother Preeti Parihar and my father Sanjay Parihar for their timeless unconditional love and support. Also, my younger brothers Shikhar and Prithvi were a source of motivation. All along my uncles Rajay Parihar, Anupam Chauhan and Anuj Chauhan and their families were source of great strength. I would like to extend my sincere thanks to them.

Thanks to PhD scholars Tejas Gandhi and Shahbaz Khan for theirs suggestions. I would also like to mention my batch mates not only for helping me throughout my M.Tech. but also making my stay at IITK memorable. Among them are Rishabh, Ayan, Abhra, Abdu, Mahajan, Purohit, Nayek, Chauhan, Bhoyar, Shail, Shashwat, Hegde, Shiv, Modi, Sayantan, Mohit, Satyajeet, Ankit, Tiwari, Naman, Rajesh, Rajdeep and Dhananjay. Without them life would have been extremely boring.

Contents

List of Tables

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

List of Figures

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Chapter 1

Introduction

Since its advent, computer technology has seen tremendous growth and development. It has now evolved to become an integral part of today's world and will remain in the future. Computers help different organisations, industries and service sectors to scale up. Automating several processes have dramatically reduce human error as well as their time requirement. These machines have also revolutionised communication. Entertainment world have also benefited a lot. Basically, computers are omnipresent in the present world.

With computers omnipresent, everyone spends significant duration of time interacting with these machine or more precisely interaction with the programs running on these machine. And so it important to have basic level knowledge about computer and computer programs, at least for a professional. This can be inferred from the fact that many universities programs have compulsory courses in introductory computer programming.

Such introductory computer programming courses are generally run in computer labs by assigning simple programming problems to students. The students code solutions to these programming problems and submit them for grading. The students then receive grades and feedback accordingly. Grading is not only responsible for motivation and for providing direction learning to students but is also an implicit feedback.

Maintaining timely and quality grading consumes lot of human resources. Teach-

ing assistants are post graduate students appointed to assist professors in course work. The professors get assistance in monotonous work and teaching assistants (TAs) get learning experience and some financial aid. Universities and academic institutions appoint a number of teaching assistants for this purpose. These TAs give a considerable amount of time and effort.

In spite of all the input, grading is not immune to human error and the turnaround time is not as fast as required for optimal learning. Not to mention the inconsistency among different teaching assistants (TAs) which causes variablity in grading. Unintentional bias in judgement may also affect the grading. Moreover, manual grading limits the number of students that can be enrolled in such introductory programming courses.

Grading policies for such programming assignments are quite objectively defined. We also have means to obtain various feature and characteristics for a solutions submitted. It is an alluring idea, to utilize these fact and come up with an automated grading system having performance comparable to that of the teaching assistants (TAs). We are motivated by prospects of developing such a real-time automated grading system that can mitigate the problems faced while grading large number students. This system can be used to give students an estimate of grades as soon as they submit a solution or assist TAs by giving recommendations when they are grading. It can also be used to detect potential grading anomalies.

Intelligent Tutoring System (ITS) [Das15] is cloud based base web application designed, developed and hosted at IIT Kanpur under the supervision of Prof. Amey Karkare by the department of Computer Science and Engineering, to conduct introductory programing course. ITS has been used successfully for two semester to run introductory programming course named ESc101. It documents a wide range of students' activity and feature and characteristics of solutions submitted by the students. Availability of such a rich data set to work on and platform to test, strengthens our motivation further and gives a start point to our venture.

1.1 Objective

Firstly, we state the assumptions made for our problem. We have an online platform on which students are assigned programming problems. Students are required to solve each of these problems and submit code in a decided programming language as solution. This platform is capable of compiling and running the code. The solution is judged by comparing the test cases output of solution for fixed set of test cases against corresponding correct answers. These test cases are set along with the problem.

The platform logs a wide variety of data relate to both students' activities and solution submissions. With access to the log database we can mine for the metadata needed.

Our goal is to design and develop an automated grading model to grade or mark solutions submitted for programming problems assigned to students. The model uses the above discussed meta data or features to compute and output the grades. The grades is a real number between zero and maximum marks for the problem.

1.2 Thesis Outline

The structure of the thesis is as follows.

Chapter 2, named Background and Related Work, aims to explain concepts, methods and tools required to understand the work. These concepts have been used or applied directly or indirectly in this thesis. Final section contains a brief discussion on work done previously in the area we have worked.

Chapter 3, named Grading Model, presents the grading model designed and developed in this thesis. Incremental development is explained along with the intuitive ideas behind it.

Chapter 4, named Learning Weight Parameters for Grading Model, describes what and how were the weights for the grading model were learned.

Chapter 5, named Experiments and Results, explains the experiments done to evaluate the performance of our grading model/tool and presents the results.

Chapter 6 , named Autograding ReST Service, discusses about a backend server developed to serve grades computed by the model/tool for submissions made on ITS. The server is developed to be integrated to Intelligent Tutoring System (ITS).

Chapter 7, named Conclusions and Future Scope, presents some conclusions we have derived from this thesis and discusses about potential future work.

Chapter 2

Background and Related Work

2.1 Platform and Related Concepts

We discuss about online education platforms and introduce a similar platform, Intelligent Tutoring System, also known as ITS [Das15]. We also describe concepts related to the platform as well as our work.

2.1.1 Intelligent Tutoring System

We have worked closely with Intelligent Tutoring System (ITS) for significant amount of time. In fact it was because of ITS that we had access to such rich real world data to carry out our analysis and conduct necessary experiments.

ITS is an web based education platform that conducts courses online. It was developed by my colleague Rajdeep Das at IIT Kanpur as his M.Tech. thesis under the supervision of Prof. Amey Karkare in the department of Computer Science and Engineering. It aimed to provide enhanced introductory programming learning experience to the masses by the means of technology and at the same time to be used as a platform for data collection and analysis.

Computer aided education [Cin13] has provided access to quality education online through various web based platforms. It has crossed geographic barriers to reach to the masses, enabled community interaction and has provided means to manage

large scale courses. However, such online courses that run on these platforms on massive scale lack effective problem solving experience and do not effectively cater to requirements at individual level. And some courses like introductory programming require problem solving for effective learning and in some cases need feedback at individual level.

There are a few issues that makes this experience difficult to deliver for generic online education platforms. These factors include requirement of a standard Linux programming environment with configured build tools and students' capability to learn and effectively use these programming tools. Moreover, the students need not learn Linux environment and its programming tools to learn computer programming.

ITS was developed to address these issues of providing effective problem solving experience and to provide feedback at individual level, for introductory programming courses, on an online platform. It attempts to tackle these issues from an architectural angle. ITS is a cloud based web app with the only requirement of standard web browser with connectivity. The app is login based which provides editor to write and submit programs. The program is compiled and run on the cloud systems and the results get displayed on the browser.

Since all the functionalists of writing, compiling and running the code are now on the web app, it is no longer required for students to set up their local environment and learn to use it. Also as the web app requires login, every activity can be tracked and documented. This personalized information can be used to provide individual feedback to students. In this manner ITS attempts to provides a solution to our problems.

2.1.2 Grading

In any course it is a general practice to assign students with tasks or homework or assignments and conduct examinations. These assignments help students to practice and get a better understanding of the concepts and the examinations help in evaluation.

The assignments and examinations are then judged and awarded score, i.e. grades according to their degree of correctness. This process of judgement of correctness to award grades or marks to a assignment is called grading or marking.

From a student's perspective, who is undergoing a course, tracking her progress is extremely important, as is, from a instructor's perspective providing students feedback. Grading serves a basic level of feedback to student and also as a basic level measure to track performance of student. Thus, grading is of significant importance to maintain the effectiveness and quality of programming courses for it is one of the key factors that drive the motivation and control the direction of learning of the students.

Many introductory programming courses grade manually by hand the programs written by students. Universities use Teaching Assistants (TAs) for the purpose of grading. TAs are generally post graduate students that assist professors. A significant amount of effort and time is invested to maintain the quality of grading. This process of grading may become a bottleneck when the course is scale to larger number of students.

ITS also uses a similar system where students are assigned programming problems. The students then submit solutions to these problems which are graded by TAs manually by hand.

2.2 Datasets

All along we have worked with real world datasets collected by ITS. These datasets have been collected for ESc101: Fundamentals of Computing, an introductory programming course conducted in even semester of academic year of 2014-15 at IIT Kanpur. C programming language was used for the entire course.

ESc101 conducted weekly labs of about 3 hours for enrolled students where students were assigned programming problems. The students were required to code solution to their assignments and submit it within the lab duration. The solution were tested against a set of test cases (TCs). Finally the Teaching Assistants (TAs)

graded the submitted solutions by awarding them marks/grade. TAs judged the submitted codes by manual inspection and by considering the number of TCs passed. Lab exam was also conducted which carried much more weightage than the lab assignments. Grading was done in similar manner but with a more robust grading process to ensure precision.

We have used 2 variations of these data sets.

2.2.1 Assignment Grades Dataset

There were 2 types of events conducted, lab and exam. This dataset refers to the data collected for lab events.

ITS logged various activities of each student. It also stored the submissions and data related to these submission such result of test cases (TCs). Most importantly, the final grades awarded by the teaching assistant we accurately recorded. A grade for a submission or a assignment is a whole number upto maximum marks for the assigned problem.

Assignment Grades Dataset consists of all the lab problem assignments of enrolled students along with their submissions and data related these submissions with the awarded grades.

This dataset consists of 64 sets of soultion submissions, each for a differnet question in some scheduled lab. Each solution submission set consists of 99 to 101 student solution submissions. We have graded marks and maximum marks for each lab schedule for each question along with logged data related to the submission.

2.2.2 Exam Grades Dataset

Out of the 2 types of events, lab and exam, this dataset refers to data collected for exams.

Similar to assignment grade dataset, this dataset also contains logged student activities and submission data including number of test cases passed. The grading data for exam is bit different. Each solution submission in lab exam is graded by 2

different TAs. If the difference between the marks awarded was below a threshold higher marks were awarded, else re-evaluation was done to award grades. These grades are also whole numbers upto maximum marks for the assigned problem.

Exam Grades Dataset of consists all the exam problems of enrolled students along with their submission and data related to these submission with 2 different grades, final grade awarded and grade awarded by the alternate TA which were not considered.

This dataset consists of 8 sets of solution submissions, each for a different question in a scheduled lab exam. Each solution submission set consists of 201 students.

2.3 Linear Regression with Ordinary Least Squares

Liner Regression [MPV12] can be described as an approach to model relationship between a *dependent variable* say y and set of *explanatory variables* say X.

As the name linear suggests this approach considers a linear function to describe the relationship between the dependent variable and the explanatory variables.

This relationship can be described by the following expression, for i^{th} data point, assuming X consists of p variables, $x_{i1}, x_{i2}, \ldots x_{ip}$ and a constant x_{i0}.

The constants, $\forall i\ x_{i0}$, can be set to 0 or 1 depending on the underlying system being modeled. Setting the constants to 1 implies that the model assumes that y depends on a constant also whereas setting it to 0 assumes no dependence on constant.

Linear regression model in the linear relationship also assumes an *error variable* ε, which may be defined as an unobserved random variable.

$$y_i = \beta_0 x_{i0} + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i$$

The above expression is only for a single i^{th} data point. Extending the expression for all n data points in vector form, we have.

$$y = X\beta + \varepsilon$$

where,

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, X = \begin{bmatrix} x_{10} & x_{11} & x_{12} & \dots & x_{1p} \\ x_{20} & x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n0} & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}, \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}, \varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

This expression can be used to estimate the parameters β if both y and X are known for a set of data points. There are various techniques to estimate parameters β. One such simple and common estimator is Ordinary Least Squares [MPV12].

Ordinary Least Squares (OLS) is a technique that can be used estimate parameters β over a linear regression model when both dependent variable y and explanatory variables X are known for all data points in a data set. The estimation is done by minimizing the sum of squared error/residuals. The closed-form expression to estimate parameters β is as follows.

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

This closed-form expression can be conveniently used to estimate required parameters for any linear regression model.

2.4 Performance Measures

We briefly discuss some performance measures that are used to analyse and compare performance of our grading model/tool.

2.4.1 Spearman's Rank Correlation Coefficient (ρ)

Spearman's Rank Correlation Coefficient [WM03], denoted by ρ or r_s, is statistical measurement of association strength between two ranked variables. It can be defined as nonparametric measure of dependence among two variables.

Its value is a real number that ranges between -1 and $+1$. A value $+1$ or -1 for two variables implies that each variable is a perfect monotone function of the other, where as, 0 implies that there exist no association.

The positive and negative signs indicates direction of association. Positive sign $(+)$ indicates that the value of one variable tends to increase with increase in value of other and negative sign $(-)$ indicates that the value of one variable tends to decrease with the increase in the value of other. The magnitude of ρ is closer to 1 when the two variables are perfectly directly monotonically related.

Assuming the 2 variables can take n values and d_i is the difference in ranks in the i^{th} corresponding values of the two variables, we can calculate the Spearman's rank correlation coefficient as follows.

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

Thus Spearman's rank correlation coefficient can be used to understand the ranking relation among two variables.

2.4.2 Mean Average Precision (MAP)

Information retrieval systems attempt to responds to queries with the most appropriate subset of items from the universe of items. Example of such query may be top-k or bottom-k by some criteria.

Precision [MRS08] is an evaluation measure for models that can act as information retrieval system. It measures the quality of retrieved set of items over a particular query.

