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 thesis will discuss the work done towards improving the feedback provided supplied to users of the Infandango system. By applying machine learning methods to submission data from a previous year a model is created which can provide a single score to the user, representing their progress so far. A discussion of the efficacy of different machine learning methods will provide reasons for choosing one model over the others. This will then be integrated with the Infandango system, providing a visual mechanism to display the score.

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*)

Table of Contents

Chapter 1

Introduction

Infandango is an open source web-based system for automated grading of Java code submitted by students[10]. 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, justifying primary aspects of the design. Chapter 4 will be reserved for discussing the process through which the final model was chosen, documenting the results of various analyses. Implementation of the feedback device with the current Infandango system will be explained in Chapter 5 and the report will end with a conclusion and summary of the work.

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 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.

2.1.1 Current feedback

The primary source of feedback in Infandango is displayed in Figure 2.1. Each submission 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 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.

2.1.2 Current data

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. This information for one year has been anonymised and made available for use. This has only happened for one year because the questions have changed since previous years and therefore the data would be inconsistent with the current questions.

2.2 Literature

Khan Academy[4] 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[8] written by David Hu about Khan Academy demonstrates that different feedback measures can affect user performance significantly. Khan Academy gives users certain kinds of exercises, for example choosing the approriate position on a number scale. It can then generate endless variations of this problem for the user to continue attempting until they are deemed proficient[1]. The original Khan Academy system required a user to get 10 consecutive exercises of a certain type correct before they can be deemed proficient 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

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

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. Although the current system does not require perfection like Khan Academy did, it is possible that users are more reluctant to move on from exercises in which they get a minor error, and have just less than 100%. Providing encouragement for the user to move on is one aim of this feedback design.

2.2.1 Programming Assessment

In *Automated Evaluation of Programming Assignments*[9], Kaushal and Singh describe various measures used as part of an automated marking system: regularity, efficiency, integrity and accuracy. Notably, Infandango only gives feedback on one of these areas, accuracy. The paper tracks 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. This shows that accuracy is not the only measurement that should be used to provide feedback and these are possibilities to be considered for Infandango.

2.2.2 Visualisation

In his book *Visual Display of Quantitative Information*[11], Edward Tufte discusses methods of efficiently and suitably 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.

Given that data-ink is unavoidable and desirable information, 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, explaining that it should not always be removed.

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

These principles shall be considered when designing the visualisation of the feedback score.

2.2.3 Machine Learning

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

The literature has shown us that machine learning methods can be useful in generating feedback for students, and this can be combined with a visual display for intuitive understanding. The first design decision to be made is the language and libraries to be used when writing the program.

3.1 Language and Tools

Two languages are immediate possibilities for implementation: Java[5] and Python[6]. Java because I had the most experience with it and some parts of Infandango are written in Java. Most of Infandango was, however, written in Python with which I also had experience. Due to the emphasis the project has on machine learning, R[7] was another approriate language. The final decision was to use Python with scikit-learn[3] and pybrain[2] 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.

3.2 Proposed Design

Using the data 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 in Infandango.

3.2.1 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, 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 solution is a binary feature: correct or incorrect

This means a user can keep submitting solutions to an exercise even after they get one solution correct. However, in Infandango once a user gets 100% there is no reason for the user to submit another solution because they have already perfected that exercise. The second difference means our data will be numerical instead of binary and this needs to be considered when designing the model. 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*.

With an idea of how the model was going to work, some exploratory work was done on the data. Although users are encouraged to answer all questions, the do not receive marks directly for each question. This 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.

