Improving feedback for a web-based marking system

Paul Thomson

Supervisor: Ewan Klein

Fourth Year Project Report
Software Engineering
School of Informatics
University of Edinburgh
2013

Abstract

This thesis will discuss the work done towards improving the feedback provided to users of the Infandango system.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Paul Thomson*
Supervisor: Ewan Klein)

Table of Contents

Chapter 1

Introduction

Infandango is an open source web-based system for automated grading of Java code submitted by students[8]. The aim of this project is to improve the feedback provided to students by using machine learning methods to display a visual representation of their current progress. In chapter 3 the design process will be discussed, explaining why certain choices were made. The implementation of these decisions will be discussed in chapter 5 with evaluation and conclusion following.

Chapter 2

Background

2.1 Infandango

Infandango is an automated web-based marking system for student submitted programming exercises. A student can view the list of warm-up, optional and core exercises and choose to submit a file for one of them. This file is then compiled and tested by Jester in a sandbox. Between the web frontend and Jester there is a PostgresSQL database which stores which files have been submitted for which users and how these files performed on the JUnit tests.

2.1.1 Current feedback

The primary source of feedback in Infandango is displayed in Figure 2.1. Each submissions is marked with a set of JUnit tests and the fraction of these tests which are correct is displayed. This fraction is converted into a percentage and displyed on a green or red background depending on the value. More general feedback is also available which displays similar information but the results are displayed by week rather than by question.

2.2 Literature

Khan Academy[3] is a website which provides users with online education material:

> Our online materials cover subjects ranging from math and finance to history and art. With thousands of bite-sized videos, step-by-step problems and instant data[1]

A blog post[7] written by David Hu about Khan Academy demonstrates that different feedback measures can affect user performance significantly. The original Khan Academy system required a user to get 10 consecutive exercises of a certain type correct before they can be deemed proficient[1] at that type of exercise. In an attempt to improve this system, a logistic regression model is used to calculate the probability that a user passes the next exercise successfully, with a threshold of 94% representing the new proficieny level. Over a 6 day period 10% of users tested the new method. Users of the new system earned 20.8% more proficiences, attempted 15.7% more exercises and required 26% less exercises per proficiency. Hu summarises by saying the boost seems to come from allowing users to move on from exercises which they already proficient at, without requiring them to complete their streak thus wasting time on something they already understand.

---

[1]A proficiency is earned when a user is deemed to be "proficient" at a certain kind of exercise

7 - [core]	Safer Fixed Divider	4 / 5	(80%)
8 - [core]	Safer Quadratic Solver	5 / 5	(100%)
9 - [core]	Squares Loop	12 / 13	(92%)
10 - [optional]	Lopsided Number Triangle	0 / 1	(0%)
11 - [optional]	Gambler's Ruin	0 / 2	(0%)

Figure 2.1: This is a crop of what will be displayed to the user for a given week

Chapter 3

Design

With promising results for machine learning based visual feedback a similar design could be made for Infandango.

3.1 Language and Tools

Two languages are immediate possibilities for implementation: Java[4] and Python[5]. I had most experience with Java, and some parts of Infandango were already written in Java. Most of Infandango was, however, written Python and I had some experience with it as well. After researching appropriate languages for machine learning R[6] become a third possible choice. Follow research on all languages Python seemed most appropriate: integrating with Infandango would be straightforward and scikit-learn[2] would remove the challenge of implementing machine learning methods.

3.2 Proposed Design

Using anonymised data from previous years a model can be built using machine learning methods to provide feedback. Using this model and some recent scores from the user, the following score can be predicted.

3.2.1 Model

The Khan Academy model is built based on one assumption: a user can always *generate another example* and it will be of the *same type*. These are two qualities which Infandango does not share. Infandango has a preset number of manually written exer-

cises and they are not grouped *strictly* by similarity. The first proposed model was to take the average for each week and use those averages as features for the model and the next week's average as the class. There are however some problems with this model:

- The semester has 8 weeks worth of exercises. If all 7 weeks are used as a feature and the score for week 8 is predicted then the user would not receive any feedback until the last week. If feedback is to be generated for each week then a separate model needs to be trained for each week: one model where there is only 1 week of evidence and week 2 is predicted, another where there are 2 weeks of evidence and week 3 is predicted and so on. Neither of these options are desirable.