Precision, for a particular query, can be defined as following.

$$\text{precision} = \frac{|\{\text{relevant items}\} \cap \{\text{retrieved items}\}|}{|\{\text{retrieved items}\}|}$$

Where relevant items are the correct set of items that system should return for the query and retrieved items are the set of items returned by the system as an response to the query.

Precision serves as a measure when the result is in form of set. We need an extended measure for systems that respond with ranked sequence of items. Average Precision [MRS08] considers the order or rank of the sequence of items returned.

It can be defined as average of precisions at every position in the ranked sequence of items. It is computed as follows.

$$\text{AveP} = \frac{\sum_{k=1}^{n}[P(k) \times rel(k)]}{n}$$

where,

n is the number of items queried for,

$P(k)$ is the precision for top-k items,

$rel(k)$ is an indicator function equal to 1 if item at rank k is relevant and 0 otherwise.

A system evaluation cannot be done on the bases of a single query. We need to take multiple queries into account. Mean Average Precision (MAP) [MRS08] is a measure that takes into account multiple queries. MAP may be defined as mean of Average Precisions for a set of quires and is computed as following.

$$\text{MAP} = \frac{\sum_{q=1}^{Q} \text{AveP}(q)}{Q}$$

where,

Q is the number of queries under consideration,

AveP(q) is the Average Precision for q^{th} query.

Thus, MAP can be used to measure correctness of a model which can act as a information retrieval system.

2.5 Related Work

In this section we discuss some of the work done previously which is related to the work done in this thesis. It is related to assessment or marking or evaluation or testing or grading of programming problem solutions in an environment somewhat similar to that of Introductory Programming courses. Discussion is not exhaustive but limit to the study and survey done while working on this thesis.

BOSS by Luck and Joy [LJ99] in 1999 presented a system that managed code submissions and was capable of semi-automated testing of student coursework. But it could only return either 100% or 0%. So BOSS could act only as an assistant to actual marker/grader.

CodeLab also know as WebToTeach by Arnow and Barshey [AB99] in 1999 develop a system which was web based which could accept submission via web client. It also assessed solutions at binary level and focused on short answers. It also had capability to cover a range of concepts by enforcing students to solve particular problems before allowing them to move on.

RoboProf by Daly [Dal99] also in 1999 is a more wider system which is capable of handling presentation of notes and exercises to students, accepts solution submissions and test them against test data, provides feedback to the student and achieves the results. Although the feed back is immediate but is very limited to binary comparison of students' output against the test data.

Automated Assessment and Experiences of Teaching Programming by C. A. Higgins et al [Hig+05] in 2005 presented a more improved system with richer and wider range of functionalities. It was capable of catering to different user profiles namely Student, Tutor, Teacher, Administrator and Developer and had modules of Submission Marking & Archiving, Courses, Account and Auditing, all with a good interface. For marking the tool extracted a set of features and marked according a customizable marking mechanism.

Some systems were developed that used static analysis to compare submitted

solutions against model programs provided forehand. Few such systems are ELP by Truong, Roe & Bancroft in 2005 [TBR05], WAGS by Norshuhani et al in 2006 [Zam+06] and system proposed by Khirulnizam et al in 2007 [K+07]. These system needs to be provided with different model solutions to cover the variety of student solutions.

Many automated assessment systems implement dynamic testing of submissions for Introductory Programming courses. Examples of such systems include Ala-Mutka in 2005 [Ala05], TRY by Reek in 1989 [Ree89], Scheme-robo by Saikkonen [SMK01], Malmi & Korhonen in 2001, Online Judge by Brenda, Andy, Andrew & Chong in 2003 [Che+03], Quiver by Ellsworth, Fenwick & Kurtz in 2004 [EFK04] and RoboProf by Daly & Horgan in 2004 [DH04].

The BOSS Online Submission and Assessment System by Mike Joy, Nathan Griffiths & Russell Boyatt [JGB05] in 2005 developed an assessment mechanism for solution of Introductory Programming course which along with dynamic testing of test cases used JUnit tests for Java Programs.

Ability-training-oriented automated assessment in introductory programming course introduced AutoLEP by Tiantian Wang, Xiaohong Su, Peijin Ma, Yuying Wang, Kuanquan Wang [Wan+11] in 2010 introduced a system which provided students feedback regarding syntactic and structural defects immediately at the time of submission. Assessment is based on dynamic testing and system returns feedback with test cases passed and failed by the submission. Also, a static analysis against model solutions based semantic analysis feedback is generated.

Code Hunt: Experience with Coding Contests at Scale by Judith Bishop, R. Nigel Horspool, Tao Xie & Nikolai Tillmann [Bis+15] in 2015 presented a platform CodeHunt that hosts coding contests. The platform assigns skill rating and score to the submissions. Due to nature of the contest, rating represented by an integer value 1, 2 or 3, is determined on the succinctness of the submission. The smaller the submitted code higher the score. Score of a submission is rating multiplied by the problem's difficulty. Problem difficulty is also an integer ranging from 1 to 5.

Participant score is accumulated total score.

We have studied all the discussed related worked to understand the problem we address in this thesis. However, we have developed our own idea from intuition.

Chapter 3

Grading Model

This chapter presents the grading model developed in this thesis, the core of our work. We explain the theoretical idea in an incremental fashion to help readers empathise with the development derivation of the grading model for our grading tool. Initially, the intuitive idea is discussed and the feature set used is listed. Our primitive grading model, namely initial grading model, which consist of multiple steps follows. Finally, we present a simplified single step model, final grading model, which is similar to the initial multiple step model but is more elegant and is better suited for analysis, experimentation and implementation.

3.1 Intuitive Idea and Feature Set

For grading programming assignments of introductory programming courses, universities deploy teaching assistants (TAs). TAs are postgraduate students that help the instructors with courses by taking up tasks such as grading of assignments. TAs follow a set grading policy. This grading policy is quite objectively defined, at least for courses like introductory programming.

Using the fact that programming assignments have objective grading policies we attempt to capture these policies and develop a grading model that performs similar the TAs. We try to identify the major factors involved in grading policies.

A close look at the grading policies guideline reveals the following factors at play

for a programming problem solution submission.

- grades $\propto$ fraction of test cases (TCs) passed
- grades $\propto$ 1/time taken to solve
- grades $\propto$ fraction of successful compilations
- grades $\propto$ 1/program run-time
- grades $\propto$ 1/program memory requirement
- grades $\propto$ 1/global and unused variables
- grades $\propto$ indentation precision
- grades $\propto$ relevant comments

We have used the first 3 factor in our model. Other factors are not currently used. The reason for not using these factors directly is that they have minimal impact on grades and they are indirectly correlated with the first 3 factors. Feature set extraction of first 3 factors is more efficiently plausible than the other factors. We now discuss the application of these first 3 factors and introduce the feature score functions associated with each feature.

3.1.1 Fraction of Test Cases Passed

The primary and most important factor to determine the correctness of a solution program is the number of test cases it passed out of all the set test cases.

Each problem on the introductory programming course web platform consists of a set of test cases. These test cases are preset by the problem setter. Each test case consists of an input and the corresponding correct output. So, if a code submission, for the test case input produces output equal to the corresponding correct test case output it is said that the code submission has passed this test case, else it is said to have failed or not passed.

Now, we attempt to understand the grading policy on the number of test cases passed. If a submission passed all the test cases it is considered a correct solution and all but few marks are awarded depending upon other guidelines. If there exists a failed test case the solution is considered incorrect and partial marks are awarded proportional to the number of test cases passed.

We attempt to apply both the guidelines to our grading model. We present score functions for each grading guideline. These functions will use the number of test cases passed and total number of test cases to compute a score, which is a real number in the range 0 to 1, both inclusive.

Firstly, we present the score function for a complete solution having k test cases passed out of n test cases, $f_c(k, n)$. We define the complete solution score function as an indicator function.

$$f_c \in_{\mathbb{R}} \{0, 1\}$$

$$f_c(k, n) = \begin{cases} 1 & \text{if } k = n \\ 0 & \text{otherwise} \end{cases}$$

The complete solution score function, for submission passing k out of n test cases, can also be defined as follows.

$$f_c(k, n) = \left\lfloor \frac{k}{n} \right\rfloor$$

The above function, complete solution score function is used to apply complete solution grading guideline.

Secondly, we present the partial solution score function to capture the partial solution cases. Cases when not all test cases pass for a submission are captured by this function. Partial solution sore function, $f_p(k, n)$ is a simple linear function of number of test cases passed k and total number of test cases n for a submission. It evaluates to a real number and ranges from 0 to 1.

$$f_p \in_{\mathbb{R}} [0, 1]$$

$$f_p(k, n) = \left(1 - \left\lfloor \frac{k}{n} \right\rfloor\right) \times \frac{k}{n}$$

We use partial solution score function to apply the partial solution grading guideline.

Hence, using these two score function we attempt to capture the feature of number of test cases passed for a submission.

3.1.2 Time Taken to Solve

Another significant factor to distinguish a proficient programmer from a novice one is on the basis of time take to solve the problem. Clearly, a better programmer will require much less time time to solve than a beginner.

Programming problems are assigned to student in a scheduled course programming lab. This scheduled lab has a fixed duration of few hours. The solutions assigned in labs have to submitted within the schedule of respective labs.

While grading, TAs give benefit of some marks to solutions submitted earlier over other solution submissions. The earlier the submission made the higher the marks obtained. So, we establish a inverse relation between grades and submission timestamp. A solution submission timestamp refers to unix timestamp of the time of submission.

Time score function is an attempt to capture the grading policy regarding time of submission. We define this function as a linear function on submission timestamp, start timestamp and end timestamp of the submission's lab. For submission made at unix timestamp t_s and if lab start timestamp and end timestamp are t_{start} and t_{end} respectively, the time score function, f_t may be defined as follows.

$$f_t \in_{\mathbb{R}} [0, 1]$$

$$f_t(t_s, t_{start}, t_{end}) = \frac{(t_{end} - t_s)}{(t_{end} - t_{start})}$$

We use this time score function to apply the submission time based grading policy and use the feature of time taken to solve.

3.1.3 Fraction of Successful Compilations

Experienced programmers commit relatively less compilation errors as compared to beginners. They write elegant code in more organised manner. More organised manner includes proper indentation and relevant comments. TAs take into account indentation and comments directly. Using this feature of fraction of compilation errors or fraction of compilation successes, we attempt to capture the effect of indentation and comments on grades indirectly.

The lower the compilation errors the higher the grades. We introduce the compilation score function to apply the grading policy on indentation, comments and compilation errors. Whenever a students attempts to submit or evaluate/test their solution code on the online platform, the code is compiled and run against the test cases. The result of compilation is recorded which is success or fail. We extract compilation data and use it in our compilation score function.

For a submission having e_s compilation errors and e_{tot} being the total number of compilation attempts we define the compilation score function, f_e as follows.

$$f_e \in_{\mathbb{R}} [0, 1]$$

$$f_e(e_s, e_{tot}) = \begin{cases} 1 - \dfrac{e_s}{e_{tot}} & \text{if } e_{tot} > 0 \\ 0 & \text{otherwise} \end{cases}$$

We rely on this compilation score function to directly imply grading policy over fraction of successful compilations. Indirectly we attempt to apply grading policy on comments and indentation.

3.2 Initial Grading Model

In this section we describe our initial approach to the grading model. This approach is more inclined towards the intuitive idea. It is primitive and lays the foundation for our final grading model.

This model utilizes the feature set and the score functions introduced in previous section. We present initial grade score function on feature score functions. This is a multiple step approach.

The idea is to define a grade score function which is a weighted function of the feature score functions. The weights provide an implementation of flexible grading policy. Through this function we attempt to capture the intuitive idea behind the manual grading of programming assignments of the TAs.

The approach is as follows. We present 2 intermediate grade score functions, Hard Grade Score Function g_H and Soft Grade Score Function g_S. Finally we present the primitive function Initial Grade Score Function g_0, which combines both the intermediate hard and soft grade score function.

3.2.1 Hard Grade Score Function

As the name suggests hard grade score function, g_H, is over features that have relatively higher degree of objectivity and are easy to quantify. Also these features directly affect the grades awarded to submissions. It can be said that these features have less or no grey area for TAs to judge manually as compared to features that we will cover in soft grade score function.

Hard Grade Score Function is a weighted function over a subset of features of the feature set mentioned in previous section. We use the following feature grade functions in the hard grade score function. This function score is directly proportional to the grades awarded to the solution. It is a measure of correctness.

- Complete solution score function over fraction of TCs passed, f_c.

- Partial solution score function over fraction of TCs passed, f_p.

The hard grade score function is a linear weighted function of the 2 feature score functions listed above. The weights are real numbers in the range 0 to 1 inclusive. We know that the feature score functions also have a range of real numbers between 0 and 1 inclusive. The hard grade score function will also evaluate in the range of real number between 0 and 1 inclusive.

The hard grade score function for a submission/solution S is defined as.

$$g_H(S) = f_c(S) + w_p \times f_p(S)$$

where,

$g_H(S) \in_{\mathbb{R}} [0, 1]$, is the hard grade score for submission S.

$f_c(S) \in \{0, 1\}$, is the complete solution score function, for submission S.

$f_p(S) \in_{\mathbb{R}} [0, 1]$, is the partial solution score function, for submission S.

$w_p \in_{\mathbb{R}} [0, 1]$, is the weightage of partial solution.

