Improving feedback for a web-based marking system

Paul Thomson

Fourth Year Project Report

Software Engineering

School of Informatics

University of Edinburgh

2013

Abstract

This report discusses the work done towards improving the feedback supplied to users of the automated web-based marking system, Infandango. By applying machine learning methods to submission data recorded from students of a previous class, a model has been created which can predict a single score for the user which represents their progress so far. Through processes of optimisation and testing a Support Vector Machine has been chosen. This has been integrated with the Infandango system, providing a visual mechanism to display the cumulative feedback to the student.

Acknowledgements

Thanks to Ewan Klein for the regular meetings, feedback and various other help he gave me throughout the project. Thanks to Iain Murray for all the help with the machine learning parts and other feedback from the group meetings. Thanks to friends and family that helped with spelling and grammar checking.

Table of Contents

1 Introduction **1**

1.1 Infandango 1

1.1.1 Overview 1

1.1.2 Current Use 1

1.1.3 Future Use 2

1.1.4 Current feedback 2

1.2 Project Goals 2

2 Literature **5**

2.1 Automated Marking Systems 5

2.1.1 RoboProf 6

2.1.2 Automated Evaluation of Programming Assignments 6

2.2 Visualisation 7

2.3 Cognitive Modelling 8

2.4 Machine Learning and Feedback for an Online Marking System . . . 9

2.5 Conclusion 11

3 Design **13**

3.1 Proposed Design 13

3.1.1 Model 13

3.1.2 Visualisation 17

3.2 Integration with Infandango 18

4 Model Selection **21**

4.1 Feature selection 21

4.1.1 Question Identity 22

4.2 Preliminary models 22

4.2.1 Classifiers 22

4.2.2 Regression . . . 23
4.2.3 Model definitions . . . 23
4.3 Training and optimising . . . 25
4.3.1 K-fold cross-validation . . . 25
4.4 Results . . . 26
4.4.1 Linear Classifiers . . . 26
4.4.2 Support Vector Machines . . . 31
4.4.3 Artificial Neural Network . . . 37
4.4.4 Final Model Selection . . . 39

5 Implementation **43**
5.1 Introduction . . . 43
5.2 Language and Tools . . . 43
5.3 Visual Design . . . 44
5.4 Prediction Model . . . 44
5.5 System Integration . . . 45
5.6 Technical Issues . . . 45
5.6.1 Neural Network Hidden Layers . . . 45
5.6.2 Slow Neural Networks . . . 46
5.7 Conclusion . . . 46

6 Conclusion **49**
6.1 Evaluation of Goal Completion . . . 49
6.1.1 Research of Literature . . . 49
6.1.2 Feedback Design . . . 49
6.1.3 Implementation and Integration . . . 50
6.1.4 User Evaluation . . . 50
6.1.5 Overall Feedback Improvement . . . 51
6.2 Further Work . . . 51

A K-means Clustering **53**

Bibliography **57**

Chapter 1

Introduction

1.1 Infandango

1.1.1 Overview

Infandango[17] is an automated web-based marking system for student submitted programming exercises, currently only for the Java language. A student can view the list of warm-up, optional and core exercises and choose to submit a file for one of them. Jester is the testing daemon for Infandango which compiles and runs code inside a sandbox for security purposes. Between the web frontend and Jester there is a PostgreSQL database which stores the source code of each submission and the score information for each marked submission.

Each question has a label: **warmup** questions are simple questions which can be skipped if the user feels confident, **core** questions are questions which the user is highly encouraged to try and may affect the end coursework mark, and **optional** questions are provided for particularly interested students.

1.1.2 Current Use

The system has been used with the first year Java programming course at the University of Edinburgh for a few years. The database information for these years has been kept and retains all the information about submissions: marks, submission time, number of resubmissions. The database for one year has been anonymised and made available for use in this project if necessary. Only one year is available because the questions have changed since previous years and therefore the data would be inconsistent with the current questions.

1.1.3 Future Use

Although Infandango was designed for a specific use, it has remained a general framework which can be adapted to have different content and is available as open-source software on bitbucket.org[14]. Work is also currently being done to expand the number of possible languages that can be tested through Infandango[28].

1.1.4 Current feedback

The primary source of feedback in Infandango is displayed in Figure 1.1. Each submission is marked with a set of JUnit tests and typically one mark is given for each test that is passed. The marks are distributed such that at least 25 marks are available for each week. The fraction of passed tests to total tests is converted into a percentage and displayed on a red (0 - 40%), orange (40%-70%) or green (70%-100%) background. More general feedback is also available which displays similar information but the results are displayed by week rather than by question.

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 1.1: This is a cropped screenshot of what is displayed to the user for a given week in the current Infandango system

1.2 Project Goals

The general goal of this project was to improve the feedback for Infandango. In order to achieve this, a list of objectives was created to facilitate the research, design and implementation of a suitable modification to the system:

1. Research and investigate current systems for providing automatic feedback to students. Examples where this feedback pertains to code are of particular interest.

2. Through evaluation of both the literature and the system choose an appropriate approach for providing feedback.

3. Implement the approach and then, when it is functional, integrate with the Infandango system.

4. Perform user evaluation of the completed work to determine any impact it has on the learning of the student

Most of this report will be discussing how these objectives have been realised and the extent to which they have been completed will be discussed in Chapter 6.

Chapter 2

Literature

This chapter will first look at some general aspects of automated marking systems which already exist, and may give insight into how to improve the feedback of Infandango. The way the feedback is displayed defines what information is displayed to the student and so there will be discussion of how to visualise the feedback. The area of Cognitive Modelling will be discussed because it has had work performed to specifically give better feedback to students that use automated marking systems. The final piece of literature that will be discussed is about a system which is particularly similar to Infandango and which, recently, changed their method of feedback.

2.1 Automated Marking Systems

There are many other systems like Infandango and covering all of them in this report would be excessive. The systems which are discussed exemplify certain aspects of automated marking systems which are worth discussing with respect to Infandango. By looking at these systems and comparing the different forms of feedback they give it is hoped that weaknesses and potential improvements to Infandango's feedback system can be identified.

These systems are being discussed primarily for their marking and feedback methods, thus the other details of implementation will be largely ignored, but some form of code submission can be assumed for all these systems.

2.1.1 RoboProf

RoboProf[8] provides small programming exercises which start simple and get progressively harder, with immediate feedback about the correctness of the program guiding the student. For example, a student may be provided with a program which loops over a range of numbers and prints them. The student may then be asked to modify the for-loop in order to print out a different series of numbers. The immediate feedback gives the students encouragement and motivates them to keep trying more questions. This approach is similar to Infandango because both systems return immediate feedback based purely on the correctness of the output of the problem, and this is a common approach in other systems too[4].

In the paper one problem is identified: the immediate feedback is sometimes too addictive, and the students may end up spending more time than is necessary performing the exercises.

2.1.2 Automated Evaluation of Programming Assignments

Kaushal and Singh describe various measures used as part of an automated marking system: regularity, integrity, efficiency and accuracy[13].

Regularity The time at which a submission is made with respect to the deadline for that submission

Integrity The originality of the document determined through a plagiarism detector

Efficiency A qualitative measure of the time taken and memory used by a student's programs

Accuracy The percentage of test cases for each program that the student's submission passes

The experimenters track the change in these measures as students use the system and find that there is a general improvement on all categories when feedback is based on these measurements. There are other systems which use similar measures of feedback[12] and this automated feedback has be shown to be at least as good as manual feedback[10].

One disadvantage of having different measures for feedback is that this still requires manual work in the specification of the feedback. For example, the ASSYST system[12] uses a metric first proposed by Berry and Meekings[6] to measure the style

of the submitted programs. This means tutors must specify four values which create a simple graph which determines the mark a student receives based on their style score. The specific details can be found in the paper, but what it means is that each time a new measurement is to be used for a new feedback measure, then a new metric system must be implemented as well as the measurement itself. A system in which the measurement itself is enough, and the rest of the process is automated, would be an improvment.

2.2 Visualisation

In his book *Visual Display of Quantitative Information*[26], Edward Tufte discusses methods of efficiently and effectively displaying information. In Chapter 4 "Data-Ink and Graphical Redesign", Tufte introduces the idea of Data-Ink: The amount of "ink" which is actually used to represent the data you are interested in. In Tufte's words:

> Data-ink is the non-erasable core of a graphic, the non-redundant ink arranged in reponse to the variation in the numbers represented.

Data-ink is desirable, unavoidable information so Tufte says designers should strive towards a high data-ink ratio, removing as much non-information ink as possible to avoid overwhelming the data. Tufte then continues by giving examples where redundant information is desirable. One of these examples uses a graphic from Linus Pauling's General Chemistry[19]. Figure 2.1 shows how, based on the low data-ink ratio principle, Tufte removes the grid marks and part of the frame which leaves little else but the data itself.

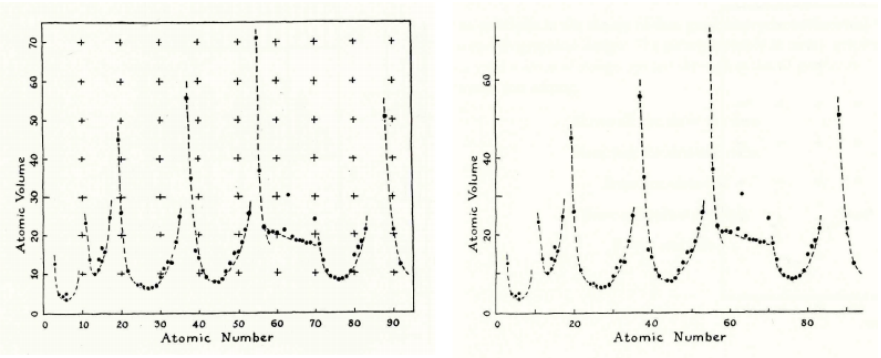

Figure 2.1: (a) shows the original graph and (b) shows how Tufte recommends changing the graph to remove redundant information

Tufte concludes the chapter with 5 maxims summarising this principle.

> Above all else show the data.
>
> Maximize the data-ink ratio.
>
> Erase non-data-ink.
>
> Erase redundant data-ink.
>
> Revise and edit.

Infandango currently adheres to these principles quite well. For example, the score which is provided for each question is displayed as a percentage on a block of colour, which is dependent on the score. At first the block of colour may seem redundant since it is a feature related directly to the score so it doesn't seem to add much information. However, the score alone would not tell the user if that score is "good" or "bad", that is a judgement the user would have to make for themselves. Providing this information as a colour is a simple, intuitive way of telling the use whether their performance on an exercise is acceptable or not.

2.3 Cognitive Modelling

The cognitive modelling of students can provide access to important information for automatic tutoring systems. A cognitive model is a computer model of the cognitive processes of a student, which allows insight into how the student might approach the problem, or how difficult the problem might be for the student. Building this model manually requires a lot of human effort[15] and so automating this process is desirable.

SimStudent is a machine learning agent designed to build cognitive models automatically via machine learning. The machine learning method used for parts of SimStudent is a First Order Inductive Learner, which is given some some observations and background knowledge from which it draws hypotheses. Using algebra as an example, SimStudent will be given some basic operators as background knowledge (add, subtract, multiply, divide) and will then be given an algebraic problem to solve. It will then suggest the next step to the student who may reject or accept the suggestion, thus providing negative or positive feedback to SimStudent to modify its production rules.

As long as the students perform consistently, SimStudent can model the performance of the students with an accuracy of 83%[16]. Being able to model and predict a student's performance can allow the tutoring system to target feedback at the areas

which it knows the student might struggle with, thus catching problems before they occur and understanding why certain mistakes happen. This research shows that machine learning can be used to generate feedback which is better or equivalent to manual feedback.

2.4 Machine Learning and Feedback for an Online Marking System

Khan Academy[1] 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[2]

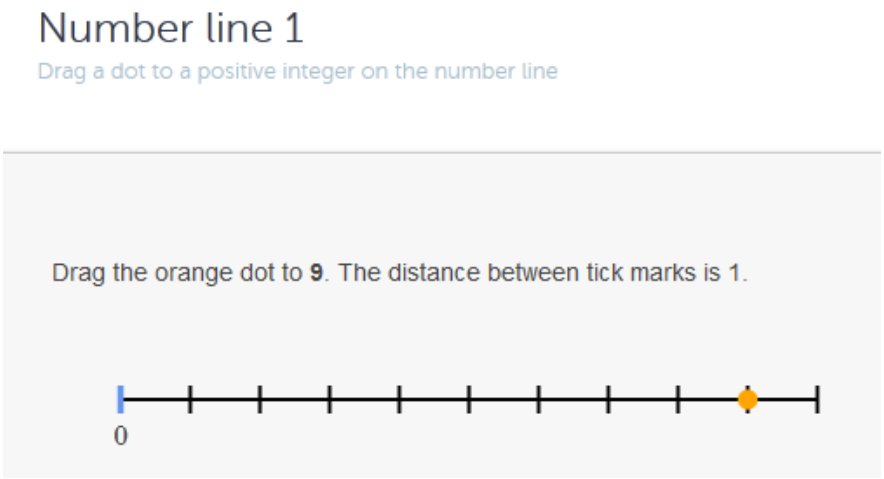

Figure 2.2: A screenshot of the online Khan Academy system, showing a simple exercise where the user must choose the correct position on a number line.