- Even after 3-5 weeks there would only just be enough data to start getting reasonable results CITE HERE

- Grouping data like this means we have a lot less training examples

A similar alternative to this model is to treat each question within a week separately, giving the model a much larger dimensionality. Although this does remove the latter two problems, the first problem still remains. This also raises the likelihood of data being missing for a feature (it is more common for a student to miss one exercise within a week than the whole week).

Both of these models have been working with the assumption that we want to learn something about *specific* questions. So if, for example, a question was particularly hard then the model might be able to learn that it should predict lower scores for that question. However, we have seen the difficulties that these models incur. For this reason a much simpler model was created. The model has N features, each a percentage. Each feature represents a score from the corresponding previous question. The class is the score for the $N+1$ problem, so when a user is using the system it will try to predict their next score given their previous N scores. This creates a moving window of previous scores, with each feature simply being *a question* rather than, for example *week 3, question 2b*.

3.2.1.1 Unexplored Alternatives

There are many unexplored possibilities for other features like a weighted moving average or the number of problems unfinished. With this project it was important to complete the system quickly so that user tests can be done over a period of weeks.

Other possibilities for features and other learning methods can be explored in later projects.

3.2.1.2 Training the model

Django is used to retrieve the training data from the anonymised data from a previous year. This data is filtered into groups of *N+1* consecutive results, with the final result being the class. For each set of *K* results *K-N* sets of results are created. This provides the machine learning methods with a lot more training data than alternative models.
In order to decide which learning method to use sci-kit learn's cross validation method splits the data into training and testing sets, with a testing size of 20%. Different machine learning methods were then trained on this data and tested by comparing their R2 scores. In Figure 3.1 it can also be seen that these tests are done multiple times over different values of *N*. The highest accuracy is obtained by Logistic Regression at *N* = 4. Logistic Regression is also the most consitent of the methods, thus making it the most appropriate choice.

3.2.2 Visualising the results

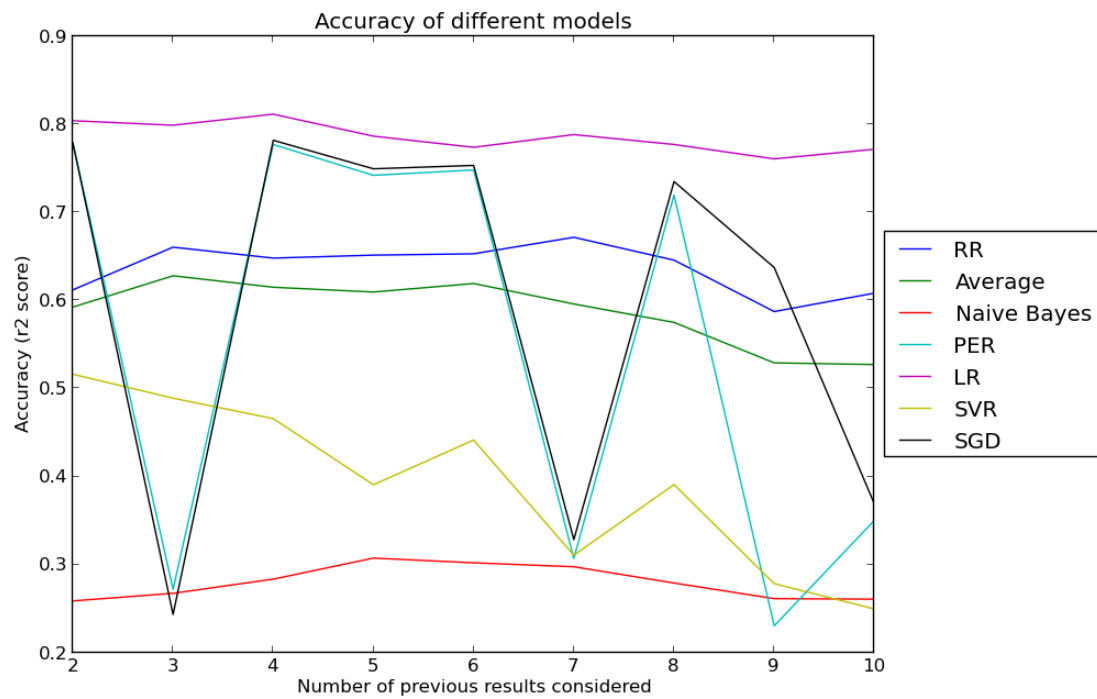# 3.3 Integration with Infandango

Figure 3.1: Comparison of different machine learning methods comparing their accuracy against the number of previous exercises considered

Chapter 4

Choosing a model

There are many aspects to choosing a model and these choices greatly impact on the final performance of the model. Unlike the Khan Academy example, these questions do not have a binary PASS or FAIL quality, they only have a percentage correct. For this reason the problem is treated as a regression problem (predicting the percentage score of the next question) opposed to a classification problem (predicting if the next submission will PASS or FAIL).