Let us try to understand the intuition behind this function. Since complete score function f_c is an indicator function only one expression of the two sumed expression evaluates to non-zero value and other will evaluate to zero. Complete solution submissions are given full marks and the partial solution are given partial marks according to the partial solution weightage w_p. Since all the functions and weights have value between 0 and 1 inclusive, it is clear that the hard score function also lies between 0 and 1 inclusive.

Now, this hard grading score function can be written with full function domain variables.

$$g_H(k, n) = f_c(k, n) + w_p \times f_p(k, n)$$

where,

$k \in \mathbb{Z}_{\geq 0}$, is the number of test cases passed for the submission.

$n \in \mathbb{Z}_{\geq 0}$, is the total number of test cases for the submission.

Replacing the function with their expression we have.

$$g_H(k,n) = \left\lfloor \frac{k}{n} \right\rfloor + \left[\left(1 - \left\lfloor \frac{k}{n} \right\rfloor \right) \times w_p \times \frac{k}{n} \right]$$

This is the hard grade score function which will be used in our initial grade score function.

3.2.2 Soft Grade Score Function

Soft Grade Score Function, g_S is the counterpart of hard grade score function. It covers the remaining features that have relatively lower degree of objectivity and are relatively difficult to quantify. These features have higher manual TA judgement dependency because these features have more grey area.

Soft grade score function is also a weighted function over the remaining subset of features mentioned in previous section. It is also directly proportional to the grades and acts as a measure of programing soft skills. Following are the two features covered by soft grade score function.

- Time score function over time taken to solve, f_t
- Compilation score function over fraction of successful compilations, f_e

The soft grade score function is again an linear weighted function of the 2 feature score functions mentioned above. Similar to the hard grade score function, in this case also, the weights are real numbers with value between 0 and 1 inclusive. We know that the feature score function also evaluate in the real range between 0 and 1 inclusive. And we have designed the soft grade score function to evaluate in the range of 0 and 1 inclusive. For this we put an extra constraint on the weights of having sum equal to 1.

We define the soft grade score function for a submission/solution S as.

$$g_S(S) = w_t \times f_t(S) + (1 - w_t) \times f_e(S)$$

where,

$g_S(S) \in_{\mathbb{R}} [0, 1]$, is the soft grade score for submission S.

$f_t(S) \in_{\mathbb{R}} [0, 1]$, is the time score for submission S.

$f_e(S) \in_{\mathbb{R}} [0, 1]$, is the compilation score for submission S.

$w_t \in_{\mathbb{R}} [0, 1]$, is the weightage of time score.

$w_e = 1 - w_t$, is the weightage of compilation score.

The intuition behind this function is that we are just dividing the weightage among the two feature score functions. Since, these weightage values are variable the function can be customised for flexible grading policy. Since all the functions and variables evaluate to a value between 0 and 1 inclusive and we have a constraint of weightage values to have sum equal to 1, the soft grade score function is just a weighed average to feature score function. And since the feature score function have a evaluation range between 0 and 1 inclusive, the weighted average will also be between 0 and 1 inclusive. Hence, we also maintain the range of soft grade function between 0 and 1 inclusive.

Now, we rewrite the soft grade score function with full domain variables.

$$g_S(t_s, t_{start}, t_{end}, e_s, e_{tot}) = w_t \times f_t(t_s, t_{start}, t_{end}) + (1 - w_t) \times f_e(e_s, e_{tot})$$

where,

$g_S(S) \in_{\mathbb{R}} [0, 1]$, is the soft grade score for submission S.

$t_s \in \mathbb{Z}_{\geq 0}$, is the unix timestamp for the submission time.

$t_{start} \in \mathbb{Z}_{\geq 0}$, is the unix timestamp for the start time of lab/submission-window.

$t_{end} \in \mathbb{Z}_{\geq 0}$, is the unix timestamp for the end time of lab/submission-window.

$e_s \in \mathbb{Z}_{\geq 0}$, is the number of compilation errors made for the problem.

$e_{tot} \in \mathbb{Z}_{\geq 0}$, is the total number of compilation attempts made for the problem.

$w_e = 1 - w_t$, is a restriction imposed on the weights.

Now rewriting the above expression with expanded function definitions.

$$g_S(t_s, t_{start}, t_{end}, e_s, e_{max}) = w_t \times \frac{(t_{end} - t_s)}{(t_{end} - t_{start})} + w_e \times \left(1 - \frac{e_s}{e_{tot}}\right)$$

This definition of soft grade score function will be used to compute initial grade score function.

3.2.3 Initial Grade Score Function

We have discussed the two independent intermediate steps in our grading model in the form of hard and soft grade score functions. Combining the two grade functions, soft grade score function and hard grade score function, we present initial grade score function g_0. This function is the final step in our initial grading model.

Initial grade score function, g_0, aims to capture all the intended features that affects grades of a solution submission. It relies on the two intermediate grade score functions, hard and soft, which together capture all the features and grading policies followed by the TAs over these features. As both the score functions are directly proportional to grades, so is the initial grade score function. g_0 uses the following two functions.

- Hard grade score function, g_H.
- Soft grade score function, g_S.

The initial grade score function is also a linear function. More specifically, it is simply a weighted average of the two above grade score functions. We know that both the grading functions evaluate to a real number between 0 and 1 inclusive. We also restrict the weights w_H and w_S, real numbers in range 0 and 1 inclusive, to have sum equal to 1 as the function is weighted average. Hence this initial grade score function also evaluates to a real number between 0 and 1 inclusive.

We define the initial grading policy as follows for a submission S.

$$g_0(S) = w_H \times g_H(S) + (1 - w_H) \times g_S(S)$$

where,

$g_0(S) \in_{\mathbb{R}} [0, 1]$, is the initial grade score function for solution S.

$g_H(S) \in_{\mathbb{R}} [0, 1]$, is the hard grade score function for solution S.

$g_S(S) \in_{\mathbb{R}} [0, 1]$, is the soft grade score function for solution S.

$w_H \in_{\mathbb{R}} [0, 1]$, is the weightage of g_H.

$w_S = 1 - w_H$, is the weightage of g_S.

The intuition is that we are just dividing the variable weightage of the initial grade score function among the hard grade score function and soft grade score function. This provides provisions for flexible grading policy with customizable weight distribution over hard features and soft features. Since, the 2 intermediate score functions represent different type of features, the degree of impact of each type of feature set can be controlled. This granularity helps us to implement the grading policy similar to that implemented manually by the TAs.

Now, rewriting the function with all the domain variables.

$$g_0(k, n, t_s, t_{start}, t_{end}, e_s, e_{tot}) = w_H \times g_H(k, n) + w_S \times g_S(t_s, t_{start}, t_{end}, e_s, e_{tot})$$

where,

$g_0 \in_{\mathbb{R}} [0, 1]$, is the initial grade score function.

$g_H \in_{\mathbb{R}} [0, 1]$, is the hard grade score function.

$g_S \in_{\mathbb{R}} [0, 1]$, is the soft grade score function.

$w_H \in_{\mathbb{R}} [0, 1]$, is the weightage of g_H.

$w_S \in_{\mathbb{R}} [0, 1]$, is the weightage of g_S.

$w_S = 1 - w_H$, is a restriction imposed on the weights.

$k \in \mathbb{Z}_{\geq 0}$, is the number of test cases passed for the submission.

$n \in \mathbb{Z}_{\geq 0}$, is the total number of test cases for the submission.

$t_s \in \mathbb{Z}_{\geq 0}$, is the unix timestamp for the submission time.

$t_{start} \in \mathbb{Z}_{\geq 0}$, is the unix timestamp for the start time of lab/submission-window.

$t_{end} \in \mathbb{Z}_{\geq 0}$, is the unix timestamp for the end time of lab/submission-window.

$e_s \in \mathbb{Z}_{\geq 0}$, is the number of compilation errors made for the problem assigned.

$e_{tot} \in \mathbb{Z}_{\geq 0}$, is the total number of compilation attempts made for the problem

assigned.

Thus we have derived our Initial Grade Score Function. This function attempts to capture different grading policies depending upon the weights.

The initial grade score will compute a real number value between 0 and 1 inclusive when provided the required features of a submission. This initial grade score for a submission is directly proportional to the grades awarded by our model. Higher the initial grade score, g_0, higher the grades/marks awarded by our grading model. If for a submission S, g_0 is 1.0 the grades/marks awarded by our grading model it equal to the maximum marks, mm, and if g_0 is 0.0 then the grades/marks is equal to minimum marks, i.e., 0, in many grading systems.

For a submission S we have all the required features extracted, we know the procedure to compute its initial grade score function $g_0(S)$, we also know the maximum marks for the problem set. Then we can use the following relation to get the grades/marks awarded to submission S by our initial grading model.

$$marks(S) = g_0(S) \times mm(S)$$

where,

$marks(S) \in_{\mathbb{R}} [0, mm(S)]$, is the marks/grades awarded by our initial grading model to solution S.

$g_0(S) \in_{\mathbb{R}} [0, 1]$ is the initial grade score computed by the function for submission S.

$mm(S) \in \mathbb{Z}_{\geq 0}$, is the maximum marks set for the problem for submission S.

Thus, we have a model/tool namely initial grading model for solution submissions for problems assigned in an introductory programming course.

3.3 Final Grading Model

In this section we finally present the Final Grading Model which is similar to and is an extension of Initial Grading Model. Basically, it is a refinement of Initial Grading Model.

We first discuss the significance and motivation for extending the initial grading model to final grading model. Following it we intuitively introduce the final grade score function. Lastly we summarise the final grading model and its usage. This wraps up the core theory of the thesis.

3.3.1 Significance and Motivation

Initial grading model consists of multiple steps and 5 weight variables. g_0 is directly motivated by the grading policies set in the introductory programming course and by TAs' judgement. It can be said that the initial grading model follows the approach more aligned to intuition and could be simplified further.

Final grading model aims to simplify the initial grading model. Through the final grading model we attempt to reduce number of steps and number of weightage variables. It need not be intuitive.

Reducing the model to a simpler form has various advantages. Less number of steps allows easier computation and implementation of the model, making it more analysis friendly. Less number of weightage variable reduces the complexity of determining the correct values for these variables. Evolution of simplified model is more efficient as compared to a complex one. These little advantages prove very significant for our work when we try to improve the model or develop a variant of it and also help when we try to analyse the performance of our model or try to find most appropriate values for the weightage variables.

3.3.2 Intuition

Now we present the final grading model and final grade score function and intuition behind it. We explain the changes made to initial grade function that refines it to final grade score function.

There are two major modifications in the new grade function.

1. First modification is on partial solution grade score. We have removed the indicator part of the function. The new definition of partial grade score function

is as follows.

$$f_p(k, n) = \frac{k}{n}$$

where,

$k \in \mathbb{Z}_{\geq 0}$, is the number of test cases passed for the submission.

$n \in \mathbb{Z}_{\geq 0}$, is the total number of test cases for the submission.

The intuitive purpose is to make the function simpler. It does not change the grading model. By simple adjustment of the weights the former model can be retained.

2. Second modification is on compilation grade score function. The new function is also a linear function with slight difference in its definition. One of the parameters has also changed. Total number of compile attempts by the attempting student, e_{tot}, gets replaced by maximum number of compilation failures by any student attempting the same problem, e_{max}. Number of compilation failures made by the student for the submission, e_s, will remain of use. Now, the compilation grade score function is inversely proportional to ratio of e_s to e_{max}. Initally the score was inversly proportional to the ratio of e_s to e_{tot}. The new definition of compilation grade score function is as follows.

$$f_e(e_s, e_{max}) = \begin{cases} 1 - \dfrac{e_s}{e_{max}} & \text{if } e_{max} > 0 \\ 0 & \text{otherwise} \end{cases}$$

where,

$e_s \in \mathbb{Z}_{\geq 0}$, is the number of compilation errors made.

$e_{max} \in \mathbb{Z}_{\geq 0}$ is the maximum number of compilation errors made by any student for the same problem.

The previous/former compilation score function is unfair to students having low number of compilation attempts and is independent of other students. We need a function which can relatively distinguish students' solution submissions on the basis of compilation results.

Consider three students A, B and C that attempted a common problem. The compilation attempts and compilation failures for A are 2 and 1 respectively. For B these values are 10 and 4 respectively and for C it's 20 and 0 respectively.

So if former compilation grade score function is used to compare the students we have, $B > A > C$. Higher score is better and so B outscores A. But A had only 1 compilation failure and B had 6 compilation failures. More over the scores of A and B are independent of compilation results of C.

If the new compilation grade score function is use to compare then we have, $A > B > C$. This relative score is more intuitive as A has made lower number of compilation errors as compared to B. Also these scores are relative to a common student C, that made maximum number of compilation failures attempting the same problem.

We will completely expand the initial grade score function according to the above new intermediate feature functions. Then we will reduce it to a mathematically simpler form. Finally we restructure and rename variables for simplicity, to present the final grade score function.

The initial grade score function is as follows.

$$g_0(k, n, t_s, t_{start}, t_{end}, e_s, e_{max}) = w_H \times g_H(k, n) + (1 - w_H) \times g_S(t_s, t_{start}, t_{end}, e_s, e_{max})$$

where,

$g_0 \in_{\mathbb{R}} [0, 1]$, is the initial grade score function.

$g_H \in_{\mathbb{R}} [0, 1]$, is the hard grade score function.

$g_S \in_{\mathbb{R}} [0, 1]$, is the soft grade score function.

$w_H \in_{\mathbb{R}} [0, 1]$, is the weightage of g_H.

$w_S = 1 - w_H$, is the weightage of g_S.

$k \in \mathbb{Z}_{\geq 0}$, is the number of test cases passed for the submission.