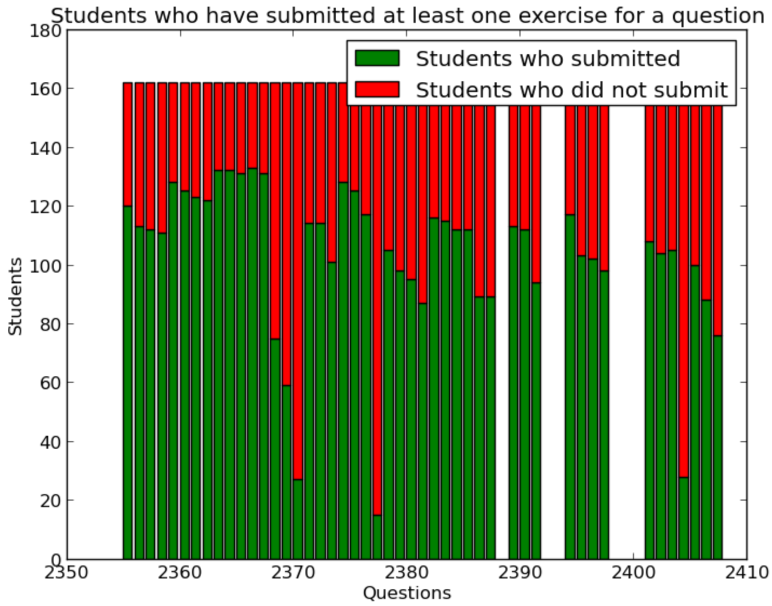


Figure 3.1: Submission rates for all questions

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 a week to get no score for a week. Comparing Figure 3.3 to Figure 3.2 a rise in the amount of 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.

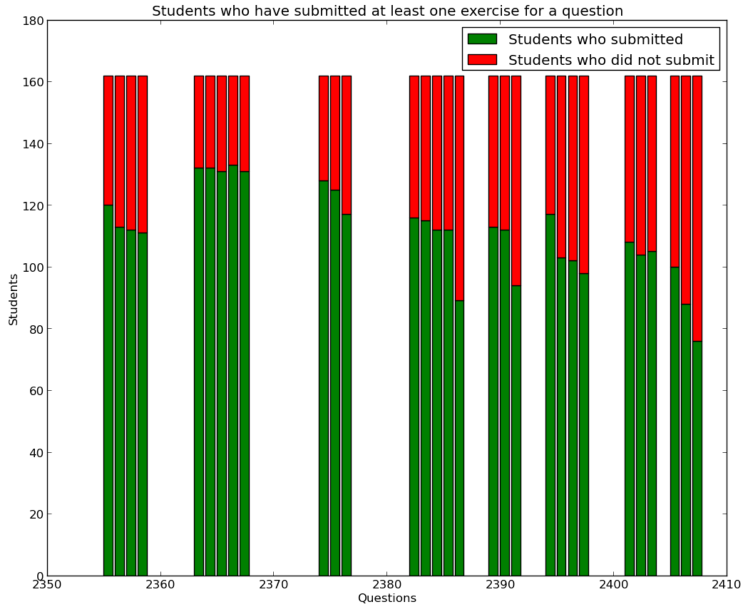


Figure 3.2: Submission rates for only core questions

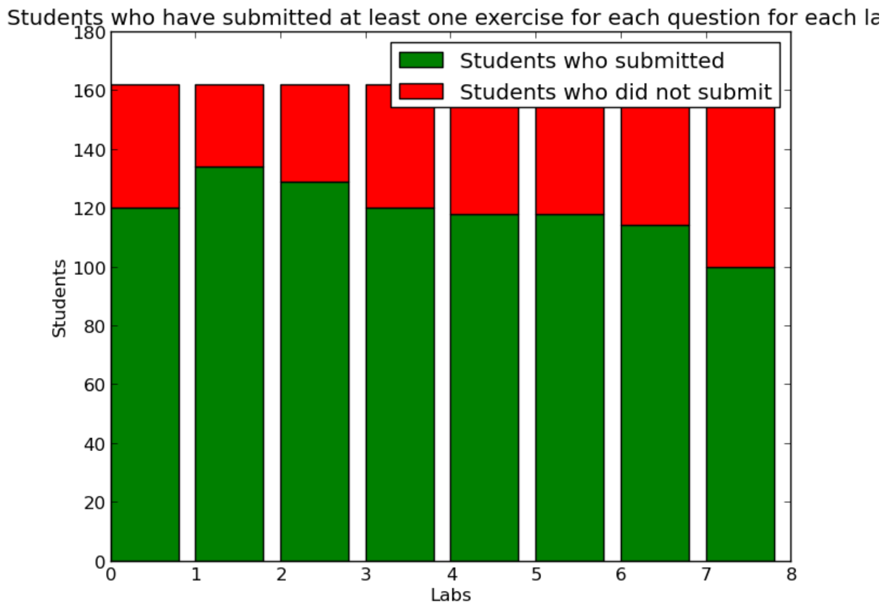


Figure 3.3: Submission rates for at least one question per week

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. Problem 3, however is still present: a new model needs to be trained for each question. A solution to this problem will be discussed later.

3.2.1.1 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.

3.2.1.2 Moving Window

Previously an assumption has been made: the identity of the question is necessary. 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.