4.1   Feature selection

The Khan Academy model is built based on one assumption: a user can always *generate another example* and it will be of the *same type*. These are two qualities which Infandango does not share. Infandango has a preset number of manually written exercises and they are not grouped *strictly* by similarity. The first set of features used was the average mark for each week, giving 8 features: one for each week). The first reason this is, theoretically, a good set of features is that for every training example the features represent the same questions. This means the model has the potential to learn about each individual week (this will not hold true for some later feature sets). Also by using the average mark the problem of missing data is much less common. However these features are not suitable for a number of reasons:

- The semester has 8 weeks worth of exercises. If all 7 weeks are used as a feature and the score for week 8 is predicted then the user would not receive any feedback until the last week which would be too late for the user to change their learning habits accordingly. If new feedback is to be generated each week then a separate model needs to be trained for each week (since the number of features

11

would be changing): one model where there is only one week of evidence and week 2 is predicted, another where there are two weeks of evidence and week 3 is predicted and so on. This would be needlessly complicated and would only update the feedback weekly

- Even after 3-5 weeks there would only just be enough data to start getting reasonable results CITE HERE

- Grouping data like this means we have a lot less training examples

A similar alternative to these features is to treat each question within a week separately, increasing the dimensionality considerably. Although this does remove the latter two problems, the first problem still remains: we would need to train separate models in order to continually update feedback. This also raises the likelihood of data being missing for a feature (it is more common for a student to miss one exercise within a week than miss the whole week).

Both of these models have been working with the assumption that we want to learn something about *specific* questions. So if, for example, a question was particularly hard then the model might be able to learn that it should predict lower scores for that question. However, it is not desirable to train a model to predict each question. For this reason a much simpler model was created. The model has *N* features, each a percentage. Each feature represents a score from a question. The model attempts to predict the score for the *N+1* problem. So when a user is using the system it will try to predict their next score given their previous *N* scores. This creates a moving window of previous scores, with each feature being an *anonymous* question rather than, for example *week 3, question 2b*.

A final feature was created to utilise the rest of the data not included in the preceding *N* exercises: The average. This will allow a model to take into account the overall performance of a student, rather than just their recent performance.

4.2   Selection of preliminary models

The package being used for the majority of the machine learning is scikit-learn (SKL): a general-purpose machine learning library for python. Since the purpose of this project is not to implement machine learning methods, but to explore the problem using machine learning methods, it is quicker and simpler to use a pre-existing library.

Using this library makes choosing the preliminary models a much less difficult decision: use lots of models. With the relatively low dimensionality of the problem it is very quick to train and test models so there is little reason to exclude a model from the test set. The following are the models from SKL which are used:

- Support Vector Machine

- Logistic Regression

- Linear Regression

- Ridge Regression

- Lasso

- Elastic Net

An interesting non-linear model which is missing from this list is Neural Networks. SKL does not have the capability of creating and training a Neural Network but a python library that specialises in Neural Networks is pybrain. Although using a Neural Network required learning and using another library it is useful to have multiple non-linear models.

4.2.1 Training the models

The baseline to test against is quite simple: the average of all the features. Although simple this model has given better results than some of the machine learning models. To compare the models a python script was created which split the data into a training and test set. The models are trained on the training data and the mean square error is calculated using the testing data. However, since the value for N is not fixed this test is performed for values of N ranging from 2 to 9. The errors can then be plotted against N to show how the accuracy changes with the number of features but also how each model performs against the others.