$n \in \mathbb{Z}_{\geq 0}$, is the total number of test cases for the submission.

$t_s \in \mathbb{Z}_{\geq 0}$, is the unix timestamp for the submission time.

$t_{start} \in \mathbb{Z}_{\geq 0}$, is the unix timestamp for the start time of lab/submission-window.

$t_{end} \in \mathbb{Z}_{\geq 0}$, is the unix timestamp for the end time of lab/submission-window.

$e_s \in \mathbb{Z}_{\geq 0}$, is the number of compilation errors made for the problem assigned.

$e_{max} \in \mathbb{Z}_{\geq 0}$, is the maximum number of compilation error made by any student attempting the same problem.

$$g_0(k, n, t_s, t_{start}, t_{end}, e_s, e_{max}) = w_H \times g_H(k, n) + w_S \times g_S(t_s, t_{start}, t_{end}, e_s, e_{max})$$

We know that,

$g_H(k, n) = \lfloor \frac{k}{n} \rfloor + w_p \times \frac{k}{n}$

$g_S(t_s, t_{start}, t_{end}, e_s, e_{max}) = w_t \times \frac{(t_{end} - t_s)}{(t_{end} - t_{start})} + w_e \times \left(1 - \frac{e_s}{e_{max}}\right)$

Substituting the above function definitions in the expression of g_0, we have,

$$= w_H \times \left[\left\lfloor \frac{k}{n} \right\rfloor + w_p \times \frac{k}{n} \right] + w_S \times \left[w_t \times \frac{(t_{end} - t_s)}{(t_{end} - t_{start})} + w_e \times \left(1 - \frac{e_s}{e_{max}}\right) \right]$$

$$= w_H \left\lfloor \frac{k}{n} \right\rfloor + w_H w_p \frac{k}{n} + w_S w_t \frac{(t_{end} - t_s)}{(t_{end} - t_{start})} + w_S w_e \left(1 - \frac{e_s}{e_{max}}\right)$$

Now renaming the constant variables without loss of generality as per following.

$w_H = w_c$,

$w_H \times w_p = w_p$,

$w_S \times w_t = w_t$,

$w_S \times w_e = w_e$

Renaming is done for simplicity and to keep the same weight parameter names. We have derived a new function over same set of input parameters except for e_{tot}. e_{tot} has been replaced by e_{max}. A constraints on weight parameters, $w_c + w_p + w_t + w_e = 1$,

is imposed to fulfil the weight parameter constraints on inital grade score fucntion.

$$= w_c \left\lfloor \frac{k}{n} \right\rfloor + w_p \frac{k}{n} + w_t \frac{(t_{end} - t_s)}{(t_{end} - t_{start})} + w_e \left(1 - \frac{e_s}{e_{max}}\right)$$

Hence we have derived a new grade score function from initial grade score function named final grade score function.

3.3.3 Final Grade Score Function

In the previous section we have derived and presented the final grade score function g which now uses modified feature functions and weightage variables. We again state the final grade score function clearly.

The final grade score function is a single step grade score function. It takes in 7 feature variables as input to computes a score which is real number in the range 0 and 1 inclusive. We impose the restriction on the weight to have sum equal to 1. This restriction allows us to maintain the range of scores to lie between 0 and 1, both inclusive.

$$g(S) = w_c \times f_c(S) + w_p \times f_p(S) + w_t \times f_t(S) + w_e \times f_e(S)$$

where,

$g(S) \in_{\mathbb{R}} [0, 1]$, is the final grade score function for solution S.

$f_c(S) \in_{\mathbb{R}} [0, 1]$, is the complete solution score function for solution S.

$f_p(S) \in_{\mathbb{R}} [0, 1]$, is the partial solution score function for solution S.

$f_t(S) \in_{\mathbb{R}} [0, 1]$, is the time feature score function for solution S.

$f_e(S) \in_{\mathbb{R}} [0, 1]$, is the compilation feature score function for solution S.

$w_c \in_{\mathbb{R}} [0, 1]$, is the weightage variable for complete score.

$w_p \in_{\mathbb{R}} [0, 1]$, is the weightage variable for partial score.

$w_t \in_{\mathbb{R}} [0, 1]$, is the weightage variable for time score.

$w_e \in_{\mathbb{R}} [0, 1]$, is the weightage variable for compilation score.

$w_c + w_p + w_t + w_e = 1$, is restriction on the weights to have sum equal to 1.

Now, writing the function with its domain variables or input parameters which are the features for a submission.

$$g(k, n, t_s, t_{start}, t_{end}, e_s, e_{max}) = w_c f_c(k, n) + w_p f_p(k, n) + w_t f_t(t_s, t_{start}, t_{end}) + w_e f_e(e_s, e_{max})$$

where,

$k \in \mathbb{Z}_{\geq 0}$, is the number of test cases passed for the submission.

$n \in \mathbb{Z}_{\geq 0}$, is the total number of test cases for the submission.

$t_s \in \mathbb{Z}_{\geq 0}$, is the unix timestamp for the submission time.

$t_{start} \in \mathbb{Z}_{\geq 0}$, is the unix timestamp for the start time of lab/submission-window.

$t_{end} \in \mathbb{Z}_{\geq 0}$, is the unix timestamp for the end time of lab/submission-window.

$e_s \in \mathbb{Z}_{\geq 0}$, is the number of compilation errors made for the problem assigned.

$e_{max} \in \mathbb{Z}_{\geq 0}$, is the maximum number of compilation error made by any student attempting the same problem.

Replacing the functions with their definitions we have.

$$g(k, n, t_s, t_{start}, t_{end}, e_s, e_{max}) = w_c \left\lfloor \frac{k}{n} \right\rfloor + w_p \frac{k}{n} + w_t \frac{(t_{end} - t_s)}{(t_{end} - t_{start})} + w_e \left(1 - \frac{e_s}{e_{max}}\right)$$

The function is again a weighted average of all the feature score function but in a single step in this case. All the weights are restricted to have a value between 0 and 1 inclusive and all the feature score functions evaluate to a value between 0 and 1 inclusive. As the function is a weighted average we restrict the weights to have a sum of 1.Thus, the function also evaluates to a real number between 0 and 1 inclusive.

The final function like the initial function also uses 4 feature functions but 4 weight parameters instead of 5. The new grade score function provides a single step grading model. Although there might be some difference in the granularity of policies applied through these weights but this function is easier to analyse and work with weights.

Obtaining marks form the final grade score is pretty straightforward and similar to that of initial grade score function. Since the final grade score function evaluates

between 0 and 1 inclusive we just take the product of final score and maximum marks set for the problem. The relation is as follows.

$$marks(S) = g(S) \times mm(S)$$

where,

$marks(S) \in_{\mathbb{R}} [0, mm(S)]$, is the marks/grades awarded by the final grading model to solution S.

$g(S) \in_{\mathbb{R}} [0, 1]$, is the final grade score computed by the function for submission S.

$mm(S) \in \mathbb{Z}_{\geq 0}$, is the maximum marks set for the problem being attempted in submission S.

Hence, we have derived a function and a grading model described above. For any solution submission if the features are availabe it can be graded according to grading policy set by weight parameter values. This model can be used with weights to implement a wide variety of grading policy to estimate grades for programming problem solution submission is a introductory programming course.

Chapter 4

Learning Weight Parameters for Grading Model

So far we have introduced our work, helped develop the background required to understand it and briefly mentioned related work. The last chapter explained the grading model and its theoretical foundations. The final grading function and grading model are implemented through out the remaining part of this thesis.

However, the final grade score function cannot be implemented without assigning values to the weightage variables/parameters. This chapter describes the attempt to solves this problem of assigning values to the weight parameters.

In this chapter we explain the supervised training performed to set the values of the weight parameters of the final grade function. The learned weights are presented. These weight values are used in the remainder part of our thesis.

4.1 Learning Technique and Training Dataset

We have developed a function, final grade function, which is a linear function. And we need to assign values to the weight parameters. Our strategy is to learn these weights from the grading done by the teaching assistants (TAs).

The training data is Exam Grades Dataset described in Section 2.2.2. The dataset is briefly but completely described for convinience as follows.

The dataset was collect by ITS for ESc101, an introductory programming course at IIT Kanpur in 2014-15 even semester. It consists of sets of 201 solution submissions each. There are 8 such sets and each set is for different question. We have the final awarded grades in the dataset and the features required by the final grade score function can be extracted. This data is our training dataset as the grades have awarded with extreme caution and cross checked multiple times. Thus, it can be said that our training dataset has, $n = 1608$, data points to perform a supervised learning.

Since final grade function is a linear function the training technique used is Linear Regression with Ordinary Least Squares [MPV12] introduced in section 2.3. Details are discussed in this section.

Now, we need a linear regression model on our linear final grade function with the exam grade dataset. We present the linear regression model in the remainder of this section.

Consider the final grade score function.

$$g(k, n, t_s, t_{start}, t_{end}, e_s, e_{max}) = w_c f_c(k, n) + w_p f_p(k, n) + w_t f_t(t_s, t_{start}, t_{end}) + w_e f_e(e_s, e_{max})$$

The definitions have been described in previous chapter more than once. More over the functions may be treated just as a black box for our purpose. The aim is to learn over the training dataset to estimate the values of w_c, w_p, w_t and w_e.

The technique we intend to use is Linear Regression with Ordinary Least Squares which can estimate these 4 weight parameters w_c, w_p, w_t and w_e if f_c, f_p, f_t and f_e are known for all the data points. And g can be obtained by dividing the grades awarded by maximum marks. The feature function f_c, f_p, f_t and f_e can be evaluated if all the 7 features for each data point are known namely $k, n, t_s, t_{start}, t_{end}, e_s$ and e_{max}.

We had access to all the logged data and python scripts were written to mine the values of these features. We also have the grades awarded to all these submissions and the maximum marks for these submissions' problems.

We have successfully described a linear regression model for our and also have fulfilled all the requirements for learning the weights. Hence, we are set to run linear regression with ordinary least squares over our gold standard data set to estimate the values of weight variables.

4.2 Learned Weights

In this section we present the results of training performed as explained in the previous section and state the learned weightage parameter. Since, we have describe the linear regression model we need to fill in the details and run the ordinary least squares method to get an estimate of weights.

For each of the 1608 data points, i.e. submissions in our data set we have mined the following 7 features from the database containing the detailed logs of lab exam data set.

1. k^i, number of testcases passed for i^{th} submission.
2. n^i, total number of testcases for the problem for i^{th} submission.
3. t_s^i, submission timestamp for i^{th} submission.
4. t_{start}^i, start timestamp of lab exam for i^{th} submission.
5. t_{end}^i, end timestamp of lab exam for i^{th} submission.
6. e_s^i, number of compilation fails for i^{th} submission.
7. e_{max}^i, maximum number of compilation fails for any submission attempting the problem for i^{th} submission.
8. $marks^i$, grades/marks awarded to the i^{th} submission.
9. mm^i, maximum marks for i^{th} submission's problem.

Using these mined features we can calculate the following 4 feature function scores for all the 1608 data points.

1. $f_c^i(k^i, n^i)$, complete solution score function.

2. $f_p^i(k^i, n^i)$, partial solution score function.

3. $f_t^i(t_s^i, t_{start}^i, t_{end}^i)$, time score function.

4. $f_e^i(e_s^i, e_{max}^i)$, compilation score function.

Now, the final grade score g for each corresponding data point can simply be obtained by dividing the grades awarded by TAs $marks^i$ by the maximum marks mm^i of the problem of the submission.

$$g^i(k, n, t_s, t_{start}, t_{end}, e_s, e_{max}) = \frac{marks^i}{mm^i}$$

where,
g^i is the score for i^{th} data point. $marks^i$ is the grades awarded by the TAs for i^{th} data point in our exam grades dataset. mm^i is the maximum marks for i^{th} data point's problem in the exam grades dataset.

So, the direct closed form expression to estimate the weightage parameters is.

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

So, now we just need to define the matrices to get the weightage parameters. The matrices are as follows.

$$y = \begin{bmatrix} g^1 \\ g^2 \\ \vdots \\ g^n \end{bmatrix}, \hat{\beta} = \begin{bmatrix} \hat{w_c} \\ \hat{w_p} \\ \hat{w_t} \\ \hat{w_e} \end{bmatrix}, X = \begin{bmatrix} f_c^1 & f_p^1 & f_t^1 & f_e^1 \\ f_c^2 & f_p^2 & f_t^2 & f_e^2 \\ \vdots & \vdots & \vdots & \vdots \\ f_c^n & f_p^n & f_t^n & f_e^n \end{bmatrix}$$

After evaluating we have,

$$\hat{\beta} = \begin{bmatrix} 0.04663081 \\ 0.78131624 \\ 0.05072156 \\ 0.15732873 \end{bmatrix}$$

Parameter	Value
$\hat{w_c}$	0.04663081
$\hat{w_p}$	0.78131624
$\hat{w_t}$	0.05072156
$\hat{w_e}$	0.15732873

Table 4.1: Estimated Weight Parameters for Final Grade Score Function

Thus we have estimated a set of values for the weightage parameters for final grade score function. We will use these parameter values through out the remainder of our thesis unless stated otherwise.

The grading model imposes an restriction on the sum of the weight parameters to be 1. However, the sum of estimated parameters is 1.03599734, 0.04 higher. This does not indicate any error or inconsistency in the estimation of these parameters. The parameter estimation technique does not take into account any such restrictions. For the purpose of grading all final grade scores above 1 will be considered as 1.