A blog post[11] written by David Hu about Khan Academy explains how they went about changing their feedback system. Khan Academy gives users certain kinds of exercises, for example choosing the approriate position on a number scale (see Figure 2.2). It can then generate endless variations of this problem for the user to continue attempting until they are deemed proficient. Proficiency is the phrase used by Khan Academy when it judges the user to be good enough at a certain type of exercise that the user should move on. The original Khan Academy system required the user to get 10 consecutive exercises of a certain type correct (a “streak”) before they are qualified as proficient for that type of exercise.

The main problem with this system is that a user could get 9 consecutive exercises correct and then make a mistake on the final exercise. Before the user could move on they would have to get 10 more consecutive exercises correct. Hu decided to search for a new system which would be less fickle, thus removing some of the frustration experienced by students.

In an attempt to improve this system, Hu proposed using a logistic regression model to calculate the probability that a user passes the next exercise successfully, with a threshold of 94% representing the new proficiency level. To explain a logistic regression it helps to have some terminology.

features The observed data which will be used to make a prediction, e.g. the mark for the previous submission

weights A constant multiplier applied to the features. The higher the weight, the more that feature contributes to the prediction

The logistic regression is quite simple: it applies the weights to the features and sums the results to get a single value. In order to turn this value into a probability it must be squashed into the range 0, 1. The function used to do this is the logistic function, where x is the value you want to squash into the range 0, 1:

$$\frac{1}{1+e^{-x}}$$

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 are already proficient at, without requiring them to complete their streak.

Although the current Infandango system does not require perfection like Khan Academy did, it is possible that users are more reluctant to move on from exercises before they achieve 100% because with the marking done automatically with objective tests it is possible and common to get 100% for exercises. Encouraging the user to move on when they do not yet have 100% could encourage students to tackle more difficult problems.

This feedback is also interesting because it provides feedback across multiple exercises at the same time, but Infandango only marks individual exercises. Adapting the Khan Academy approach to provide summary feedback over multiple exercises gives the student an idea of their global progress.

2.5 Conclusion

The review of other automated marking systems shows us that there are other forms of feedback — not just correctness of code — which can be used in such systems. However, the current implementations still require some manual specification about how the measurement is translated into student feedback. A method of feedback which automatically translated measurements into feedback would be an interesting addition to Infandango.

The literature provides a strong case for applying machine learning to the problem of improving automated feedback. Infandango currently lacks feedback which uses information from multiple exercises, and having such feedback may allow more detailed and targeted feedback. While cognitive modelling is effective, it is more suited to rule based learning, where logic can be applied to the knowledge in order to learn hypotheses. The Khan Academy approach is more applicable to Infandango as it only needs to know the mark for an exercise, not the rules which a student applies in order to solve the exercise.

Chapter 3

Design

Previous work[11][16] has shown that machine learning can be an excellent method of generating feedback for automated marking systems. Machine learning also has the advantage of allowing a variety of measurements to plug into the final feedback: if a measurement can be made of a submission or a student, this measurement can be added to the inputs of the machine learning model and the learning algorithm will automatically learn how to use this new measurement.

3.1 Proposed Design

Using the data provided of submissions from a previous year, a model will be trained using machine learning methods. The model will output a value which will be displayed somewhere in the Infandango system. In Chapter 4 these potential models will be explored further, leading to a decision on which is the best to use with Infandango.

3.1.1 Model

The Infandango system is similar to the Khan Academy system and so the method Khan Academy uses is an approriate starting point. Infandango allows a user to submit many solutions for an exercise (therefore a submission in Infandango refers to one specific solution for an exercise, not the best solution, or the final solution) and so does Khan Academy. However, there are two properties that distinguish Khan Academy from Infandango:

- For each exercise, the next solution submitted is for the same kind of question but the details are randomly generated: a user can keep submitting solutions to

an exercise even after they get one solution correct. However, in Infandango resubmissions are for exactly the same question, so once a user gets 100% there is no reason for the user to submit another solution because they have already perfected that exercise

- A submission is given a binary score: correct or incorrect, whereas Infandango gives a percentage score as there are multiple tests for the same submission

The first difference is significant, and requires a change in granularity of the data: instead of measuring progress on a submission-by-submission basis, the focus should be on the final mark a user will achieve for an exercise. So instead of trying to predict the score for the next *submission*, try to predict the final score for the next *question*.

Now that it was clear what data was going to be used and how, some exploratory work was done to learn the structure and size of the data. Although users are encouraged to answer all questions, they do not receive marks directly for each question. This means students do not have to answer all questions which makes missing data a potential problem. Figure 3.1 shows the submission rates for all the questions. It shows that submission rates can get as low as around 10% for some questions. Figure 3.2 shows the submission rates for only core questions. The submission rates are on average higher than for non-core questions. Combined with the fact that users are likely to have more motivation for core questions (since they potentially count towards their final mark) data from now on will only be considering core questions.

The reason for this missing data is not clear, but there are some potential causes.

- In a questionnaire filled out at the beginning of the course, 25% of students said they were already experienced with Java. These students may find other work more worthwhile
- Since not all exercises necessarily contribute to the final grade received for the course students may not prioritise them
- Students may become disengaged from the course

With the amount of missing data still being significant changing to yet another level of granularity was considered: predicting on a week-by-week basis. Using this method there could be a lot less missing data: take the average of all the questions for a week to get the score for that week. This means there will still be a data point for a student if they miss one question, and they would have to have no solutions for all questions in

Figure 3.1: Submission rates for all questions

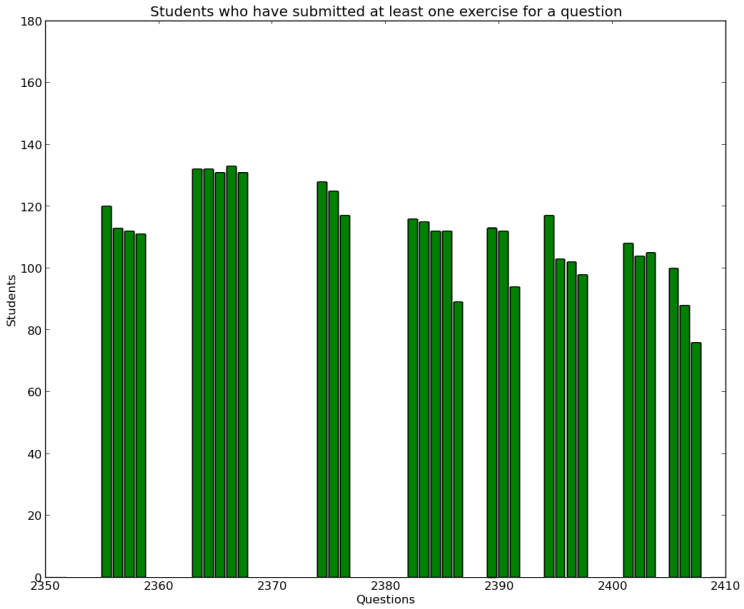

Figure 3.2: Submission rates for only core questions

a week to get no score for a week. Another solution would be to give students a 0 for exercises they do not complete in a given week, thus penalising them for not having completed the exercises. Comparing Figure 3.3 to Figure 3.2 a rise in the amount of

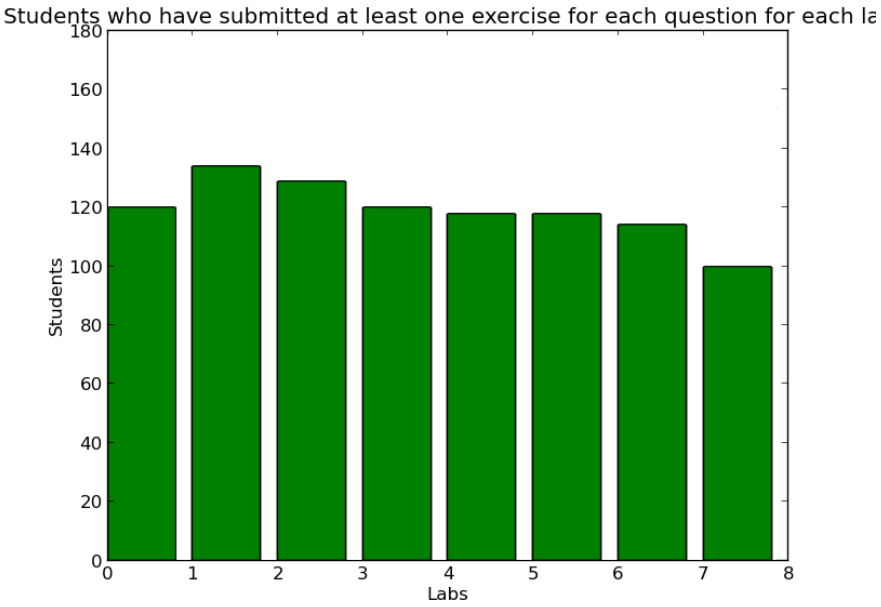

Figure 3.3: Submission rates for at least one question per week

data points can be seen. On average there is about 115 data points for a given week, and about 100 data points for a given question.

Although this does provide slightly higher yield there are disadvantages to predicting on a week-by-week basis:

1. It would take a week before there was even one data point to start giving feedback from

2. Feedback would only be updated weekly. After a student has been working on questions for a few days the feedback is likely to be outdated

3. A separate model would need to be trained for each week. To predict the score for week 3 you want to use two features: the score for weeks 1 and 2. For week 8 you would like to use all the available features: weeks 1 to 7. Having a variable number of features means having different models for each week

Problems 1 and 2 are not really present if prediction is done question-by-question. However, problem 3 is still present: a new model needs to be trained for each question. A solution to this problem will be discussed later.

3.1.1.1 Identity based features

Starting with the Khan Academy model and adapting it based on Infandango and the available data, a model can be proposed for Infandango: For each question, train a new model which takes all previous questions as input features. The model then tries to predict the score for that question.

3.1.1.2 Moving Window

Previously an assumption has been made: the identity of the question is necessary for the score to be informative. While the identity of the question is likely to be important it is not necessary and removing this constraint allows a simple, singular alternative model to be proposed: The model always has N features and tries to predict the score of the N+1 question. For the simplicity of the following description N = 5 will be assumed. There are 5 input features. The features will be decided based on their locality to the output feature: if we want to make a prediction for question 6 then the input features will be questions 1, 2, 3, 4 and 5. All the input features occur before question 6 because when a student is completing questions they are much more likely to do them in order, and so questions 7, 8 and 9 are not likely to present when making predictions. Although this approach does remove the significant information of the identity of the questions it is simple to implement and it also provides a lot more training data: There are 30 questions, and so using this approach for each student there would be 25 potential training examples. However, using the previous model there would only be 1 item for each model.

3.1.2 Visualisation

The goal of the visualisation is to use the predicted score to generate some abstract representation of how well the user is doing. In the same way the old Khan Academy system requires an unbroken streak of 10 correct exercises, allowing the user to see this score directly may cause them to try achieve similarly stringent standards (100%), thus wasting time on exercises they can already perform well. Using the predicted score to generate categorical information than a numerical percentage may allow the user to move forward to complete exercises they have not tried yet.

3.1.2.1 Ideas

Using an image editing tool some mockup designs were created. These designs were created to be simple enough to display information at a glance and also be suitable to place on every page, so a student always has access to their current process. Figure 3.4 shows ideas using a sequence of boxes which fill as the score increases. The number of boxes is a factor which could be changed. A comprimise has to be reached between having too few boxes and therefore losing accuracy, and having too many boxes and changing the categorical data back into numerical data. 9 seemed a good compromise, meaning that any prediction which is at higher than 89% will fill all boxes, encouraging the user to move on. A similar bar shape could be used which has a continuous filling mechanism but this lacks the abstraction granted by the discretisation into boxes. Figure 3.5 shows similar ideas using a circle instead of a sequence of boxes.