The problem with testing like this is the results can vary quite a bit by chance since the amount of data is not particularly large. To remove some of this variance k-fold cross validation is performed. In k-fold cross validation the original data set is split into k subsets, each of roughly equivalent size. The procedure above is performed with k-1 sets as training data and 1 set as testing data. This is performed k times, each time
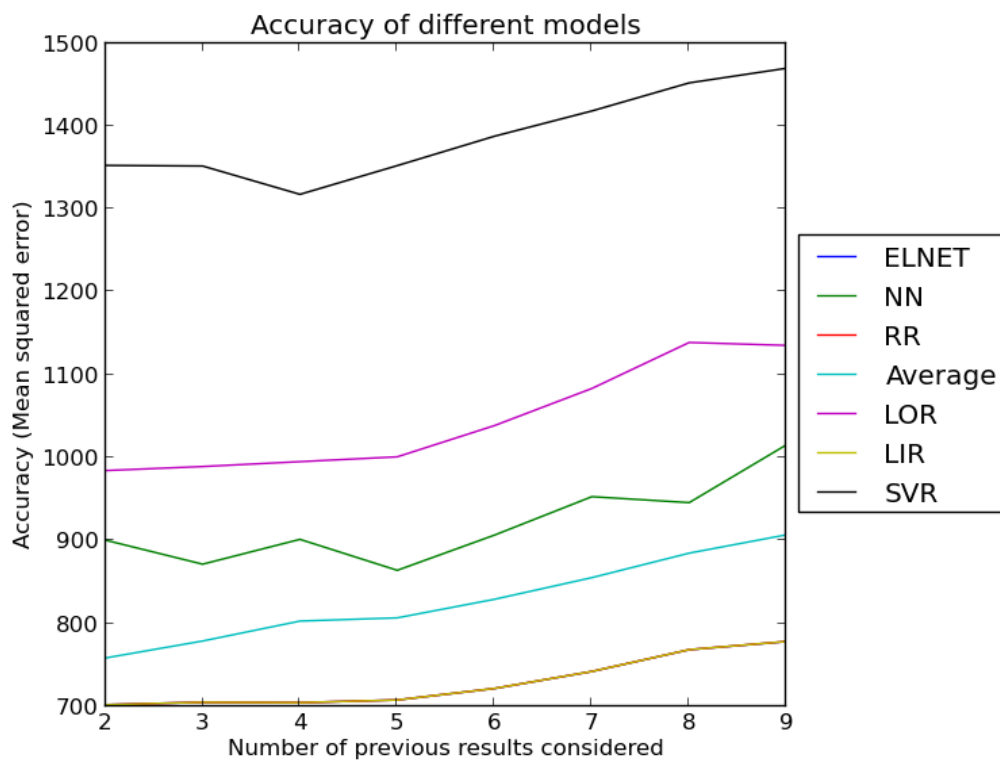
Figure 4.1: Comparison of different machine learning methods comparing their accuracy against the number of previous exercises considered

alternating which set is used as the testing set so that by the end each set has been used as the testing set. This provides more robust results which are less prone to variance.

Figure 4.1 appears to tell us that LIR (Linear Regression) performs the best. However, it also seems that RR (Ridge Regression), ELNET (Elastic Net) and Lasso are missing from the graph. However, if the graph is magnified on the Linear Regression line we get Figure 4.2

This graph shows that the linear models all perform very similarly with their maximum variance being displayed in Figure 4.2 as roughly 0.06. Although not visible in the previous figure, it can be seen in Figure 4.3 that Ridge Regression follows Linear Regression very closely.