Chapter 5

Experiments and Results

The final grading model along with the learned weights is a grading model ready for application. But the correctness and accuracy is yet to understood.

Understanding correctness and accuracy needs a scale of measurement, something to compare with. We have used the teaching assistant (TA) Grade Data Set. This dataset had 2 grades for each assignment by different teaching assistants (TAs), one finally awarded to the submission and one discarded or alternate TA grade. We have compared the alternate TA grades against our grading model grades. These comparisons are done keeping the final grades as benchmark.

Once we have a scale we need a gauge to quantify and compare, some metrics that are appropriate for its intended purpose. We have considered the fact that most important aspect of grading it that it brings order, a means to rank. We have experimented with metrics that compare the ranking with the benchmark ranking to quantify performance on different basis. There are multiple solution submission sets as per programming question/problem with 201 submissions in each TA submission set and 99 to 101 submissions in each grading tool submission set.

Using the quantified measurements we present the observations and compare the performance of alternate TA grades against grading model tool grades various graph plots and tables.

The datasets used for experiments are described in Section 2.2. Analysis of our autograding tool is done with the dataset explained in section 2.2.1, assignment

grades dataset and analysis of grading by TAs is done with dataset explained in section 2.2.2, exam grades dataset. These datasets are briefly mentioned again.

Both the datasets, exam grades and assignment grades datasets, were documented during ESc101 course, an introductory programming course, at IIT Kanpur held in even semester of academic year 2014-2015.

Exam grades dataset consist of solution submissions for 8 programming questions/problems. Each question has 201 corresponding submissions, forming 8 solution submission sets each consisting of 201 solution submissions. Each solution submission has been graded by 2 different TAs. One of them is finally awarded and other has been discarded as alternate TA grade. The alternate TA grades are analysed by keeping the awarded grades as true benchmark grades. Please note that these awarded grades are also used for learning weight parameters.

Assignment grades dataset consist of solution submissions for 64 programming questions/problems. Each question has 99 to 101 corresponding submissions, forming 64 solution submission sets each consisting of 99 to 101 solution submissions. Each solution submission has been graded by a TA and our autograding tool. The grades by TAs are the finally awarded grades to these soultion submissions. Our autograding tool grades are analysed by keeping the awarded grades as true benchmark grades.

5.1 Grade Error

The Absolute Mean Error [Inca] and Root Mean Squared Error [Incb] gives us some basic level idea of performance comparison.

Absolute Mean Error (Abs Mean Err) is the mean of absolute errors for all predicted/estimated values. Root Mean Square Error (RMSE) is the root of mean of squared errors for all predicted/estimated values. They may be defined as.

$$\text{Abs Mean Err} = \frac{1}{n}\sum_{i=1}^{n} |\textit{predicted-value}_i - \textit{true-value}_i|$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\textit{predicted-value}_i - \textit{true-value}_i)^2}$$

where,

n, is the total number of sample points.

predicted-value$_i$, is the value predicted/estimated by the model for i^{th} sample point.

true-value$_i$, is the actual value for i^{th} data point.

These error calculations for the model/tool use the final grade score as the *predicted-value* and normalised awarded grades as the *true-value*. The normalised marks by alternate TA is used as *predicted-value* and normalised awarded marks as *true-value* for TA error calcualtion.

Table below compares Grading Model/Tool normalised grades versus Alternate TA normalised grades on bases of Absolute Mean Error (Abs Mean Err) and Root Mean Squared Error (RMSE) against awarded grades. These errors are calculated for normalised grades, grades scaled to real number in range 0 and 1, both inclusive. For each question there is a submissions set. For each submission set both absolute mean and root mean squared errors are calculated on normalised grades. For these values average, standard deviation, minimum and maximum are mention in the table below.

	Grading Model/Tool				Alternate TA			
	Avg.	S.D.	Min.	Max.	Avg.	S.D.	Min.	Max.
Abs Mean Err	0.0700	0.0635	0.0110	0.3960	0.1554	0.0864	0.0872	0.2920
RMSE	0.1291	0.0900	0.0305	0.5775	0.2540	0.1281	0.1521	0.4373

Table 5.1: Avg. Grade Error Comparison - Grading Tool vs TAs

The numbers suggests that Grading Model/Tool performs comparable to TAs in terms or errors. In almost all comparisons in the table Grading Model/Tool displays marginally lower errors.

5.2 Rank Error

Rank for a solution submission s_i is defined as its position in its corresponding solution set D sorted in descending order by grades/marks. True rank $rank_{true}(s_i)$, tool rank $rank_{tool}(s_i)$ and TA rank $rank_{TA}(s_i)$ of a submissions correspond to rank when sorting is done by true marks $marks_{true}(s_i)$, tool marks $marks_{tool}(s_i)$ and TA marks $marks_{TA}(s_i)$, respectively.

If there are submissions in the solution set D having grades/marks equal to a ranked submission s_i then the ranked submission has a bucket/set of ranks $b\text{-}rank(s_i)$ containing ranks of all these submissions.

$$b\text{-}rank_{true}(s_i) = \{rank_{true}(s_j) \mid marks_{true}(s_i) = marks_{true}(s_j); \text{ and } s_j \in D\}$$

$$b\text{-}rank_{tool}(s_i) = \{rank_{tool}(s_j) \mid marks_{tool}(s_i) = marks_{tool}(s_j); \text{ and } s_j \in D\}$$

$$b\text{-}rank_{TA}(s_i) = \{rank_{TA}(s_j) \mid marks_{TA}(s_i) = marks_{TA}(s_j); \text{ and } s_j \in D\}$$

Rank error $rank\text{-}error(s_i)$ for a solution submissions s_i is definied as minimum absolute difference between its true ranks and tool or TA ranks. Please note that a submissions will either have tool ranks or TA ranks since the datasets are disjoint.

$$rank\text{-}error_{tool}(s_i) = \min_{r_{tool \in b\text{-}rank_{tool}(s_i)}\ ,\ r_{true \in b\text{-}rank_{true}(s_i)}} (|r_{tool} - r_{true}|)$$

$$rank\text{-}error_{TA}(s_i) = \min_{r_{TA \in b\text{-}rank_{TA}(s_i)}\ ,\ r_{true \in b\text{-}rank_{true}(s_i)}} (|r_{TA} - r_{true}|)$$

Number of rank errors $num\text{-}rank\text{-}error\text{-}ss(D, j)$ for a submission set D with rank displacement less than equal to r is defined as.

$$num\text{-}rank\text{-}error\text{-}ss_{tool}(D, r) = \mid \{s_i \mid rank\text{-}error_{tool}(s_i) \leq r; \text{ and } s_i \in D\} \mid$$

$$num\text{-}rank\text{-}error\text{-}ss_{TA}(D, r) = \mid \{s_i \mid rank\text{-}error_{TA}(s_i) \leq r; \text{ and } s_i \in D\} \mid$$

The following graph on y-axis, number of rank errors *num-rank-error*(r), denotes the average number of submissions across all the solution sets having rank error less than or equal to the x-axis rank r. Let m_{tool} be total number of tool submission sets available and let m_{TA} be the total number of TA submission sets available.

$$\textit{num-rank-error}_{tool}(r) = \frac{1}{m_{tool}} \sum_{i=1}^{m_{tool}} \textit{num-of-rank-error-ss}_{tool}(D_i, r)$$

$$\textit{num-rank-error}_{TA}(r) = \frac{1}{m_{TA}} \sum_{i=1}^{m_{TA}} \textit{num-of-rank-error-ss}_{TA}(D_i, r)$$

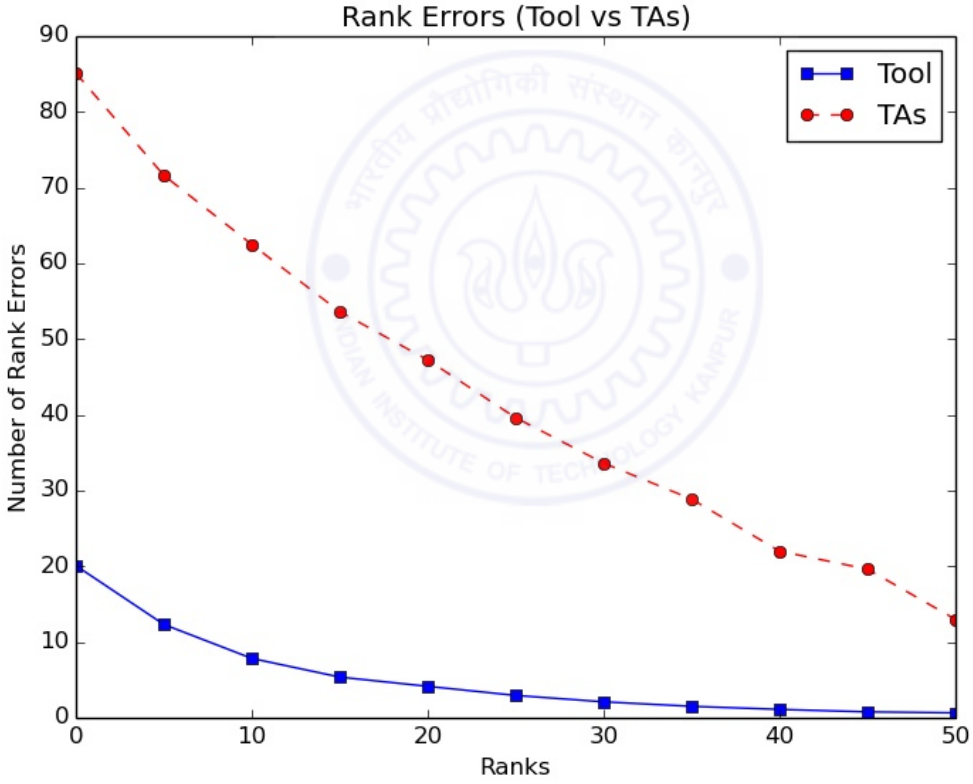


Figure 5.1: Avg. Rank Error Comparison - Grading Tool vs TAs

Although a single submission set for a question consists of 201 submissions for TA data set and 99 to 101 submissions for tool dataset our autograding tool exhibits lower rank errors as compared to TAs.

Welch's t-test [Rum07] is statistical test for the null hypothesis that 2 independent samples have identical average (expected) values. On performing t-test for average

number of rank errors comparing means of tool and TAs, the *p-value* = 0.0002258064. The low *p-value* implies that the null hypothesis of tool and TA having identical means may be rejected. Hence, the differences in rank errors are statistically significant.

Following two sections helps us to understand the performance in terms of ranking at a little more finer level of granularity. Some statistics for positive rank displacement, improvement in rank with respect to its rank according to awarded grades and for negative rank displacement, deterioration in rank with respect to its rank according to awarded grades.

5.2.1 Positive Rank Error

Taken into consideration here are rank errors only for submissions that exibited improvement in their ranks, i.e., TA or tool ranks are better than true ranks. Following is a table with few such statistics comparing our autograding tool and TAs.

- Avg. Number of Errors: Denotes the average number of solutions exhibiting positive rank error. It is calculated by taking average of, number of submissions exhibiting improvement in rank for a submission set, across all the submission sets.
- Average Rank Error: Denotes the average magnitude of positive rank error, in ranks. It is calculated by taking average of, average magnitude of rank improvements in a submissions set, across all the submission sets.
- Maximum Rank Error: Denotes the maximum magnitude of positive rank error, in ranks. It is calculated by taking the average of, maximum improvement in rank in a submissions set, across all the submission sets.
- S.D. of Rank Error: Denotes the standard deviation of magnitude of positive rank error, in ranks. It is calculated by taking the average of, standard deviations of magnitude of improvement in rank for a submission set, across all the submission sets.

	Grading Tool	TAs
Avg. Number of Errors	9.73	44.12
Average Rank Error	18.04	31.17
Maximum Rank Error	35.10	83.25
S.D. of Rank Error	10.60	21.94

Table 5.2: Positive Rank Error - Grading Tool vs TAs

Even considering that TA dataset has 201 submissions for each question and Tool data set has about 99 to 101 solutions for each question, the errors for tool are lower or equal to the TAs.

Welch's t-test [Rum07] is statistical test for the null hypothesis that 2 independent samples have identical average (expected) values. On performing t-test for average number of positive rank errors comparing means of tool and TAs, the *p-value* = 0.0013670157. The low *p-value* implies that the null hypothesis of tool and TA having identical means may be rejected. Hence, the differences in positive rank errors are statistically significant.

5.2.2 Negative Rank Error

Now, considering rank errors only for submissions that exibited deterioration in their ranks, i.e., TA or tool ranks are worse than true ranks. Following is a table with few such statistics comparing our autograding tool and TAs.

- Avg. Number of Errors: Denotes the average number of solutions exhibiting negative rank error. It is calculated by taking average of, number of submissions exhibiting deterioration in rank for a submission set, across all the submission sets.

- Average Rank Error: Denotes the average magnitude of negative rank error, in ranks. It is calculated by taking average of, average magnitude of rank deteriorations in a submissions set, across all the submission sets.

- Maximum Rank Error: Denotes the maximum magnitude of negative rank error, in ranks. It is calculated by taking the average of, maximum deterioration in rank in a submissions set, across all the submission sets.

- S.D. of Rank Error: Denotes the standard deviation of magnitude of negative rank error, in ranks. It is calculated by taking the average of, standard deviations of magnitude of deterioration in rank for a submission set, across all the submission sets.