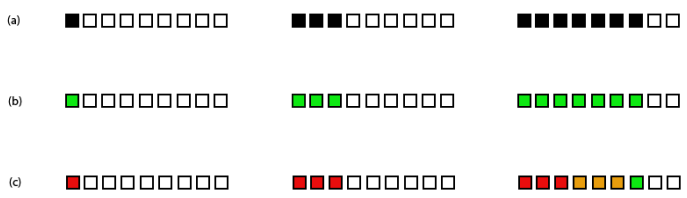

Figure 3.4: (a), (b) and (c) show the progressions of different visualisations as the score increases. (a) and (b) are monochrome while (c) changes depending on the score

3.2 Integration with Infandango

The final step in completing the project is to integrate the system with Infandango. This is a non-trivial task whose outcome will determine the future usability of this addition to the system. Infandango uses Django[9] to fetch the information from the database which is then displayed via HTML and CSS. This includes information such as the submissions and the scores received for each submission. The feedback score is of a similar nature so, in line with the design of Infandango, the feedback score will be

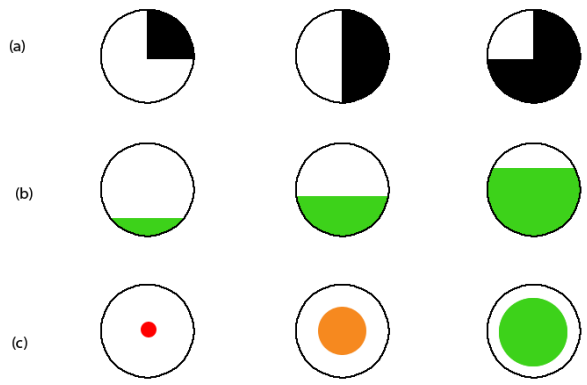

Figure 3.5: How different visualisations change with an increasing score. Each of the three methods have different colour schemes and filling methods which can be swapped between the methods

displayed using HTML and CSS after being fetched by Django. The precise details of this integration will be discussion in Chapter 5.

Chapter 4

Model Selection

In Chapter 3 two data represenations were described which could be used in this scenario: moving window approach which treats the question results anonymously and the identity based approach which must be have a separate model to predict each question. To find out which data representation to use the first step is to choose some machine learning models to test on. Selecting the best model is a frequent problem which is still an active area of research. The approach here will choose a selection of models from which to start. These models will be trained and tested on the data provided from the submissions of a previous year using k-fold cross-validation. The test results will then be analysed to choose a model with the lowest predicted error.

4.1 Feature selection

The first action to be taken when creating a model is to determine the features. The features determine the inputs to the model; the data which the model can use to learn and predict. Although the content of the inputs has been mostly decided already — the submission score — there is one aspect which has not been decided: how many scores should be considered? The moving window model must decide how large the window should be and the identity based model should consider the performance across the different model instantiations. The number of input features is a factor which will be discussed in the results section.

4.1.1 Question Identity

The two data representations differ on essentially one issue: is the question identity retained by the input features? One model says yes: retain the identities of the questions by always putting the same question into the same position in the input vector (which means a new model must be created for each question). The second approach says no: create a moving window of input features from the questions immediately before the question we want to predict. This approach ignores a lot of the information: If the model is predicting the score for question 20 then it takes the results from questions 14 - 19 and ignores the other 13 questions. The model cannot take all the scores individually or it would be identical to the previous model, so a crude method of using all that extra data is to add an extra feature which is the numerical average of the unused data. By adding this feature to the moving window approach it remains simpler but doesn't completely ignore all the questions outside of the window.

4.2 Preliminary models

Most models will be chosen from the scikit-learn[20] package except for Artificial Neural Networks (ANN) which are not present in scikit-learn. Pybrain provides the implementation of ANNs used here. As this is primarily an implementation issue this will be discussed further in Chapter 5.

Deciding on the models to use is a difficult process. However, since scikit-learn provides algorithms optimised for speed, most models are very quick to train and test. This allows the option of starting with a variety of models and then evaluating how they behave against each other. Depending on the type of data the expect and output, the models can be split into two different types: classifiers and regression models.

4.2.1 Classifiers

Classifiers attempt to identify the *class* an observation should be assigned to depending on the input features. A *class* is a discrete category, for example dog or cat. However what the models are trying to predict is the score of the next exercise which is a continuous value. To resolve this problem some sort of discretisation needs to be performed to change a percentage into a class. The granularity of this discretisation is important as it determines the amount of accuracy that can be achieved.

A natural discretisation is already present in the current design: the visualisation

on the webpage. The visualisation uses 9 blocks to display the score. For the sake of this application, any further accuracy would be lost when the information is displayed. So, in order to use classifiers on this problem, the output feature (the score we want to predict) will be discretised into one of 9 classes, distributed evenly over the range 0-100%.

4.2.2 Regression

A regression model differs from a classifier because it tries to predict a continuous, numerical value. In this case, the regression will try directly predict the score for the next exercise. Unlike the classifiers, no adaptations need to be made to the data before supplying it to the regression models since regressions output a continuous value like a percentage.

4.2.3 Model definitions

4.2.3.1 Linear Models

The following models are all linear regressions: they expect the target value to be a linear combination of the input features. The differences between them occur in the constraints applied to the coefficients.

Linear Regression This is the simplest linear model. It uses Ordinary Least Squares to estimate the coefficients of the input vector. This means it tries to minimise the residual sum of squares between the result given by the labelled data and the result predicted by the model. The equation for a linear regression is the sum of the weighted input:

$$y = w_0 + w^T x$$

w_0 is the bias value which is a constant unaffected by the input vector. The vector w is the weight vector which is calculated during training, and w^T is the transpose of this vector to allow it to be multiplied by x, the input vector.

Ridge Regression This model adapts Linear Regression by adding to the minimisation value a penalty depending on the size of the coefficients. Scikit allows control of this penalty through the *alpha* parameter.

LASSO This model applies a similar penalty as Ridge Regression except use L1 norm instead. Scikit allows control of this penalty through the *alpha* parameter.

Elastic Net Elastic Net combines the penalties of Ridge Regression and LASSO to provide a tradeoff between the two functionalities. This tradeoff can be controlled through the l1_ratio parameter.

Cross-validation Scikit also provides versions of these models which automatically apply cross-valdiation to determine the best values for their respective parameters.

Logistic Regression Although the name is misleading, this model is a classifier[24]. Logistic Regression forms performs a linear regression for each class: if an observation belongs to the class the output becomes 1, otherwise the output is 0. When a new object is to be classified, the linear expression is evaluated for each class and the result which is the largest is the class which is chosen[29]. This method is known as the one-vs-all method. It can also use either L1 or L2 regularisation. The type of regularisation is the method used to calculate error, and so it defines how much a model is penalised for being wrong. In the following equations x is a vector of the input space, for example the difference between an observed point and a predicted point.

L1

$$\|x\|_1 = \sum_i |x_i|$$

L2

$$\|x\|_2 = \sqrt{\sum_i |x_i^2|}$$

4.2.3.2 Support Vector Machines

At the simplest level a Support Vector Machine (SVM) takes some input and assigns it to a binary class. At first this seems as linear as the previous models, but performing the *kernel trick* (see section 4.4.2.1 allows the SVM to deal with non-linear data. The choice of kernel can significantly affect the performance of the model. Two SVM implementations have been considered for this problem.

Support Vector Classifier Scikit provides more than one Support Vector Classifier but SVC has been chosen due to its support of non-linear kernels. SVC uses the one-against-one approach for multiclass problems.

Support Vector Regression SVR is scitkit's implementation of Support Vector Regression. It also supports non-linear kernels.

4.2.3.3 Artificial Neural Network

ANNs were chosen because they can model very complex relationships which some of the previous models cannot. Like all models, it takes the input vector and produces an output. It does this by passing the input through a series of hidden layers. These hidden layers are created and calibrated during the training process. The library pybrain[23] had to be used for this because scikit provides no implementation of an Artificial Neural Network. Pybrain provides a wide variety of options, too many to be considered in this one implementation. Here are some of the options pybrain provides: Back propagation trainer, choice of hidden layer, choice of number of hidden units.

4.3 Training and optimising

Determining the best training methods and model parameters are tasks which can always be improved. Given the timescale of the project and the need to also integrate the model with the system, there are some model parameters and optimisation methods which have not been thoroughly investigated.

4.3.1 K-fold cross-validation

When a model is being trained and tested it is important to perform the training and testing on different sets of data otherwise overfitting will occur. Naively, it would be intuitive to split the data up into two sets: train all models on the training set and test all models on the testing set. While this avoids the problem of overfitting it causes a reduction in the amount of data that the model can be trained on, thus making it less accurate. The single split may also have split the data in such a way that one of the models happens to perform worse, because observations in the training set may be irrelevant to the observations in the testing set. Thus, the standard method of measuring the error rate of a machine learning model is k-fold cross-validation[29].

In k-fold cross-validation the data is split into k distinct sets of data of roughly the same size. Training and testing is then performed k times, each time using k-1 sets of data as the training set and 1 set of data as the testing set. The set used for testing is changed after each iteration so that each set of data is used exactly once for testing, and k-1 times for training. Usually, the results will then be averaged to get a single error estimate for each model.

Although larger values of k can results in increased accuracy[3], 10-fold cross-validation will be used for all tests due to its simplicity.

4.4 Results

Standard measurements of error (e.g. mean squared error) are dependent on whether the model is a classifier or a regressor: the mean squared error does not really make sense to a classifier, because it predicts a discrete class, not a continuous number. Similarly, using the frequency of an incorrect answer for a regressor does not make sense: it is highly unlikely to predict the exact continuous value. Therefore to allow comparison of the two types of models the output of the regressor is binned according to the available classes: the range 0-100 will be split into 9 equal ranges, *bins*. A percentage is converted into a class by determining the bin that contains it. Therefore binning the continuous values allows comparison with discrete values.

Error function The function to determine the error will count the number of times the model predicts the incorrect class and divide this by the total number of examples. After multiplying this by 100 a percentage will be returned which represents the error rate.

4.4.1 Linear Classifiers

Due to the similarity of the Linear Regression based classifiers, these will be considered together and a representative model will be chosen based on the performances. As mentioned at the beginning of this chapter, both data representations have a variable number of input features and so accuracy will be discussed with respect to the number of input features.

4.4.1.1 Optimisation

The three variations of Linear Regression: Ridge, LASSO and Elastic Net each have parameters which can be tuned which affect regularisation. The method of choosing these parameters is similar to the method used to compared the models themselves. Samples are taken from the range of possible values for each parameter (Elastic Net has two parameters, while Ridge and LASSO only have one). For each of these possible values, 10-fold cross validation is performed, with the average result being used as the error for that set of parameters.

4.4.1.2 Optimisation Results

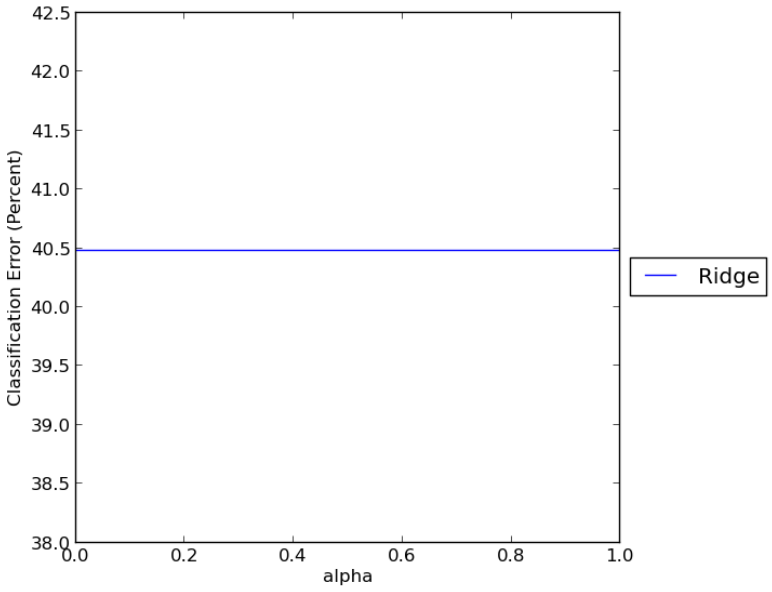

Figure 4.1: The effect on error rate of changing the alpha parameter on Ridge Regression

Figures 4.1, 4.2 and 4.3 show that none of the parameters have a drastic effect on the error rate of the models. The results for Ridge Regression show no change in the performance of the classifier when changing the parameter, alpha. The following values will be used for these models from now on: Ridge Regression: alpha = 1; LASSO: alpha = 1; ElasticNet: a = 0, b = 0.