4.3   Final model

4.3.1   Optimisation
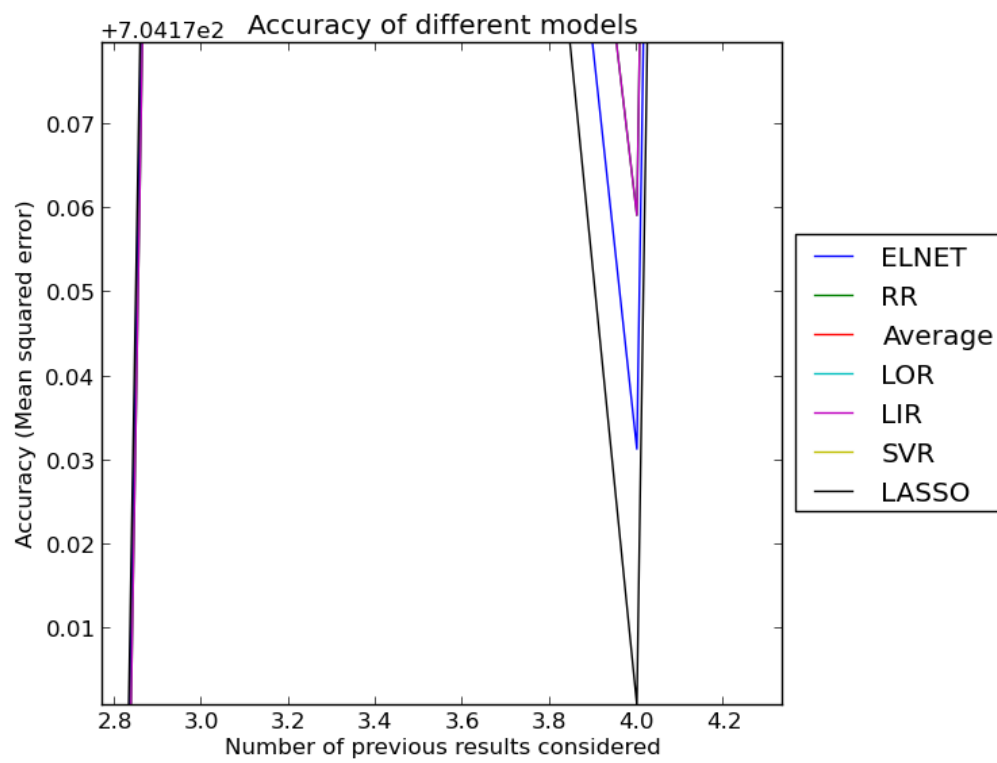
4.4   Testing the model
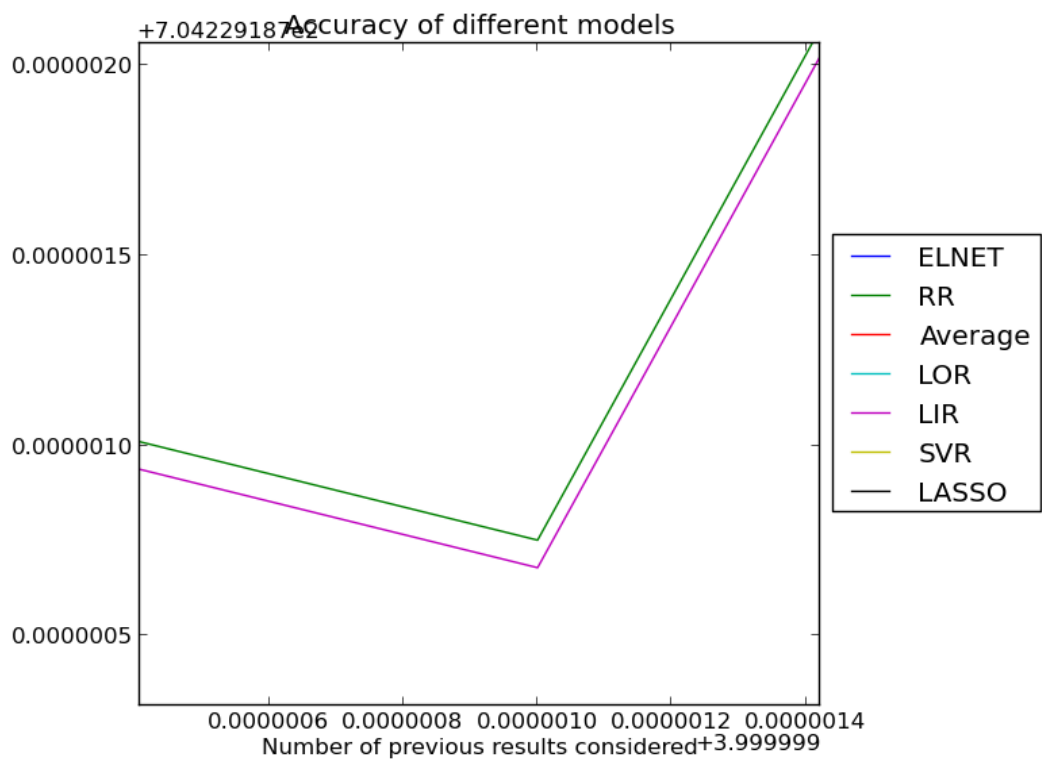
Figure 4.2: Similarity of the linear models

Figure 4.3: Ridge regression follows extremely closely to linear regression

Chapter 5

Implementation

Chapter 6

Evaluation

Chapter 7

Conclusion

Bibliography

[1] http://khanacademy.desk.com/customer/portal/articles/337790-what-is-khan-academy-.

[2] http://scikit-learn.org.

[3] https://www.khanacademy.org/.

[4] http://www.java.com/.

[5] http://www.python.org/.

[6] http://www.r-project.org/.

[7] D. Hu. How khan academy is using machine learning to assess student mastery, Nov 2011. http://david-hu.com/2011/11/02/how-khan-academy-is-using-machine-learning-to-assess-student-mastery.html.

[8] E. K. Mike Hull, Dan Powell. Infandango: Automated grading for student programming.