	Grading Tool	TAs
Avg. Number of Errors	10.35	41
Average Rank Error	5.51	22.90
Maximum Rank Error	13.73	61.25
S.D. of Rank Error	3.68	15.29

Table 5.3: Negative Rank Error - Grading Tool vs TAs

Grading tool displays much lower rank errors as compared to the TAs.

As we have mentioned Welch's t-test [Rum07], which is a statistical test for the null hypothesis that 2 independent samples have identical average (expected) values. Again, performing t-test for average number of negative rank errors comparing means for tool and TAs, the *p-value* = 0.000792847. The *p-value* is again quite low. Since it implies that the null hypothesis of identical may be rejected. Hence, these differences are also statistically significant.

5.3 Spearman's Rank Correlation Coefficient (ρ)

Performance in terms of ranking can be measured by calculating the Spearman's Rank Correlation Coefficient (ρ) [WM03].

Spearman's rank correlation coefficient is a measure of similarity of order or sequence between two different arrangements by two different variables on a common set of items. It is calculated by the following formula.

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

where,

n, is the number of items.

d_i, is the difference in rank/position of i^{th} item in two different arrangements by the two variables.

Spearman's rank correlation coefficients for ordering by both autograding tool and TAs are calculated with respect to the ordering by their corresponding true grades.

A table comparing autograding tool and TAs on different statistics of Spearman's Rank Correlation Coefficient (ρ) is presented. Spearman's rank correlation coefficient is calculated for every submission set. Then for these values average, minimum, maximum and standard deviation are calculated.

	Grading Tool	TAs
Average	0.9559	0.9163
Maximum	1.0000	0.9900
Minimum	0.6930	0.8280
Standard Deviation	0.0502	0.5416

Table 5.4: Spearman's Rank Correlation Coefficient - Grading Tool vs TAs

ρ has a real number range from -1 to $+1$. The higher the value of ρ the more similarity with the ranking by awarded grades. The autograding tool outperformances the TAs in all measures except for minimum value of ρ. Since, standard deviation is lower and average is higher it cannot be concluded that tool is inferior to TAs.

As we have mentioned t-test [Rum07], which is a statistical test for the null hypothesis that 2 independent samples have identical average (expected) values. Again, performing t-test for the ρ values of tool and TAs, the *p-value* $= 0.1005588579$. This *p-value* is low to a statistical significance degree of 10%. So, the null hypothesis of identical means may be rejected. Hence, the differences are statistically significant

for ρ also.

5.4 Mean Average Precision (MAP)

Precision [MRS08] measures accuracy for queries that are responded with a set of item. It is defined as follows.

$$precision = \frac{|\{relevant\text{-}items\} \cap \{retrieved\text{-}items\}|}{|\{retrieved\text{-}items\}|}$$

where,

relevant-items, are the expected items.

retrieved-items, are the items returned by the model.

Average Precision ($AveP$) [MRS08] for a query of n items is defined as follows.

$$AveP = \frac{\sum_{k=1}^{n}[P(k) \times rel(k)]}{n}$$

where,

n is the number of items queried for,

$P(k)$ is the precision for k items,

$rel(k)$ is an indicator function equal to 1 if item at rank k is relevant and 0 otherwise.

Mean Average Precision (MAP) [MRS08] extends $AveP$ for multiple queries and is defined as follows.

$$\text{MAP} = \frac{\sum_{q=1}^{Q} AveP(q)}{Q}$$

where,

Q is the number of queries under consideration,

$AveP(q)$ is the Average Precision for q^{th} query.

Precision uses tool grades and TAs grades for *retrieved-items* and true grades for *relevant-items*. Items for our purpose are submissions. Mean Average Precision (MAP) is calculated for two types of queries, retrieving top-k ranked and retrieving bottom-k ranked submissions in a submission set. MAP is calcualted for queries for

each submission set. The graph plots have k on x-axis and calculated MAP for top-k or bottom-k queries on y-axis.

5.4.1 Top-k Query MAP

For top-k queries the plot k versus MAP values follows.

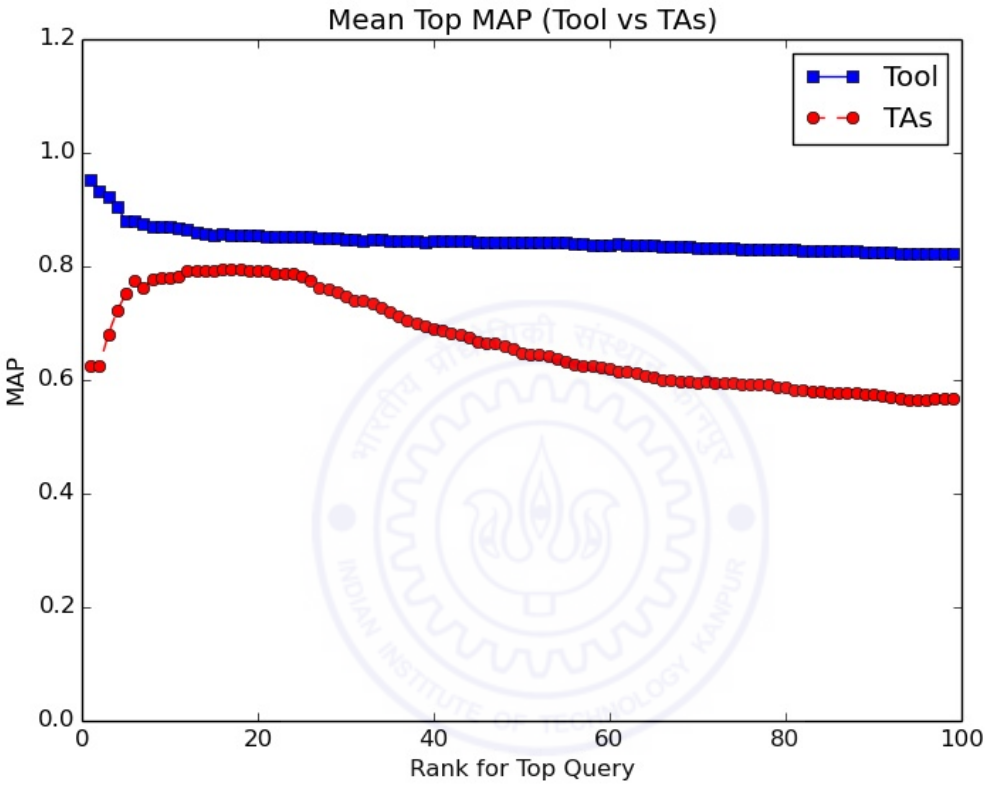


Figure 5.2: Mean MAP for Top-k Queries - Grading Tool vs TAs

Tool's performance is better in terms of MAP.

As we have mentioned t-test [Rum07], which is a statistical test for the null hypothesis that 2 independent samples have identical average (expected) values. Again here, performing t-test for the top-k query MAP values averaged over all submission sets of tool and TAs, the $p\text{-}value = 1.75644954760514 \times 10^{-41}$. This extremely low $p\text{-}value$ implies that the null hypothesis can be rejected. Hence, the differences in top-k MAP values are statistically significant.

5.4.2 Bottom-k Query MAP

For query to retrieve bottom-k ranked submissions the graph plot K versus MAP values follows.

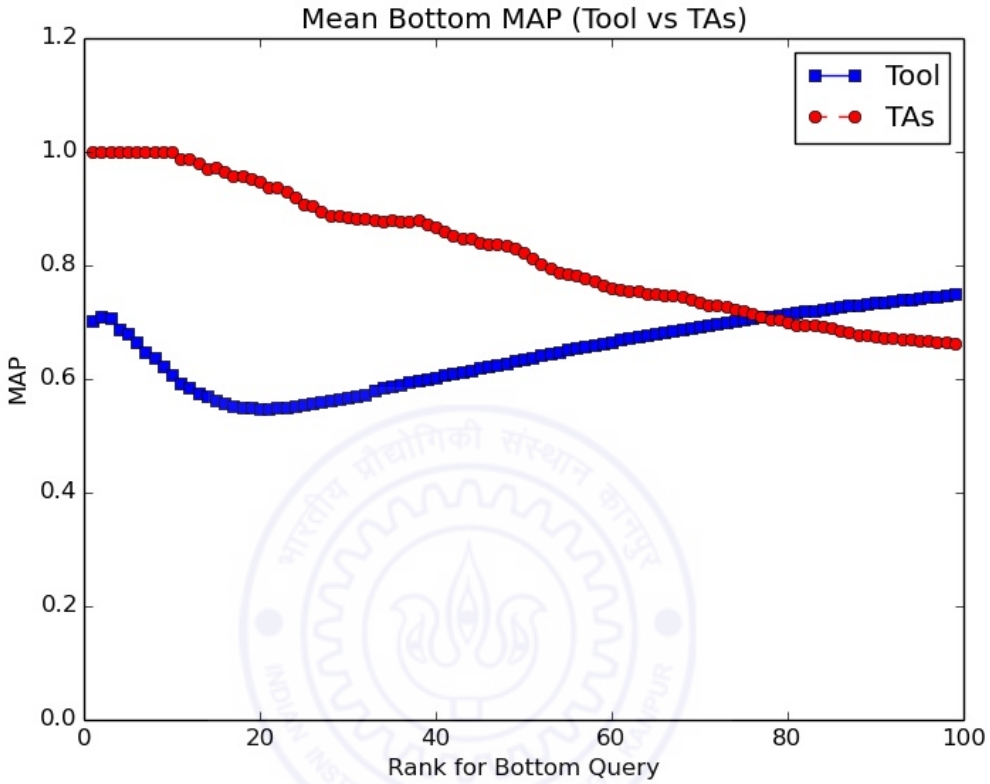


Figure 5.3: Mean MAP for Bottom-k Queries - Grading Tool vs TAs

Our autograding tool performs poorly in terms of MAP for bottom-k queries. It is because autograding tool exhibits lower precision for bottom most submissions. Upon close manual inspection it was observed that there exist a significant number of submissions which pass only a few or no testcases but have been awarded decent grades. Since, the submission has failed on almost all the testcases our autograding tool has graded these solution with very low marks. Now, these submissions are among the bottom most for the tool but at a better rank by the true grades. This is the major reason for low MAP values of out tool. The true grades are not high enough to affect top-k queries.

One possible reason for these submission being awarded good grades despite

testcase failures is the manual judgement. These solution submissions have miss the details to pass testcases but the outline or logic or comments are good enough to be manually judged for decent marks.

Our tool is not capable of manual inspection of code, let alone undertanding the logic in comments or idea through the outline.

As we have mentioned t-test [Rum07] multiple times, which is a statistical test for the null hypothesis that 2 independent samples have identical average (expected) values. Again here, performing t-test for the bottom-k query MAP values averaged over all submission sets of tool and TAs, the $p\text{-}value = 1.25025727023226 \times 10^{-26}$. The extremely low $p\text{-}value$ clearly implies that the null hypothesis can be rejected. Hence, the differences in bottom-k MAP values are statistically significant.

5.4.3 Top-Bottom-k Query Mean MAP

The plot for mean of both top-k and bottom-k ranked submissions retrieval. MAP on y-axis for every k on x-axis is the mean of top-k MAP and bottom-k MAP, as in previous two sections.

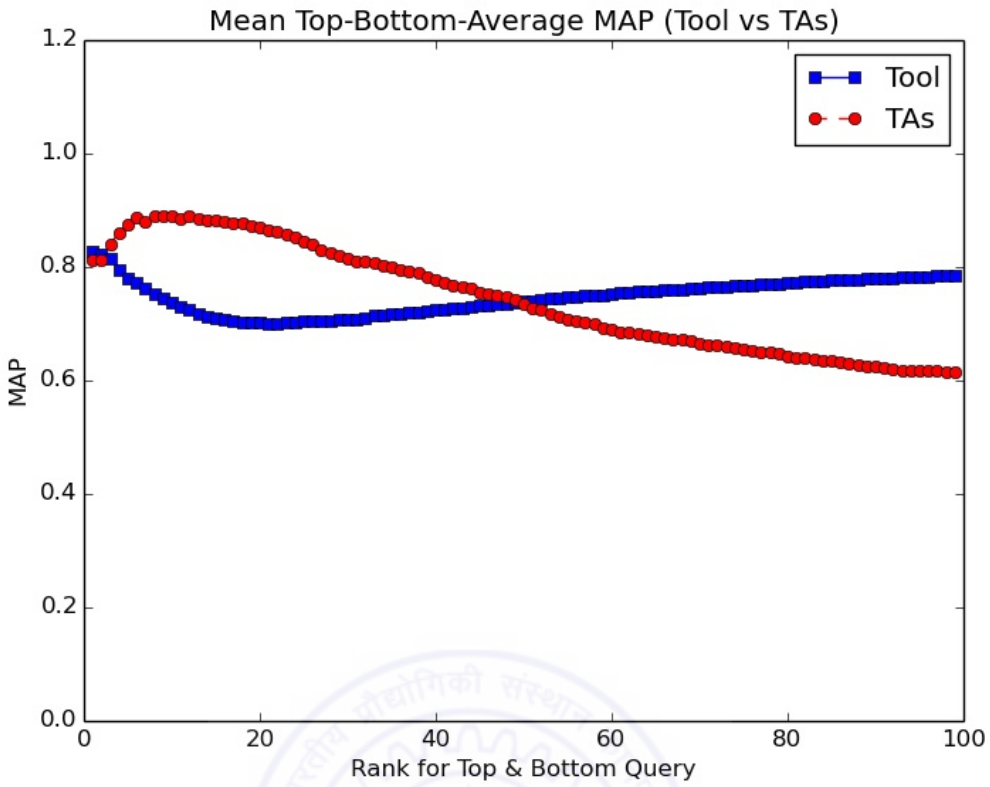


Figure 5.4: Mean MAP for Mean of Top-k & Bottom-k Queries - Tool vs TAs

The tool performs lower for high ranks due to bad performance in bottom-k queries but gradually preforms comparable for higher rank queries. The pattern is just a result for mean of top-k MAP and bottom-k MAP.