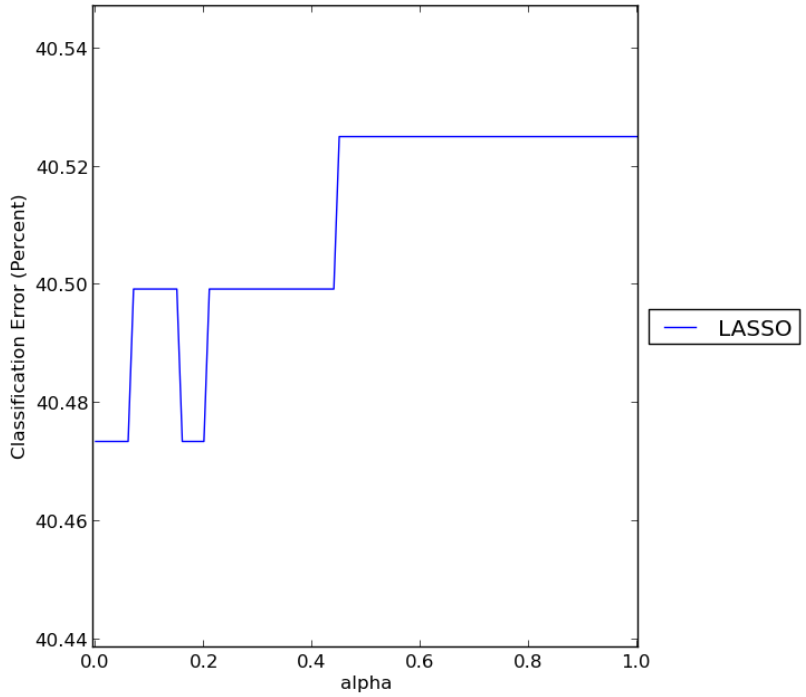

Figure 4.2: The effect on error rate of changing the alpha parameter on LASSO

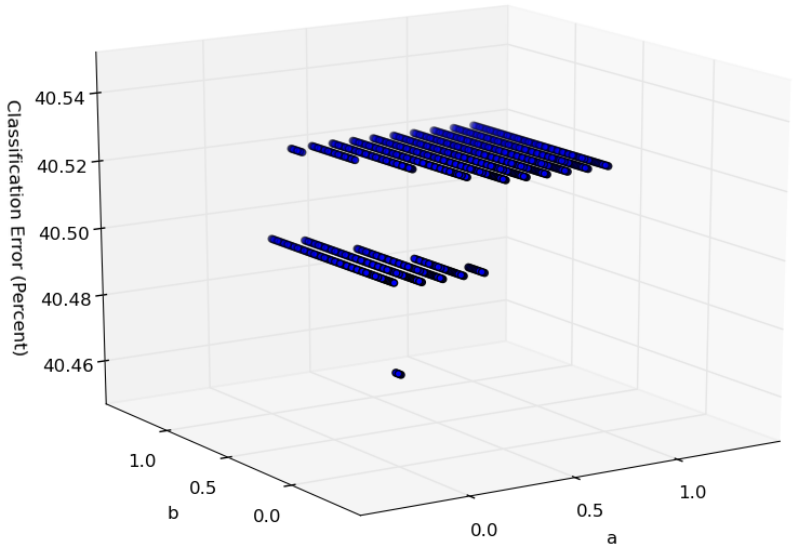

Figure 4.3: The effect on error rate of changing the a and b parameters on Elastic Net

4.4.1.3 Comparing the Linear Models

Using the cross-validation method described previously, the 4 different linear models were trained with the parameters discovered via optimisation.

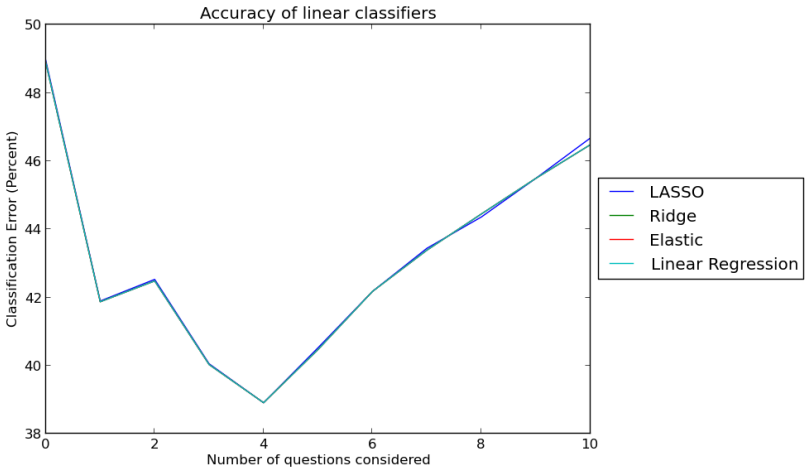

Figure 4.4: The error rates for linear models using the moving window data representation

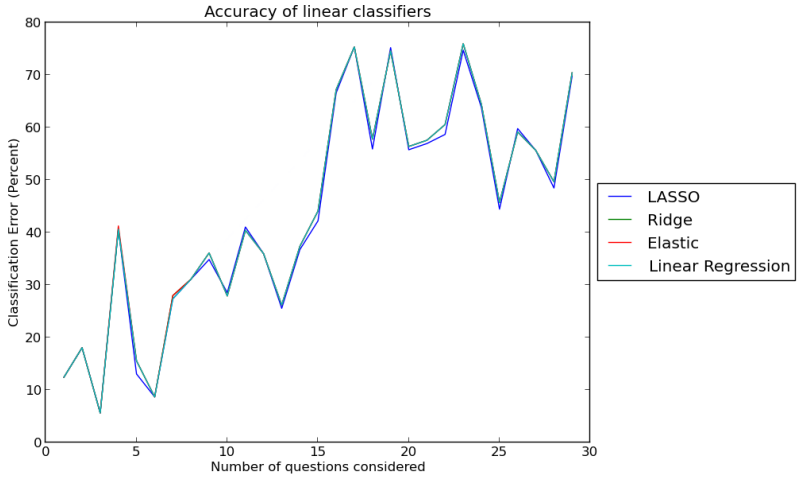

Figure 4.5: The error rates for linear models using the identity based data representation

Figure 4.4 shows that the performance of the models depends largely on the size of the moving window. For these models the lowest error rate is acheived when the window is of size 4. For this data, the error rate increases as more questions are considered

which is at first unexpected: if the model knows more about the previous questions then it should be able to have a more accurate prediction.

One possible explanation is that the data from questions further back is not as relevant. Questions in Infandango are grouped - though not strictly - by the type of the questions: a question is likely to build on knowledge of the previous question. For this reason, results of questions within the same locality are likely to be useful in predicting the results of other questions in the same locality. As this locality is increased the questions become less likely to be relevant to the target question. Not only will these questions add little information, they may also obscure the relevant information from the training algorithms.

Another potential reason for the increasing error rate is that as the window size increases there is less usable data, per observed datapoint, to train on. This is shown in Figure 4.6. Less training data means the algorithms have less time to adjust the weights used to predict scores.

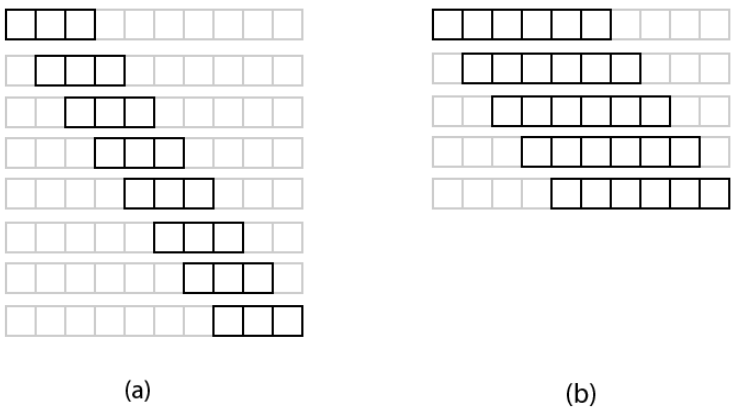

Figure 4.6: (a) shows how much data can be used with a moving window of size 3 (8 pieces of data), (b) shows the same for a window of size 6 (5 pieces of data).

Figure 4.5 shows the lowest error rate when considering 4 questions - interestingly this is the same as the moving window approach. The error rate generally increases as more questions are considered although the behaviour is erratic. The increase in error rate might be attributed to one of the reasons for the increase in error rate of the moving window approach: as more questions are considered, less of them are relevant.

One factor which might account for both the increase in the error rate and the erratic

behaviour is that, in general, this approach will have much less training data than the moving window approach.

As seen in figure 4.6 the moving window approach can, for one model, train multiple times on the same piece of data. The reason it can do this is that it does not need to consider the identity of the question. However, retaining the identity is exactly how the identity based model differs from the moving window model. For this reason it can only train once on each piece of data, whereas the moving window approach might receive 20 pieces of data from one observed item.

4.4.1.4 Conclusion of Linear Models

One interesting feature of the results so far is that all the linear models behave very similarly on both data representations. Since the graphs do not give any obvious indications of the best model, the averages have been calculated for each model for each data representation, shown in Table 4.7.

The lowest average is shared by Linear Regression, Ridge Regression and Elastic Net. Since Ridge Regression and Elastic Net are Linear Regression with added regularisation, the simpler option will be chosen. Therefore the best Linear model is Linear Regression using the Moving Window approach.

Models	Moving Window	Identity based
Linear Regression	43.15	44.07
Ridge Regression	43.15	44.09
LASSO	43.18	43.55
Elastic Net	43.15	44.11

Figure 4.7: Average error rate for the linear models on two different data representations

4.4.2 Support Vector Machines

A Support Vector Machine is an algorithm that attempts to find a maximum margin hyperplane, which is defined by the *support vectors*: the vectors which lie the same distance (margin) away from the hyperplane. This margin is maximised so that the hyperplane lies as far away from each class as possible. A new data point is labelled depending on what side of the hyperplane it lies on. Figure 4.8 shows an example

hyperplane on two dimensional data, although SVM can work on any number of dimensions.

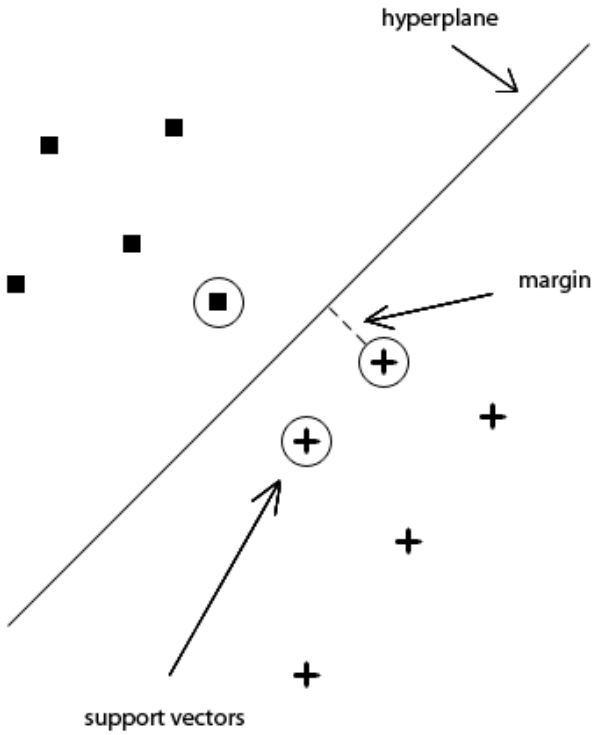

Figure 4.8: A maximum margin hyperplane that separates two classes in a two dimensional space

4.4.2.1 Kernel Functions

The simplest, linear Support Vector Machines attempt to find a maximum margin hyperplane using a method which requires calculating the dot product between vectors. In the linear case, the standard dot product is enough to allow the data to be separated. However, if the data is more complex it may not be the approriate function to use. Changing this dot product to be some other function (which must take two vectors and return a scalar value) is what is known as the “kernel trick”. The candidate functions are known as kernel functions, and allow for the SVMs to be applied to non-linear data.

4.4.2.2 Optimisation

Scikit provides 4 preset kernel functions: linear, polynomial, rbf (radial basis function) and sigmoid. In the case of polynomial and sigmoid kernels, an additional parameter, "degree", allows the specification of the degree of the kernel. The degree of a polynomial is the highest degree of its terms. In the case of a kernel function a higher degree means it can model more complex data. An appropriate way of finding good values for this parameter is to start at 1, and increase until the estimated error ceases to improve [29].