As we have mentioned t-test [Rum07] multiple times, which is a statistical test for the null hypothesis that 2 independent samples have identical average (expected) values. Again here, performing t-test for the mean of top-k and bottom-k query MAP values averaged over all submission sets of tool and TAs, the *p-value* = 0.563984354989546. The *p-value* in this case is comparatively quite high. But the two values that are averaged to obtain these values have very low *p-values*. This *p-value* implies that the null hypothesis of identical average scores cannot be rejected. Hence, the differences in this case are statistically insignificant.

Chapter 6

Autograding ReST Server

We have developed this parametrised grading model for a flexible grading policy. Then we have also trained the grade score function to estimate a set of values for its weight parameters which implement the grading policy followed by teaching assistants (TAs) in introductory programming course for 2015 even semester at IIT Kanpur.

In this chapter we discuss the application of our grading model/tool. We have worked with database and data collected by ITS [Das15]. ITS is also used as programming platform for introductory programming course at IIT Kanpur. So, since implemention and integration is possible, it was decided to implement and use this grading model on ITS and add to its features.

So, we aim to provide ReST [Elk] based grading service APIs for ITS using the database of ITS. The backend server is developed with only single html page at frontend for testing server health and the rest of the GUI implementation is left to ITS. The server is capable of running on an independent Docker [Incc] and designed to be integrated into ITS. The server is written in Python [Fou] and runs on Flask [Ron] and uses python module MySQLdb [Dus] for ITS MySQL database [Das15] connectivity.

6.1 Specifications

The purpose of the autograding server is to automatically award grades by implementing the final grade model to solution submitted for assigned programming problems in labs conducted on ITS using ITS database. We discuss the different types of requirements and system configurations in this section.

6.1.1 System Environment and Interface

The key system environment features of the machine on which the server is to be hosted are as follows. This is the assumed set of system configuration for the server hosting environment. The autograding server assumes and needs this environment setup to run.

1. Operating System: Linux based operating system.
2. Server Platform: The server is written in Python 2.7.x and the host system environment is capable of executing .py executable files.
3. Dependencies: Apart from Python 2.7.x the following software packages are installed and available for use on host system:
    - python module flask
    - python module MySQLdb
4. Environment Variables: Following environment variable are defined on host system and are accessible to autograding server:
    - DB_HOST: ITS database MySQL server's IP address.
    - DB_USER: ITS database MySQL's login username.
    - DB_PASS: ITS database MySQL's login username's password.
    - DB_NAME: ITS database MySQL's database name.

Since the final grading model is implemented only as backend service, interaction and communication with the autograding server will only be through the exposed APIs. These APIs follow Representational State Transfer (ReST) based client-server architecture and interface guidelines. Some important characteristics of autograding server interface are as follows.

1. The server will only communicate in reactive manner, i.e. the server will respond to requests and will never proactively communicate.

2. The only purpose of communication to server will be to request grades and from server it will be to respond with grades.

3. All requests to server will be in form of HTTP requests and response will be a JSON object.

4. The server will provide ReSTful web service with following ReST client-server architecture features:

    - Client-Server: Implies separation of concern among client and server. Client is not concerned with the internal working of the server and server is not concerned with UI/UX of the client.
    - Stateless: Server does not maintain any state for client.

6.1.2 Functional Requirements

The purpose of this server is to provide autograding service to ITS. So the only functionality to be addressed by the autograding server is to award grades to ITS submissions according to the final grade model. This service is to be provided by the means of ReSTful APIs so only the backed is required and no frontend is needed.

As discussed in the previous section the only mode of communication for grading with server will be via HTTP requests and response. So, we describe the format for requesting autograding server to award grades and the format in which the server will respond.

The server has a frontend HTML page to test its health. <ip>:<port>/test will render a HTML page which denotes that the server is up and running, where <ip> is the Internet Protocol address of the server and <port> is the listening port configured in the dockerfile.

6.1.3 Request to Server

The request to the server for the grades is to be made as a HTTP POST request on <ip>:<port>/score/assignment with only one POST request body parameter, where <ip> is the Internet Protocol address of the server and <port> is the port open for listening set in dockerfile.

- 'assignment-id': Integer value denoting the assignment id for which the grades is to be awarded. Assignment id is the id that uniquely identifies a problem assigned to a enrolled student as a question for a lab event in the ITS MySQL database.

6.1.4 Response from Server

Response from server will be JavaScript Object Notation (JSON) object with following fields.

- 'status': denotes the status of response. It can be either 'SUCCESS' or 'FAIL'
  - If 'status' is 'SUCCESS' then:
    1. 'status': 'SUCCESS'
    2. 'assignment_id': assignment-id for which the request was made
    3. 'score': float value denoting score between 0.0 and 1.0 inclusive
    4. 'max_marks': maximum marks set for the problem assigned
    5. 'marks': positive integer value denoting grades/marks awarded by the grading model out of maximum marks

6. 'marks_lower': positive integer value denoting lower grades/marks for estimated range, 15% of 'max_marks' less than 'marks'
7. 'marks_upper': positive integer value denoting upper grades/marks for estimated range, 15% of 'max_marks' more than 'marks'

– If 'status' is 'FAIL' the:

1. 'status': 'FAIL'
2. 'errors': array of following error codes:
   * REQUEST_PARAMETER_NOT_FOUND: Denotes that the POST request is missing the 'assignment-id' parameter.
   * INVALID_REQUEST_PARAMETER_VALUE: Denotes that 'assignment-id' parameter value is not and integer.
   * NO_ASSIGNMENT_FOUND: Denotes that the request 'assignment-id' is not found in the database.
   * NO_SCHEDULE_FOUND: Denotes that no lab schedule found for request 'assignment-id'.

6.1.5 Non-Functional Requirements

Non-functional requirements are desirable in every good software. Following are a few non-functional requirements and brief information on the attempted to deliver them.

- Performance: Performance here is in terms of response time. We have tried to optimise the SQL queries to extract all the 7 required features for computing final grade score. Moreover we have added indexes to the SQL tables. One such index is on 'section' field in 'account' table. It can be found in the 'readme' file in server's root directory of the code base.
- Reliability: Proper exception handling is implemented to prevent unexpected behaviour. Such exceptions are reverted back with error codes mentioned in previous section.

- Security: The only input accepted by the server is by HTTP post request with only 1 request parameter. This request parameter is sanitized by allowing only integer values.
- Portability: Dockerfile is present along with the code which enable easy server setup in any linux based system with docker's virtual environment.

6.1.6 Use Cases

As we have mentioned all along that the autograding server has a very specific function, below are the 2 user cases with their basic and alternate flows.

Use Case Number	1
Use Case Name	Check Server Health
Actor	System Administrator
Description	System Administrator checks server health.
Precondition	Server must be successfully running and system administrator must have access to network on which server is hosted.
Trigger/Event	System administrator enter <server-ip>:<port>/test in the web browser address bar.
Basic Flow	1. Actor enter <server-ip>:<port>/test in the address bar of a web browser. 2. The server returns a page with heading denoting that the server is running. 3. The actor gets the confirmation of the server's health.
Alternate Flow 1	1. Actor enter incorrect address. 2. 404 Page not found error returned.
Alternate Flow 2	1. The server is down. 2. Browser returns webpage not available or equivalent error.

Table 6.1: Use Case 1

Use Case Number	2
Use Case Name	Fetch Grades from Server
Actor	Client Consuming autograding APIs
Description	Client attempts to get grades for an assignment-id.
Precondition	Server must be successfully running and the actor must have the assignment-id to be graded.
Trigger/Event	Client consuming autograding APIs makes post request to <ip>:<port>/score/assignment with parameter 'assignment-id'.
Basic Flow	1. Actor makes a post request with body parameter 'assignment-id' as the integer denoting the assignment id. 2. The server returns a JSON object with the grades/marks. 3. The actor obtains the grades awarded by the autograde server form the JSON response object.
Alternate Flow 1	1. The server is down. 2. Browser returns web page not available or equivalent error.
Alternate Flow 2	1. Actor uses incorrect address to make post request. 2. 404 Page not found error returned.
Alternate Flow 3	1. Actor missed parameter 'assignment-id' in post request. 2. Error code REQUEST_PARAMETER_NOT_FOUND returned in JSON object.
Alternate Flow 4	1. The value of post parameter 'assignment-id' is not integer. 2. Error code INVALID_REQUEST_PARAMETER_VALUE returned in JSON object.
Alternate Flow 5	1. The value of post parameter 'assignment-id' is not found for assignment id in the data base. 2. Error code NO_ASSIGNMENT_FOUND returned in JSON.
Alternate Flow 6	1. The value of post parameter 'assignment-id' does not have any lab scheduled in the data base. 2. Error code NO_SCHEDULE_FOUND returned in JSON.

Table 6.2: Use Case 2

6.2 Architecture

This section presents the architecture of the autograding server. The architecture is not complex as there are only few straightforward requirements.

Below is the component/module/process based architecture diagram of the autograding server.

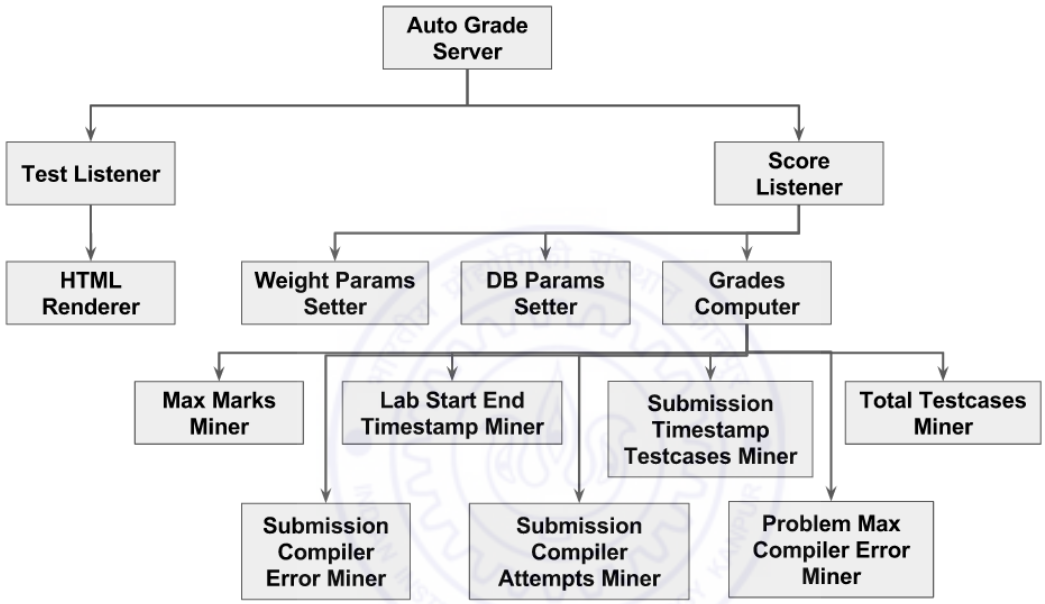


Figure 6.1: Architecture of Autograding Server

The components and their responsibilities are:

1. Auto Grade Server: Represents the whole software.
2. Test Listener: Listens to <ip>:<port>/test get requests and communicates with HTML Renderer to render the server status page.
3. HTML Renderer: Renders the test html page holding the server status.
4. Score Listener: Listens to <ip>:<port>/score/assignment post requests and is responsible to interact with Grades Computer to compute the score & grades and then finally return the score and grades in JSON object.

5. Weight Params Setter: The root component Score Listener depends on this component for setting the final grade score weight parameters. These parameters are stored in file 'config.ini' in the server root folder.

6. DB Params Setter: Called by root component Score Listener, this component sets the server's database parameters by fetching the database parameters from environment variables containing database configuration. These DB parameters are used by all the miner components to access the ITS MySQL database.

7. Grades Computer: It is responsible for computing the final grade function score and return it to the Score Listener. It uses the miner components to fetch all the features from the ITS database. Then simply computes the final grade score using the weightage parameters and features.

8. Max Marks Miner: One of the miner components. Uses the database parameters to mine ITS database for maximum marks set for the problem assignment.

9. Lab Start End Timestamp Miner: Mines the start time and end time unix timestamps of the lab scheduled for the assignment.

10. Submission Timestamp Testcases Miner: Fetches the solution submission's time's unix timestamp and number of testcases it passed from the ITS database.

11. Total Testcases Miner: Fetches the total number of test cases set for the problem assigned.

12. Submission Compiler Error Miner: Mines the number of compile attempt fails for the assignment.

13. Submission Compiler Attempts Miner: Fetches the total number of compilation attempts for the assignment.

14. Problem Max Compiler Error Miner: Mines the ITS database for the maximum number of compiler attempts fails among all the students attempting the same

problem in same lab schedule as the assignment. It is the most expensive miner amoung all above.

The architecture diagram depicts the logical view of autograding server system. Since the server has small size of source code the process view and development view follow almost the same architecture.

The deployment view can be stated as, the server docker image runs in a docker container and the ITS database rests in another docker container. Both the docker containers may or may not be on the same physical server but both the docker are in a common intranet and the ITS database docker container is accessible to the autograding server using the database environment variables of the autograding server's host system.

6.3 Technology

Again, an advantage of simple straightforward requirements, only a single framework to develop the server is sufficient.

Since the experimentation scripts are written in python, most of the mining code in the server can be derived from the scripts used to conduct experiments with the grading models. So, a python programming language based web framework with BSD License was the best option.

Out of many available python web frameworks that fulfill our needs Flask is optimised for small projects and fast development. Hence, it was chosen and the autograding server is written in python that runs on the Flask web framework. Flask is a microframework for Python based on Werkzeug and Jinja2. Our server also needs python module name MySQLdb to connect to the ITS MySQL database.

Apart from Flask and python only a single HTML page is written for displaying server running status. The page is written only using HTML and no Javascript nor CSS is required.

6.4 Performance

Since the autograding server is to be used to add functionality to ITS, the performance load on the server is not expected to surpass the load on ITS. This server is very thin as compared to ITS and hence can easily sustain the load comparable to that on ITS.

The autograding server is dependent on ITS database running on MySQL server. For a single request there are multiple miner components of autograding server that run complex SQL queries on ITS MySQL database. So, we have observed an bottleneck on the MySQL Server.

Therefore, efforts have been made to optimise the miner components and the SQL queries. To provide some flexibility to the server we have added a parameter in the 'config.ini' file in the server root folder. This parameter can be used to turn on/off the most expensive miner component and improve the performance drastically in terms of response time. This provides the system administrator a way control tradeoff between performance and accuracy.

The most expensive miner component is the *problem max compiler error miner*. It is responsible for extracting the maximum number of compilation failures/errors by any student assigned the given problem, e_{max}. This feature is in turn used to compute the compilation grade score function, f_e. If this feature is turned off then the grade score function is set to $f_e = 0.5$ for all grading requests and the grades are calculated accordingly.

The range of average response times for sets of successful grading request on a test machine is mentioned in the table below. The response time may change depending on the host machine but this comparison helps to understand the difference in response time with and without the heavy component.

Problem Max Compiler Error Miner	Response (milliseconds)
On	400ms to 800ms
Off	40ms to 100ms

Table 6.3: Response Times of Autograding Server

6.5 Integration

ITS is web app developed for scalability and has high modularity. One of the key reasons that enable high modularity at high level is the use of Dockers.

Docker (Software) is linux package that automates the deployment of projects within an operating-system-level virtual environment called containers. It provides an layer of abstraction over OS allowing easy automation of deployment. It is very light wight as compared to virtual machine.

Following the technology, the server is Docker capable. The server code consists of a file named 'Dockerfile' in the parent directory of the server root directory. The Dockerfile contains the Docker configurations sufficient for a linux system with Docker to setup container running the docker image of auto grading server. One the server's docker image is built and run in docker container the autograding service APIs are available for use.

This deployment model makes the integration of auograding server simple. Also, we have mentioned these details again in 'readme' file in the server root folder.

Chapter 7

Conclusions and Future Scope

This section concludes the thesis. Our own interpretations of conclusion on entire work and this thesis is presented and some ideas of extending and deriving applicable work are discussed.

7.1 Conclusions

At the beginning of this thesis, aim was to come up with a grading model for introductory programming courses and develop a tool implementing the grading model. Most appropriate implementation of the grading model would be to serve ITS [Das15] through a server exposing ReSTful [Elk] API service.

We have presented the final grading model which was designed during the course of this thesis. It servers as the grading model for the autograding server, a backend ReST based service to grade submission on ITS.

The working grading model successfully runs on the autograding server. The server is designed to work with docker software making it compatible with ITS. The simple integration of the server into to ITS allows it use the ITS MySQL database. Then the ReST APIs can be used as desired.

After performing various experiments and judging our grading model on factors that are more related to the purpose of grading it would be safe to say that model's performance is comparable to that of the teaching assistants. For these experiments

the weight parameter were learned which implicitly set a grading policy. These weight with different values can implement a variety of grading policies. Results and statistics show that the grading scheme/model/tool marginally outperforms or is comparable to the teaching assistants (TAs) in terms of correctness & accuracy for most of the experiments and shows inferior performance in one of the experiments.

There may be various applications of the grading model and autograding server but few have been listed below.

1. Grades awarded by the model/server can be used as recommendations to help the TAs while grading.

2. During the lab every solution submission can be followed by the grades awarded by the model/server as feedback to students.

3. Final awarded grades having high difference with the grades given by the model/server can be reported to instructors as potential inconsistencies.

7.2 Future Scope

While working on this thesis some ideas came across. Few such ideas follow.

7.2.1 Application of Autograding Server in ITS

The grading model developed in this thesis is implemented in the autograding server. The autograding server has been developed to be integrated into ITS.

The autograding server provides only a ReST based service. It is an backend server which only serves grades for solution submissions. It contains no presentation layer or frontend.

There are various ways the autograding server can be used to add features and functionalities to ITS. But any such usage or applications of autograding server in ITS would require appropriate modifications and development in ITS frontend and

logic middleware. Few such application suggested by Dr. Arnab Bhattacharya and Dr. Amey Karkare, my thesis guide and co-guide are as follows.

1. For Instructors:

   (a) On the grading page, a button to evaluate particular submission. The output will be a range of marks (fuzzy score).

   (b) On dashboard, a way to evaluate all submissions for an lab event. The tool marks/grades can be shown along side the TA marks/grades.

   (c) A way to generate report for a set of events. Events can be presented in a checkbox list to select.

2. For Students:

   - A button to self-evaluate only practice problems. Tool output in this case should be mapped to either star rating, or a progress-bar. Absolute or fuzzy numeric scores are not shown to students. For example, 0 stars for <20%, 1 star for 21-40%, 2 stars for 41-60% and so on. The button will be disabled for all students when lab/exam event is in progress.

Addition of these features will require considerable work on ITS frontend and logic middleware. This should be the next logical step in our work.

7.2.2 Scope of Improvement in Grading Model

The grading model is simply a weighted function of various features. More features can be taken into consideration in the grading model. Adding new feature will enhance accuracy and correctness and also add more flexibility. Extracting these features might need some post processing after mining the database.

1. Degree of indentation, accuracy of indentation.

2. Comments used, location and length of comments and some keywords may be searched for.

More experiments can be done with grading score function and feature score functions. Currently these functions is just weightage linear functions. Instead of a linear function, different distributions can be tried in form of parametrised functions.

A scope of improvement also exists in training the weights. Better training golden dataset can be used to train with advanced machine learning techniques.

7.2.3 Extending Grading Model

Currently we have developed a model capable of awarding grades to introductory programming solutions. We can extend the model's capabilities.

1. Lab Score: A scoring mechanism to extend students' problem score to lab score. Lab consists of multiple problems. A simple approach might be to again take weighted average of the problems score. Weights proportional to problem's maximum marks.

2. Cumulative Score: A scoring mechanism to compute students' cumulative score for multiple labs. The score should represent student's programming proficiency focusing on recent performance. One approach is again take weighted average over lab scores with weight proportional to recency of the labs.

3. Problem Difficulty Score: A score to represent perceived problem difficulty. This score should be inversely proportional to student's solution score and directly proportional to student's cumulative score.

4. Concept Score: For a student it should represent the score for particular concept or skill such as loops, conditions, etc. Simple annotation of problems with concept tags is the first requirement. Then again a weighted score of problems with the tag score should contribute to a tag score. The score should be proportional to problem difficulty and also proportional to student's problem score.

7.2.4 Scope of Improvement in the Autograding Server

The server just implements the grading model, so major functional enhancements depend entirely on the grading model. Since the server uses the ITS MySQL database scalability in terms of load on the ITS MySQL database and response time is critical.

Hence, the feature mining components can be improved to reduce load on the ITS MySQL server and decrease the response time of the autograding server. The SQL queries can be optimised. Further, some tables may be added to the database and these tables may be updated using triggers to maintain the set of features for solution in least expensive manner.

Application improvements may include adding API services such as potential outliers detection. For a question in a lab we have a set of submissions. A service that reports submissions in a set having high rank/grades difference in grading by the grading model tool and by the TAs.

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

References

[Das15] Rajdeep Das. "A Platform for Data Analysis and Tutoring For Introductory Programming". M.Tech. thesis. India: Indian Institute of Technology Kanpur, 2015.

[Cin13] Can Cemal Cingi. "Computer Aided Education". In: *Procedia-Social and Behavioral Sciences* 103 (2013), pp. 220–229.

[MPV12] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*. Vol. 821. John Wiley & Sons, 2012.

[WM03] A.D. Well and J.L. Myers. *Research Design & Statistical Analysis*. Taylor & Francis, 2003. ISBN: 9781135641085. URL: https://books.google.co.in/books?id=56EcG4noR0kC.

[MRS08] C.D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. An Introduction to Information Retrieval. Cambridge University Press, 2008. ISBN: 9780521865715. URL: https://books.google.co.in/books?id=GNvtngEACAAJ.

[LJ99] M Luck and M Joy. "Computer-based submission and assessment in BOSS". In: *Interactions Online Journal* 6 (1999).

[AB99] Arnow and Barshey. "WebToTeach: An interactive focused programming exercise system". In: *IEEE Frontiers in Education Conference (FIE)* (1999).

[Dal99] Charlie Daly. "RoboProf and an Introductory Computer Programming Course". In: *SIGCSE Bull.* (1999), pp. 155–158. ISSN: 0097-8418. DOI: 10.1145/384267.305904. URL: http://doi.acm.org/10.1145/384267.305904.

[Hig+05] Colin A. Higgins et al. "Automated Assessment and Experiences of Teaching Programming". In: *J. Educ. Resour. Comput.* (2005).

[TBR05] Nghi Truong, Peter Bancroft, and Paul Roe. "Learning to program through the web". In: *ACM SIGCSE Bulletin* 37.3 (2005), pp. 9–13.

[Zam+06] Norshuhani Zamin et al. "WAGS: A Web-Based Automated Grading System For Programming Assignments From Users' Perspectives". In: *Proceed-ings of International Conference on Programming Classes Communica-tion of Science & Technology, Malaysia*. 2006.

[K+07] AR Khirulnizam, Che WSBCWA MDJN, et al. "Development of an Automated Assessment for C Programming Exercises Using Pseudoeodes Comparison Teehnique". In: *Conference on Information Tech-nology Research and Applications, Selangor Malaysia*. 2007.

[Ala05] Kirsti M Ala-Mutka. "A survey of automated assessment approaches for programming assignments". In: *Computer science education* 15.2 (2005), pp. 83–102.

[Ree89] Kenneth A Reek. "The TRY system-or-how to avoid testing student programs". In: *ACM SIGCSE Bulletin*. Vol. 21. 1. ACM. 1989, pp. 112–116.

[SMK01] Riku Saikkonen, Lauri Malmi, and Ari Korhonen. "Fully automatic assessment of programming exercises". In: *ACM Sigcse Bulletin*. Vol. 33. 3. ACM. 2001, pp. 133–136.

[Che+03] Brenda Cheang et al. "On Automated Grading of Programming Assignments in an Academic Institution". In: *Comput. Educ.* 41.2 (Sept. 2003), pp. 121–131. ISSN: 0360-1315. DOI: 10.1016/S0360-1315(03)00030-7. URL: http://dx.doi.org/10.1016/S0360-1315(03)00030-7.

[EFK04] Christopher C Ellsworth, James B Fenwick Jr, and Barry L Kurtz. "The quiver system". In: *ACM SIGCSE Bulletin*. Vol. 36. 1. ACM. 2004, pp. 205–209.

[DH04] Charlie Daly and Jane M Horgan. "An automated learning system for Java programming". In: *Education, IEEE Transactions on* 47.1 (2004), pp. 10–17.

[JGB05] Mike Joy, Nathan Griffiths, and Russell Boyatt. "The boss online submission and assessment system". In: *Journal on Educational Resources in Computing (JERIC)* 5.3 (2005), p. 2.

[Wan+11] Tiantian Wang et al. "Ability-training-oriented automated assessment in introductory programming course". In: *Computers & Education* 56.1 (2011), pp. 220–226.

[Bis+15] Judith Bishop et al. "Code Hunt: Experience with coding contests at scale". In: *Proc. ICSE, JSEET* (2015).

[Inca] Wikipedia: The Free Encyclopedia. Wikimedia Foundation Inc. *Average absolute deviation*. URL: https://en.wikipedia.org/wiki/Average_absolute_deviation.

[Incb] Wikipedia: The Free Encyclopedia. Wikimedia Foundation Inc. *Root-mean-square deviation*. URL: https://en.wikipedia.org/wiki/Root-mean-square_deviation.

[Rum07] D.J. Rumsey. *Intermediate Statistics For Dummies*. –For dummies. Wiley, 2007. ISBN: 9780470147740. URL: https://books.google.co.in/books?id=jwmdUe0dDSAC.

[Elk] Dr. M. Elkstein. *Learn REST: A Tutorial*. URL: http://rest.elkstein.org/.

[Incc] Docker Inc. *Get Started with Docker for Linux*. URL: http://docs.docker.com/linux/started/.

[Fou] Python Software Foundation. *Python 2.7.10 documentation*. URL: https://docs.python.org/2.7/.

[Ron] Armin Ronacher. *Welcome to Flask*. URL: http://flask.pocoo.org/docs/0.9/.

[Dus] Andy Dustman. *MySQLdb User's Guide*. URL: http://mysql-python.sourceforge.net/MySQLdb.html.