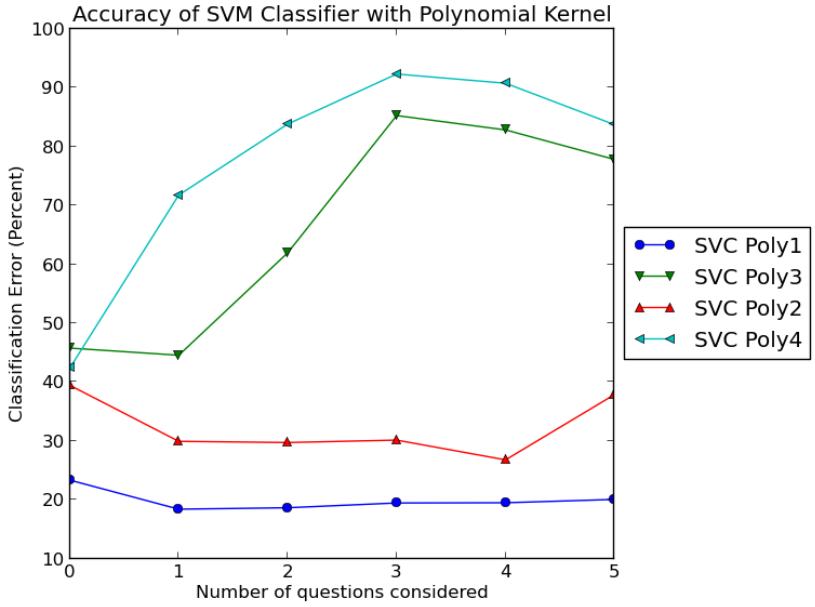

Figure 4.9: The performance of a SVM classifier using a polynomial kernel with different degree values

Figures 4.10 and 4.12 show the effect of changing the degree of the sigmoid kernel. In both cases, changing the degree has no effect on the performance of the model. Figures 4.9 and 4.11 show the effect of changing the degree of a polynomial kernel. Here, changing the degree has a significant affect. In both cases the best value is 1, with the other values performing significantly worse.

4.4.2.3 Results

With the optimisations completed, the different models could be compared against each other. Figure 4.13 and Table 4.14 show the results for the moving window ap-

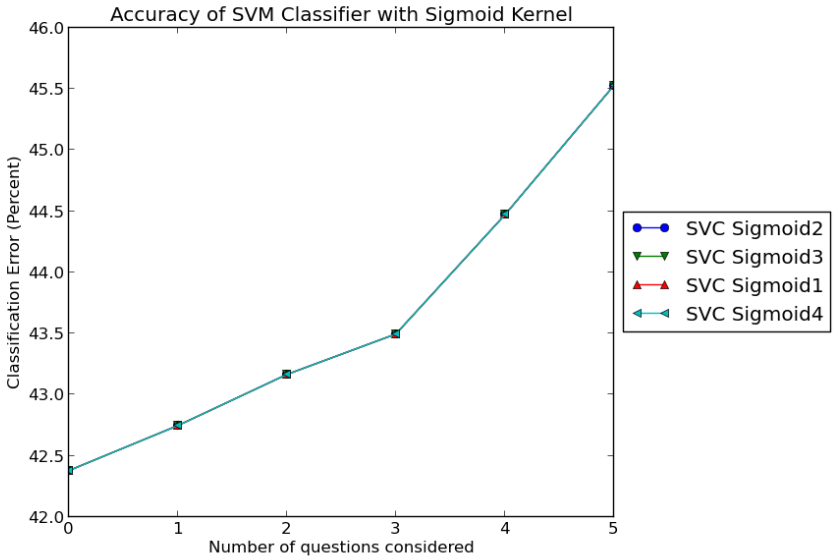

Figure 4.10: The performance of a SVM classifier using a sigmod kernel with different degree values

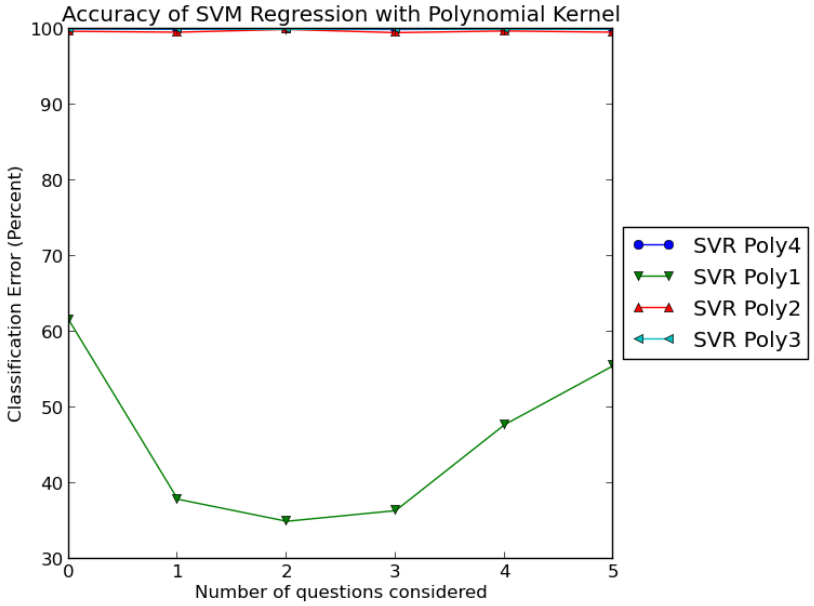

Figure 4.11: The performance of a SVM regression using a polynomial kernel with different degree values

proach. Figure 4.15 and Table 4.16 show the results for the identity based approach. The moving window approach has three similarly high performing models: SVC Linear, SVC Poly, SVC RBF. These models show a very similar performance in the iden-

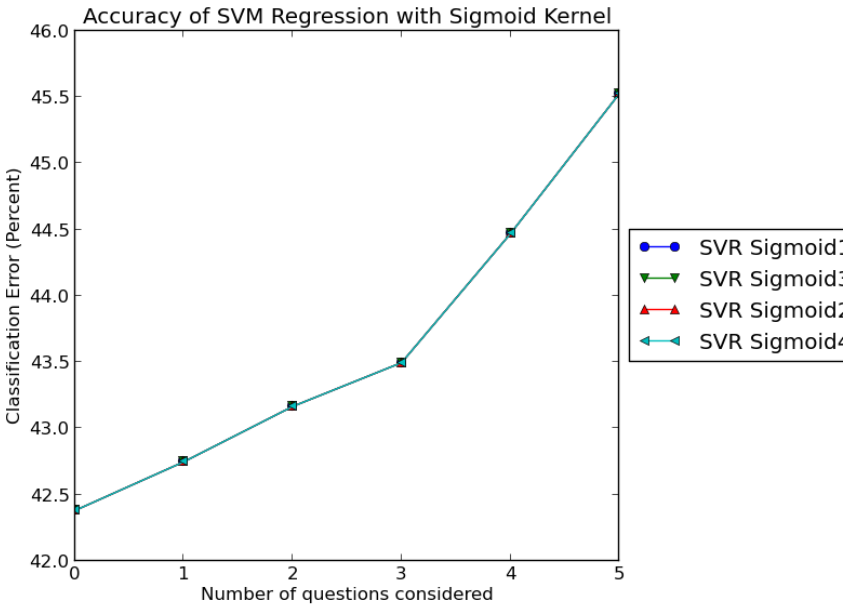

Figure 4.12: The performance of a SVM regression using a sigmoid kernel with different degree values

tity based approach, although as with all models their performance is more erratic.

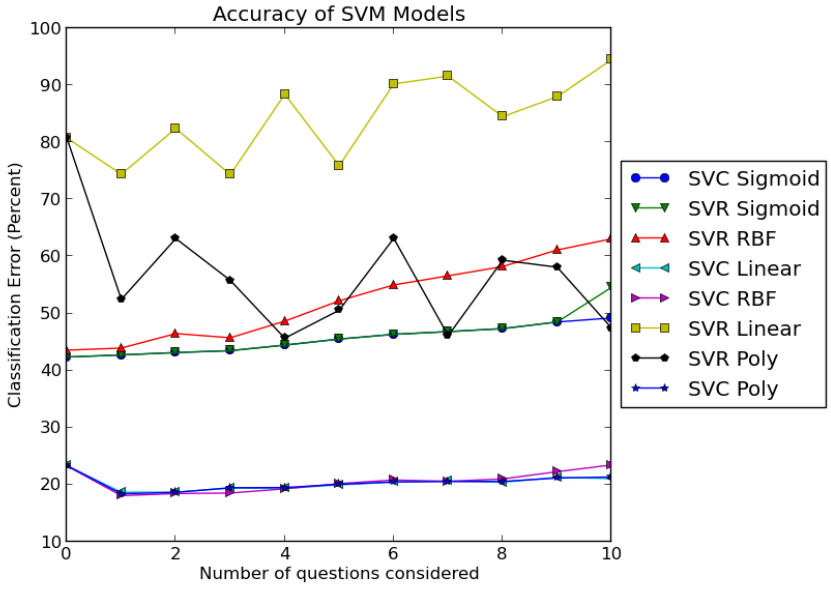

Figure 4.13: The performance of various SVM models

Models	Average Error
SVC Linear	20.30
SVC Poly	20.33
SVC RBF	20.55
SVC Sigmoid	45.46
SVR Sigmoid	45.95
SVR RBF	52.25
SVR Poly	56.60
SVR Linear	84.10

Figure 4.14: Average error rate for SVMs using the moving window approach

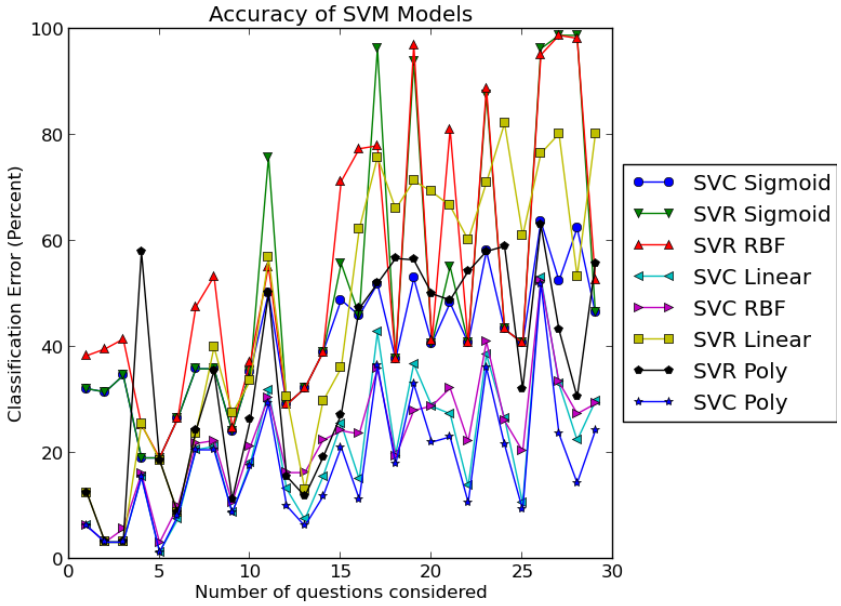

Figure 4.15: The performance of various SVM models

4.4.2.4 Conclusion

3 models perform very similarly on both approaches which makes choosing between them more difficult, although it is at least clear that it should be a classifier. Due to having the lowest combined average, the model chosen as the best SVM is classification with a polynomial kernel of degree 1.

Models	Average Error
SVC Linear	20.55
SVC Poly	17.84
SVC RBF	22.37
SVR Poly	35.56
SVC Sigmoid	40.61
SVR Linear	46.12
SVR Sigmoid	49.90
SVR RBF	53.43

Figure 4.16: Average error rate for SVMs using the moving identity based approach

4.4.3 Artificial Neural Network

The ANN implementation used for this project is provided by the Pybrain library. An ANN takes the inputs, passes all the inputs to each unit in a ”hidden” layer and the outputs of each unit in the hidden layer are passed to the output units. The units in the hidden layer are artificial neurons and can be of various types. For example, a neuron could be the weighted sum of the inputs. Figure 4.17 shows how an ANN suitable for this problem might look.

Since Pybrain is a library which focuses specifically on ANNs it has even more customisation than scikit-learn does for its models. The two parameters investigated are the type of units in the hidden layer and the number of units in the hidden. Changing these parameters allows the model to be as complex as the problem requires.

4.4.3.1 Optimisation

Pybrain provides many types of hidden layer, including GaussianLayer, LinearLayer, SigmoidLayer and TanhLayer. Due to technical issues which will be discussed in Chatper 5 the layers chosen were SigmoidLayer and TanhLayer.

Figure 4.18 shows how different parameters effect the error of the model. The degree parameter for each hidden layer type will be decided by choosing the degree at which each model achieves the lowest error rate. TanhLayer: 130, SigmoidLayer: 180.

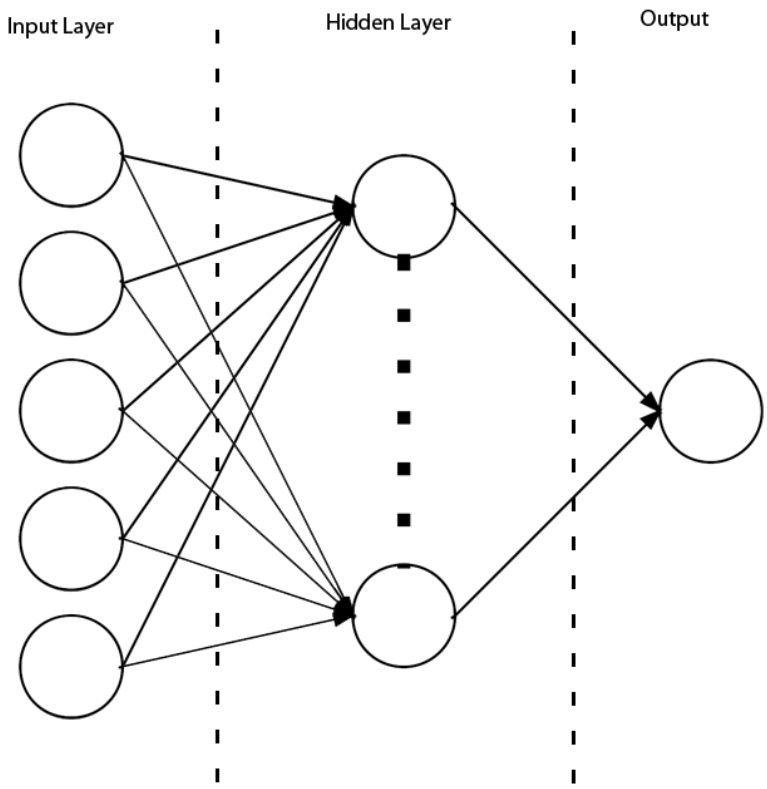

Figure 4.17: The topology of an Artificial Neural Network with 5 inputs, one hidden layer with an unspecified number of units and one output unit.

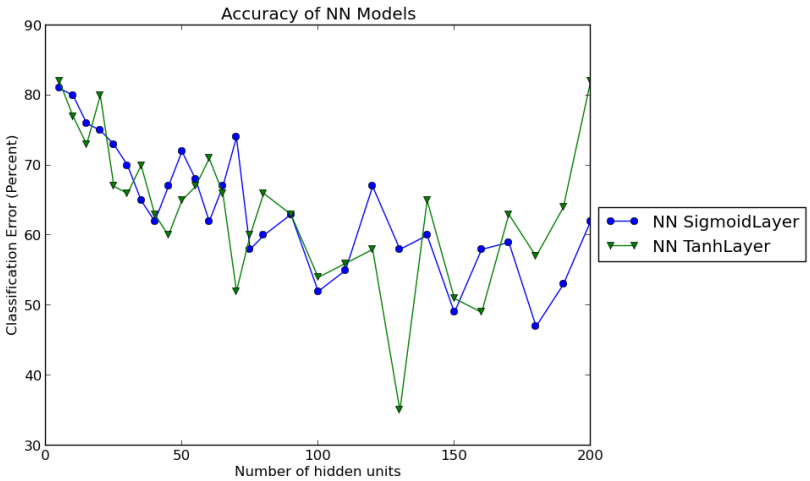

Figure 4.18: The effect of changing the number of hidden units on Neural Networks with two different hidden layer types.

4.4.3.2 Results

Using the parameters determined in the optimisation section, the models are trained and tested on each data representation.

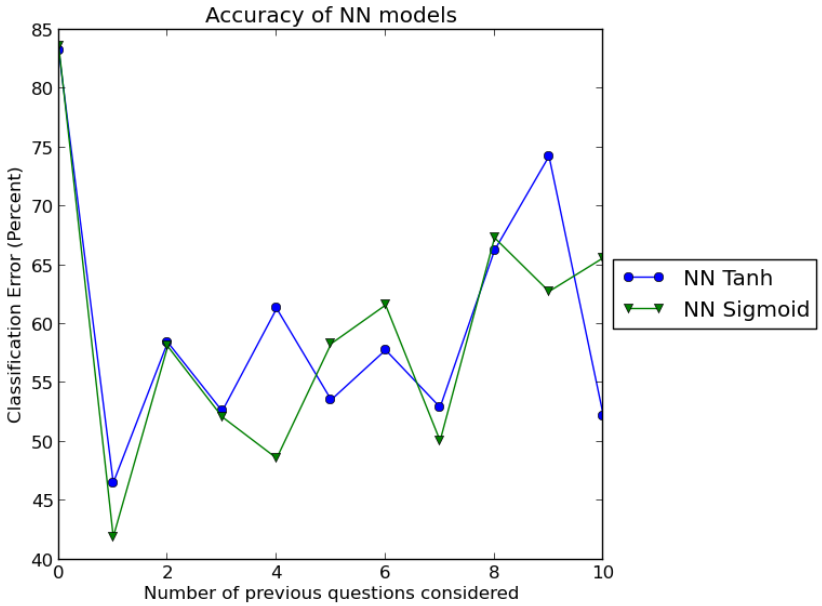

Figure 4.19: The performance of two ANNs with different hidden layer types on the moving window data representation.

Figure **??** shows little difference between the models, but on average TanhLayer performs slightly better. Both models are also quite erratic on this data representation, which differs from previous models which are generally quite consitent.

Figure **??** shows more difference between the models, with SigmoidLayer generally outperforming TanhLayer, with a few error rates which are very low.

4.4.3.3 Conclusion

Due to the different models being best for different data representations, both models will be used for the respective representations when choosing the final model.

4.4.4 Final Model Selection

The final task is to choose the model that is going to be used in the system. To do this, each of the models are trained and tested on both data representations. Each model uses the parameters determined through optimisation.

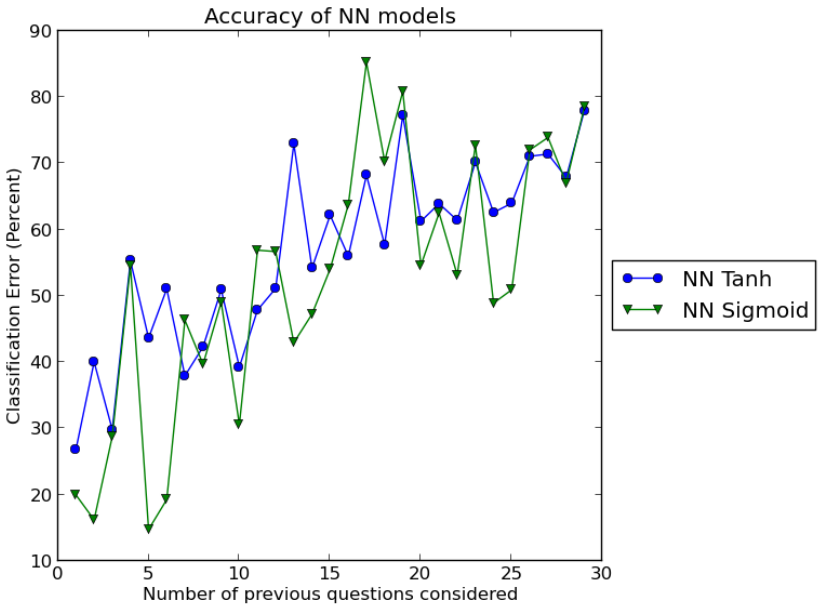

Figure 4.20: The performance of two ANNs with different hidden layer types on the identity based data representation.

4.4.4.1 Baseline

To see if the machine learning actually learns anything that a simple estimate could not have achieved, a baseline is used. The baseline used is simple and intuitive: take the average of the available features. So for the moving window approach, if there are 5 exercises in the window making the feature vector: (60, 70, 80, 90, 100) then the prediction would be 80. By measuring the accuracy of the models against this baseline we can see if performing machine learning is worthwhile.

4.4.4.2 Results

The models from the previous sections are trained and tested alongside the baseline for both data representations.

Figure 4.21 shows the error of the models on the moving window approach. A clear ordering can be seen in which only the support vector machine performs better than the baseline.

Figure 4.22 shows the error of the models on the identity based approach. All classifiers show a much less consistent performance on this data representation though the general trend remains the same as in the previous approach.

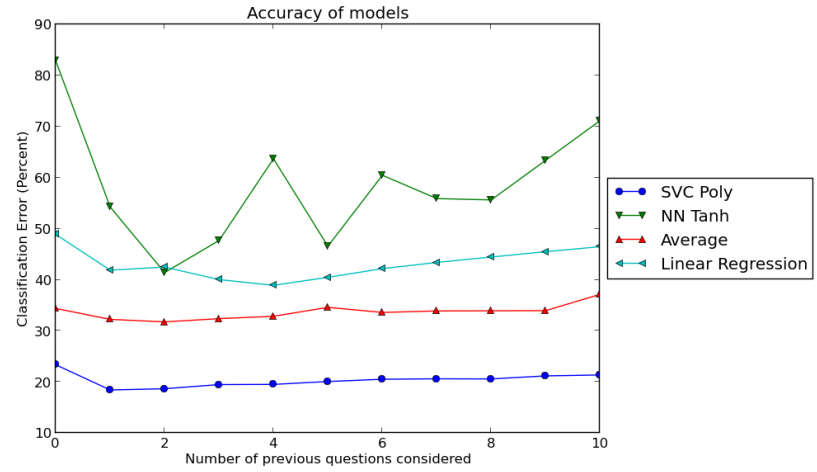

Figure 4.21: The performance of the optimised models using the moving window approach

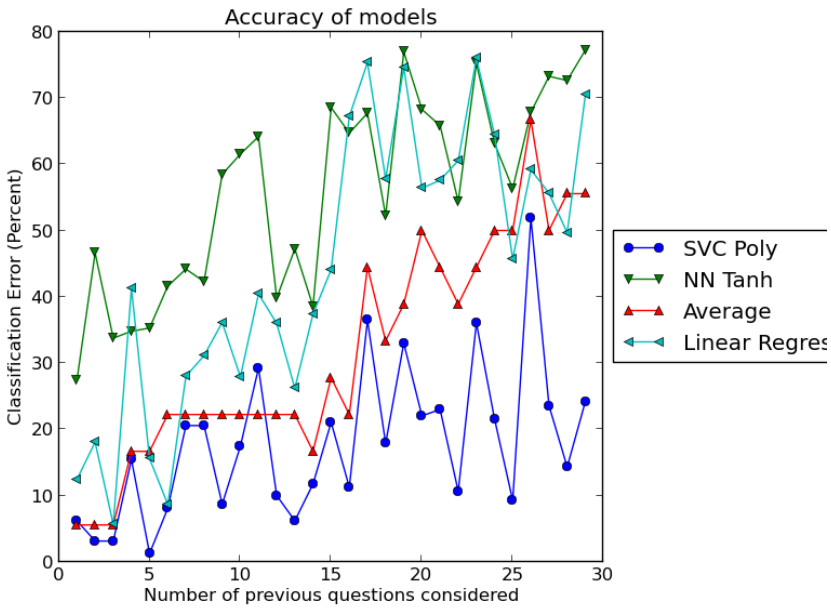

Figure 4.22: The performance of the optimised models using the identity based approach

4.4.4.3 Conclusion

Through a process of optimisations the best models were chosen for each machine learning algorithm. These models then had their performance measured against a baseline, which demonstrated that only the support vector machine performed better than

simply taking the average of the available submissions. The moving window approach maintains a consistent performance across all the questions. The identity based approach shows erratic behaviour which sometimes performs better than the moving window approach but sometimes performs worse. On average the models perform quite similarly and due to the implementation advantages of the moving window approach (Chapter 3) the final model is a Support Vector Classifier using a polynomial kernel of degree 1 on the moving window approach.

The optimal model has an average error rate of around 20%. It is hard to say if this is an adequate accuracy because the ideal error would obviously be 0%, and there is no evidence to say at what error rate the feedback starts becoming less effective. However, it seems there is at least room for improvement, as the identity based approach did sometimes achieve a significantly better error rate.

Chapter 5

Implementation

5.1 Introduction

In the previous chapter a model was chosen from many other models by measuring the error rates. Graphical mockups were created for the visual design of the feedback. This chapter will explain the process through which these two elements were implemented. The implementation required understanding of how the Infandango system works, specifically how it renders webpages. The completed system gives a visual display for the predicted score retrieved from the model.

5.2 Language and Tools

Two languages are immediate possibilities for implementation: Java[18] and Python[27]. Java was considered because I had the most experience with it and some parts of Infandango are written in Java. Most of Infandango is, however, written in Python with which I also had experience. Due to the emphasis the project has on machine learning, R[22] was another approriate language.

The final decision was to use Python with scikit-learn[20] and pybrain[23] libraries: this provides the simplest integration with Infandango (since the parts with which this will need to be integrated are written in Python) and the libraries provide a variety of machine learning methods.

5.3 Visual Design

The final design is relatively simple and since the system is web based, web technologies like Javascript and CSS were considered. The choice between implementation strategies was based on the ease of integration with Infandango: although Infandango does already use Javascript like the Scriptaculous library[25], CSS is more heavily used. For example, the lab exercise page Infandango displays a single score for an exercise with a background colour determined by the score (red for a bad score, green for a good score). Pure CSS is used here where Javascript could also have been, and so a similar CSS based approach was used for the visualisation.

5.4 Prediction Model

Most of the code used to choose the model in Chapter 4 can be used here: loading data, parsing data, training the model and performing a prediction. The final remaining step is allowing Infandango access to the model, a problem for which two solutions were immediately considered.

The first solution is to load the data and train the model when Infandango first starts. This object would then passed around to the appropriate place, staying in memory and being used whenever it is needed. Some problems with this architecture are:

1. It makes changing the model during runtime more or less impossible without restarting Infandango.

2. The data used for training would also need to be stored somewhere, which could require a separate database running alongside the active database.

3. It could potentially cause some coupling between the code used for training the model and the Infandango startup procedure, making future adaptations more difficult.

The second solution is training the model beforehand and storing/loading this model to be used when necessary. A problem with this approach is that loading the model must occur during runtime which could potentially slow the loading speed of the web page. However, this approach does overcome some of the problems of the previous approach:

1. The model can be changed easily, providing it uses the same loading interface.

2. Only the model needs to be stored, not all the data used for training.

3. Any model and training procedure can be used, as long as the same call can be made to make a prediction.

The final implementation uses the second solution: training procedures discussed in chapter 4 to train the model and serialises it using the standard python module, pickle[21].

5.5 System Integration

When a page is requested the request is passed to Django. Django will then be given the necessary information from the database to fulfil this request. Each page has methods which generate dynamic content for that page (e.g. retrieve information from the database). Since the progress bar is attaching to the sidebar — an item which appears on every page — then manually adding the prediction code to every page would take time, would be very difficult to change and is generally undesirable. Instead, Django offers "Context Processors"[7] which is a way of hooking methods into the call that loads every page, allowing this content to be used by any page without telling every page explicitly.

For example, the following function is a current context processor in Infandango.

```
def labSheetListMenu(request):
  return {"lab_sheets": LabSheet.objects.all() }
```

This first half, `"lab_sheets"` , is the name webpages can use to refer to the object. The second half, `LabSheet.objects.all()`, is the object referred to by the first half, and is a database request. This context processor allows all webpages to access the list of lab sheets, which is needed for the navigation sidebar.

5.6 Technical Issues

5.6.1 Neural Network Hidden Layers

Although Pybrain provides many different types of hidden layers only two were used: Sigmoid and Tanh. All attempts with other layers would lead to an error due to an overflow occuring, perhaps caused by the amount of data used. These errors may not happen for Sigmoid and Tanh layers as they are squashing functions.

5.6.2 Slow Neural Networks

The Pybrain library was used due to it being written in Python and its ease of use. However, a factor not fully considered at first was the speed. Although the speed is not a problem when performing a prediction, it becomes an impediment when performing optimisation and cross-fold validation. In an effort to mitigate this problem, the arac library[5] was used, in conjunction with Pybrain to provide fast implementations of the neural networks provided by Pybrain. Due to the alpha state of the project some additional time was needed after installation in order to debug the library, but a working solution was found which on average reduced neural network training time by 50%.

5.7 Conclusion

Using the Django context processor functionality a function is hooked into every web page request. This function de-serialises the trained model and uses it to provide a prediction based on previous exercise submissions. This prediction is used to determine the colours which will be displayed in each box and this is facilitated through Django. Figure 5.1 shows the control flow of this process. This concludes the implementation of the feedback mechanism, and Figure 5.2 shows a screen capture of the feedback mechanism in Infandango.

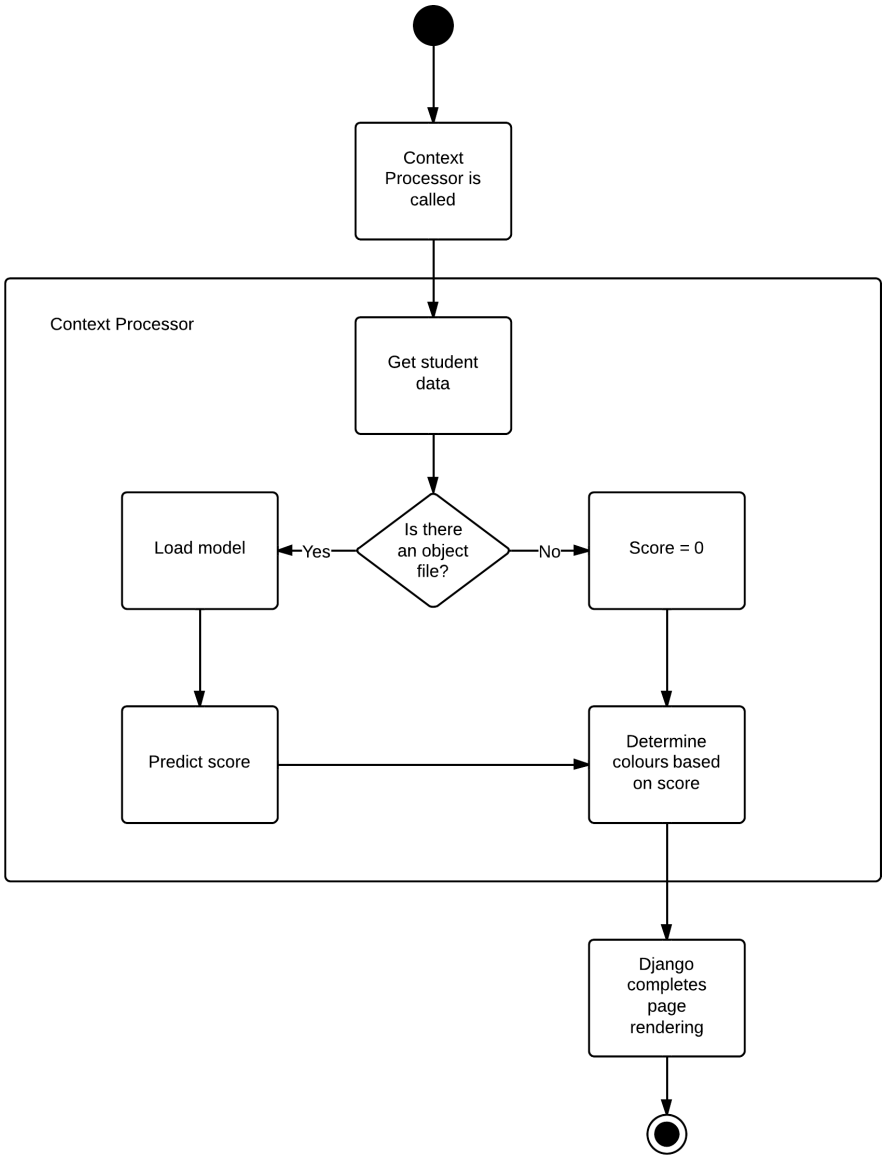

Figure 5.1: A flowchart of the context processor function and how it interacts with Django

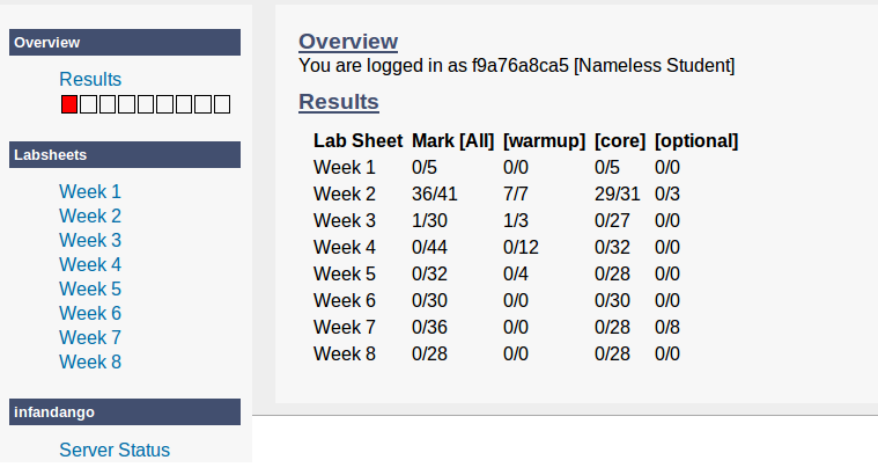

Figure 5.2: A screenshot of what a user might see using the new feedback mechanism in Infandango

Chapter 6

Conclusion

6.1 Evaluation of Goal Completion

At the beginning of the project some objectives were set in the hope that their completion would improve the feedback of the Infandango system.

6.1.1 Research of Literature

The research conducted into the current available literature around the subject provided insight into the applicability of machine learning for generating feedback, the best practices for displaying information and general machine learning practices. This insight made the lack of cumulative feedback in Infandango clear, and provided a solution to this problem using machine learning to model the students' performances.

6.1.2 Feedback Design

The final design was to create a model of the performance of the students using machine learning methods, use the model to predict the future performance of the students and then display that to the user through the Infandango webpage.

It was demonstrated that the final machine learning model was more accurate at predicting the performance of students than the baseline, showing that the machine learning is worthwhile for the sake of accuracy. As is often the case with machine learning, there are many more models and many more optimisations which could be investigated to increase the accuracy.

6.1.3 Implementation and Integration

The implementation and integration is fully complete, and Infandango can run successfully while predicting scores for students. Although this was not tested in a production environment it was tested locally with all parts, including database. The only difference between the test environment and the production environment is the accessibility from outside the computer Infandango is running on. Allowing this accessibility and introducing the system to heavy usage would be the next step in testing the implementation.

6.1.4 User Evaluation

This objective is the only objective which did not complete, due to ethical issues. User evaluation is crucial in determining the effect the feedback has on the performance of the students, measuring the rate of progress and total time spent on each exercise. User evaluation could also allow for interviews with students who have used both systems which could determine if the students found specific uses for the feedback. The results could then be analysed to determine how the mechanism would need to change in order to be fully utilised by students.

User tests had be planned from the beginning, with the current students of the Java course as the intended subjects. The first plan was to split the class up so that roughly 10% would use the new version and the rest would continue using the version without the added feedback. Measurements could then be made to see how submission rates, average scores and general motivation changed among those students testing the new system. However, this required the implementation to be completely early than had been planned (to allow deployment to happen at the beginning of the semester) so another method of testing was to be used.

The second method of testing would, during the middle of the semester, ask students of the course to volunteer to test the system. This version of the system would have the new feedback added and would be run on a self-managed server within the university network. They would do the same exercises as the main system, but they would use the new system and submit their solutions to both systems. However, at a late stage in the process objections were raised about data protection issues due to the submissions also being kept on the self-managed server. Unfortunately theses obstacles could not be overcome in the remaining time.

6.1.5 Overall Feedback Improvement

Most of the original objectives were completed, and an additional source of feedback has been added to Infandango. The method can also be extended to work with other features, allowing the feedback to be more general and potentially give more qualitative feedback.

6.2 Further Work

One result from the literature exploration is the potential for feedback based on more than just scores. The current implementation only uses scores but any data which can be numerical, ordinal or categorical can be used as well, since the machine learning models are flexible in their inputs. This allows for a lot of potential research into how different features of the models affect the outcome. For example, another feedback widget could be created which uses the submissions times as features instead. This could then try to predict the scores someone receives (someone who hands in exercises late tends to receive less marks) or it could try predict the time taken before their next submission. If this time is too long it could tell the student to start the exercises sooner.

The only use of the predictions currently is one instantaneous display. One potential area for further work is to accumulate this data and use it in some way to represent a student's progress across the whole semester. For example, a graph could be created plotting the predicted score against the days/weeks of the semester. This graph may provide insight into what questions a student struggled with and therefore which topics might be worth revising.

Some areas which could be investigated include analysis (both manual and machine) of the submitted code to look for common errors, qualitative analysis to judge some more general aspect of the code rather than accuracy using JUnit tests or looking at any work done in trying to make JUnit error messages more understandable to the beginner. There are many areas which are worth exploring to try and improvement the feedback of Infandango but covering such a wide range of topics in the period of time given was unfeasible.

Appendix A

K-means Clustering

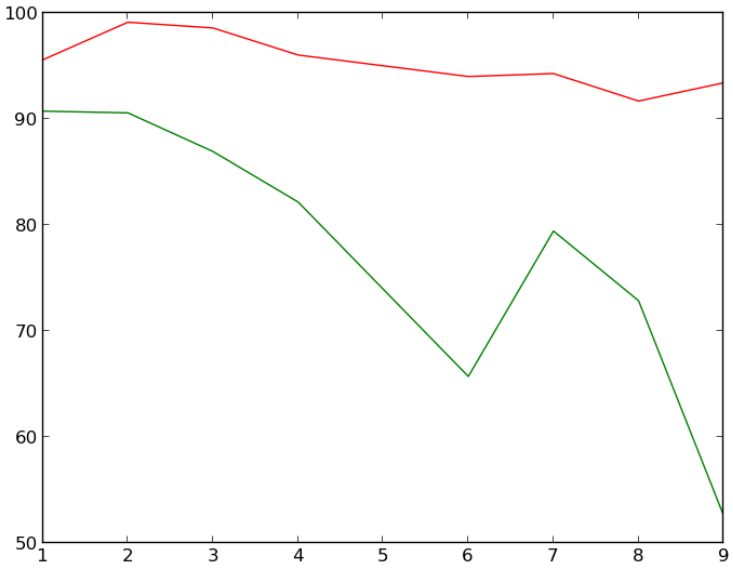

Figure A.1: A plot of the centroids of the clusters discovered through k-means clustering with 2 clusters. x-axis is the features, y-axis is score (%)

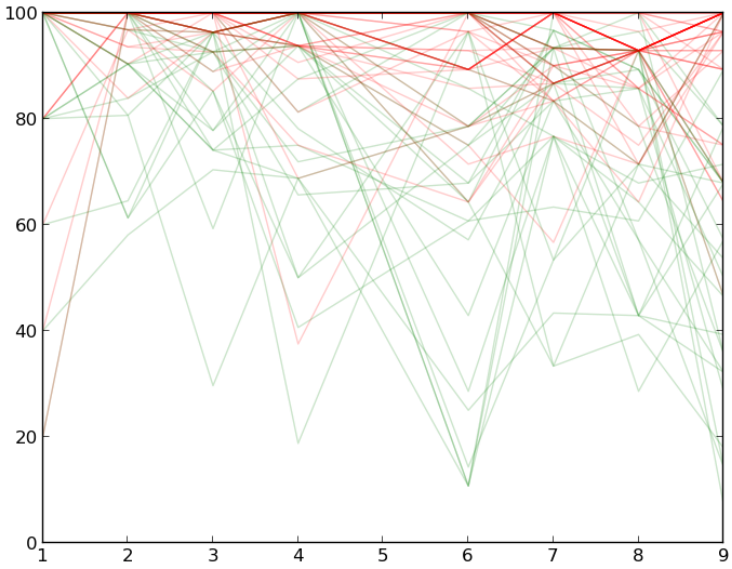

Figure A.2: A plot of all the students that submitted for each exercise. They are plotted with an opacity of 0.2 and the colours represent the cluster which they belong to after performing k-means cluster with 2 clusters. x-axis is the features, y-axis is score (%)

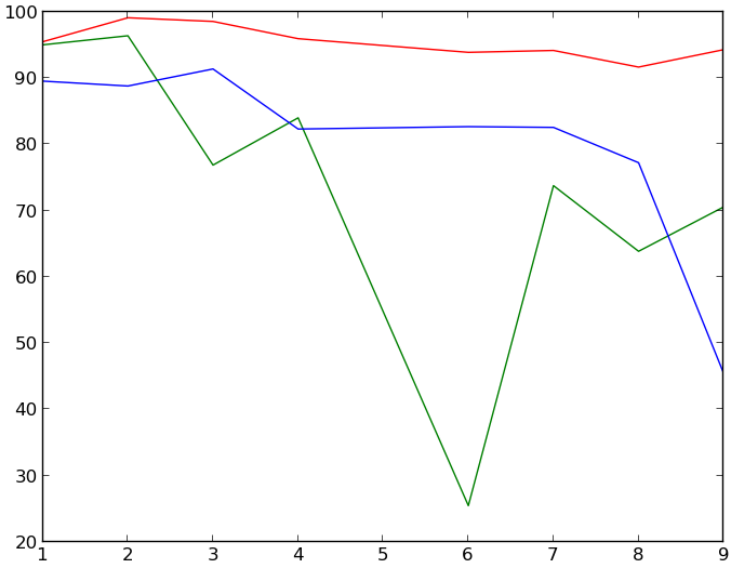

Figure A.3: A plot of the centroids of the clusters discovered through k-means clustering with 3 clusters. x-axis is the features, y-axis is score (%)

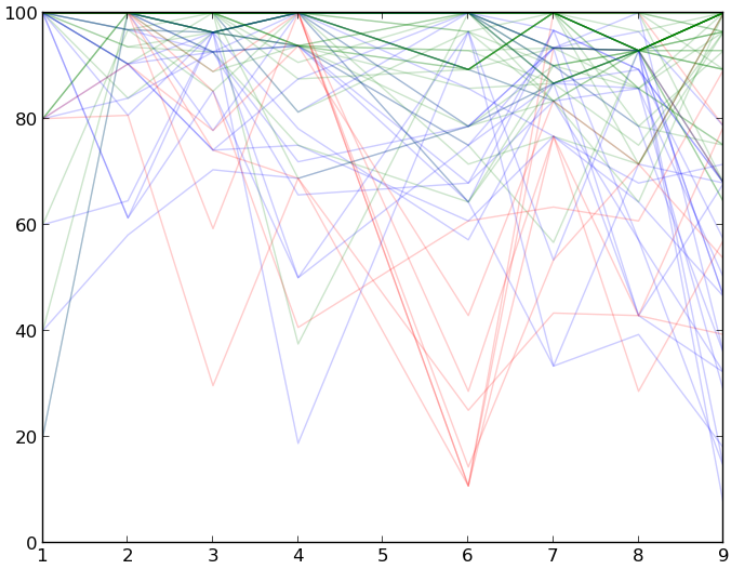

Figure A.4: A plot of all the students that submitted for each exercise. They are plotted with an opacity of 0.2 and the colours represent the cluster which they belong to after performing k-means cluster with 3 clusters. x-axis is the features, y-axis is score (%)

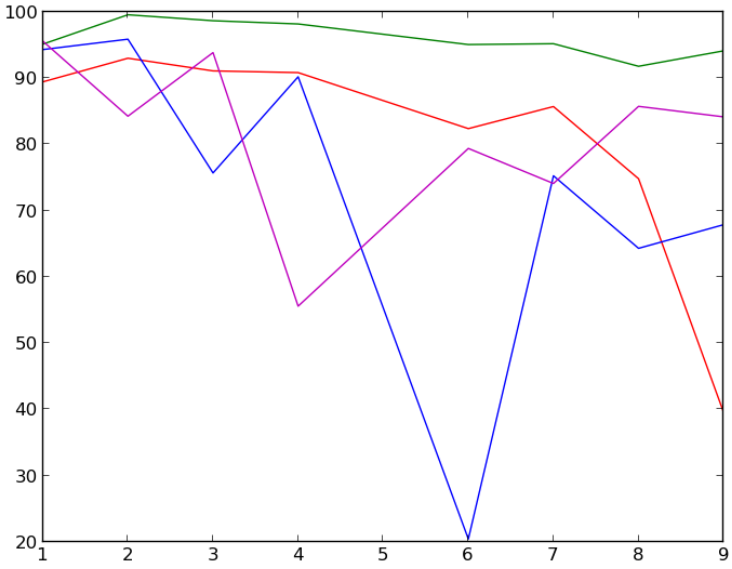

Figure A.5: A plot of the centroids of the clusters discovered through k-means clustering with 4 clusters. x-axis is the features, y-axis is score (%)

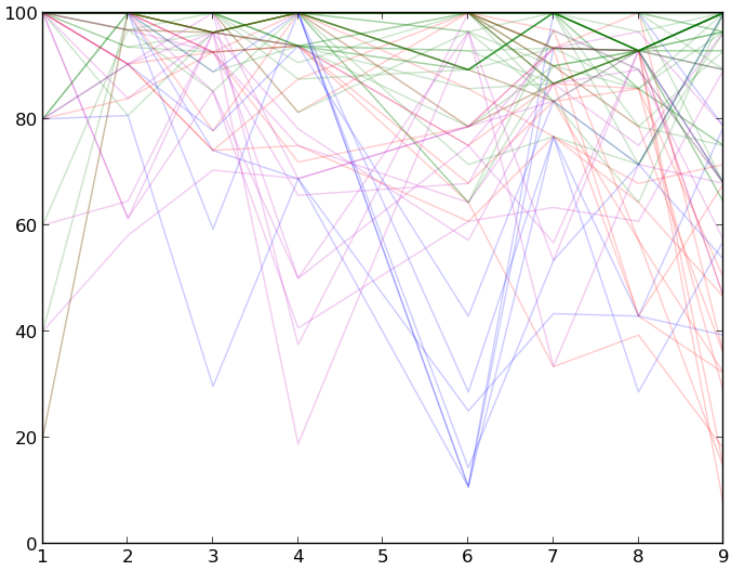

Figure A.6: A plot of all the students that submitted for each exercise. They are plotted with an opacity of 0.2 and the colours represent the cluster which they belong to after performing k-means cluster with 4 clusters. x-axis is the features, y-axis is score (%)

Bibliography

[1] Khan Academy. Khan academy website. https://www.khanacademy.org/, 2013.

[2] Khan Academy. What is Khan Academy? http://khanacademy.desk.com/customer/portal/articles/337790-what-is-khan-academy-, 2013.

[3] S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. *Stat. Surv.*, 4:40–79, 2010.

[4] David Arnow and Oleg Barshay. WebToTeach: an interactive focused programming exercise system. In *Conference on Frontiers in Education*, volume 1, 1999.

[5] Justin Bayer. Arac wiki. https://github.com/bayerj/arac/wiki, 2013.

[6] R E. Berry and B A.E. Meekings. A style analysis of C programs. *Commun. ACM*, 28(1):80–88, January 1985.

[7] Django Community. The Django template language: For Python programmers. https://docs.djangoproject.com/en/dev/ref/templates/api/, 2013.

[8] Charlie Daly. RoboProf and an introductory computer programming course. In *Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education*, ITiCSE '99, pages 155–158, New York, NY, USA, 1999. ACM.

[9] Django. Django. https://www.djangoproject.com/, 2013.

[10] Colin A. Higgins, Geoffrey Gray, Pavlos Symeonidis, and Athanasios Tsintsifas. Automated assessment and experiences of teaching programming. *J. Educ. Resour. Comput.*, 5(3), September 2005.

[11] David Hu. How Khan Academy is using Machine Learning to Assess Student Mastery. http://david-hu.com/2011/11/02/how-khan-academy-is-using-machine-learning-to-assess-student-mastery.html, Nov 2011.

[12] David Jackson and Michelle Usher. Grading student programs using ASSYST. In *Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education*, SIGCSE '97, pages 335–339, New York, NY, USA, 1997. ACM.

[13] R. Kaushal and A. Singh. Automated evaluation of programming assignments. In *Engineering Education: Innovative Practices and Future Trends (AICERA), 2012 IEEE International Conference on*, pages 1–5, july 2012.

[14] Ewan Klein. Infandango. https://bitbucket.org/ewan/infandango, 2013.

[15] Nan Li and Kenneth R Koedinger. A machine learning approach for automatic student model discovery. 2011.

[16] Cohen W. W. Sewall J. Lacerda G. Koedinger K. R. Matsuda, N. Evaluating a simulated student using real students data for training and testing. 2007.

[17] Ewan Klein Mike Hull, Dan Powell. Infandango: Automated Grading for Student Programming.

[18] Oracle. Java Website. http://www.java.com/, 2013.

[19] Linus Pauling. *General Chemsitry*. San Francisco USA, 1947.

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

[21] Python. Python Object Serialization. http://docs.python.org/2/library/pickle.html, 2013.

[22] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[23] Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Rückstieß, and Jürgen Schmidhuber. PyBrain. *Journal of Machine Learning Research*, 11:743–746, 2010.

[24] Scikit-learn. Generalized Linear Models. `http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression`, 2013.

[25] Scriptaculous. Scriptaculous. `http://script.aculo.us/`, 2013.

[26] Edward R. Tufte. *Visual explanations : images and quantities, evidence and narrative*. Cheshire, Conn. : Graphics Press, 1997., 1997.

[27] Guido van Rossum et al. Python Website. `http://www.python.org/`, 2013.

[28] James Vaughan. Python Extension. `https://bitbucket.org/ewan/infandango/commits/branch/python_extension`, 2013.

[29] Ian H Witten, Eibe Frank, and Mark A Hall. *Data Mining: Practical Machine Learning Tools and Techniques: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2011.