3.2.2 Visualisation

3.3 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 to generate the information which is 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 displayed using HTML and CSS after being generated via 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. 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 first step is to use the labelled data to train and test various models. Given the tests results, a model can be selected. However, testing on the same data as was used in training can give results which will not be seen the model is used on new data. Hence it is necessary to split the data into data to be used for training and data to be used for testing. This process is known as cross-validation and is the standard method of error prediction [12]: the models are being validated against the test set. Cross-validation is most often performed more than once on the same dataset and the results are averaged. Doing so reduces the variance in the performance of the models. Now that cross-validation has been performed each model has an error estimate. The final model is selected to have the lowest cross-validation error. However, due to the nature of cross-validation this error estimate may not be entirely accurate: we have deliberately chosen the model with the best cross-validation error, so it is possible that this error estimate is optimistic. To get the final error estimate the model is trained on all of the data used during cross-validation and tested on some separate, previously unseen data. Although it cannot be certain that this model will perform best on new data, it is at least hoped that it will perform similarly to the final error estimate.

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 assigning an index of the input vector to the same question each time (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 is quite wasteful: 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.

4.2 Preliminary models

Deciding on the models to use is a difficult process. However, since ski-kit 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, evaluating how they behave against each other. The models can be split into two categories, depending on the type of data the expect and output.

4.2.1 Classifiers

Classifiers attempt to identify the class a student should be assigned to depending on the input features. Notably, these classes should be discrete but what the models are trying to predict is a continuous percentage. 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 acheived.

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, further accuracy would be lost when the information is displayed. So, in order to use classifiers on this problem, the output feature (the question we want to predict) will be discretised into one of 9 classes, distributed evenly over the range 0-100

4.2.2 Regression

Unlike the classifiers, no adaptations need to be made to the data before supplying it to the regression models.

4.2.3 Model definitions

4.2.3.0.1 Logistic Regression Although the name is misleading, this model is a classifier. It is originially a binary classifier but this problem has been formulated to use 9 classes. The multivariate implementation used by sci-kit uses the one-vs-all method. It can also use either L1 or L2 regularisation.

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.

4.2.3.1.1 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.

4.2.3.1.2 Ridge Regression This model adapts Linear Regression by adding to the minimisation value a penalty depending on the size of the coefficients. Sci-kit allows control of this penalty through the α parameter.

4.2.3.1.3 LASSO This model applies a similar penalty as Ridge Regression except use the L1 norm instead. Sci-kit allows control of this penalty through the α parameter.

4.2.3.1.4 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.

4.2.3.1.5 Cross-validation Sci-kit also provides versions of these models which automatically apply cross-valdiation to determine the best values for their resepctive parameters.

4.2.3.2 Support Vector Machines

A support vector machine, at the simplest level, 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 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.

4.2.3.2.1 Support Vector Classifier Sci-kit 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.

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

4.2.3.3 Artificial Neural Network

This model was chosen because it can model very complex relationships, which many 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 had to be used for this because sci-kit 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 layers.

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 that 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

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 class, not a 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.

4.4.0.0.1 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

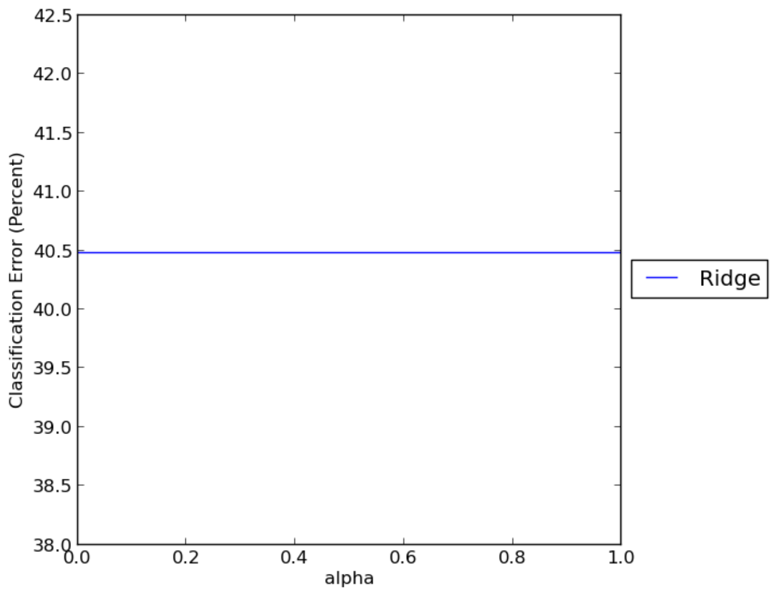


Figure 4.1: The effect on error rate of changing the alpha parameter on Ridge Regression

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.

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 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 increase as more questions are considered which is at first unexpected: if the model knows more about the previous questions

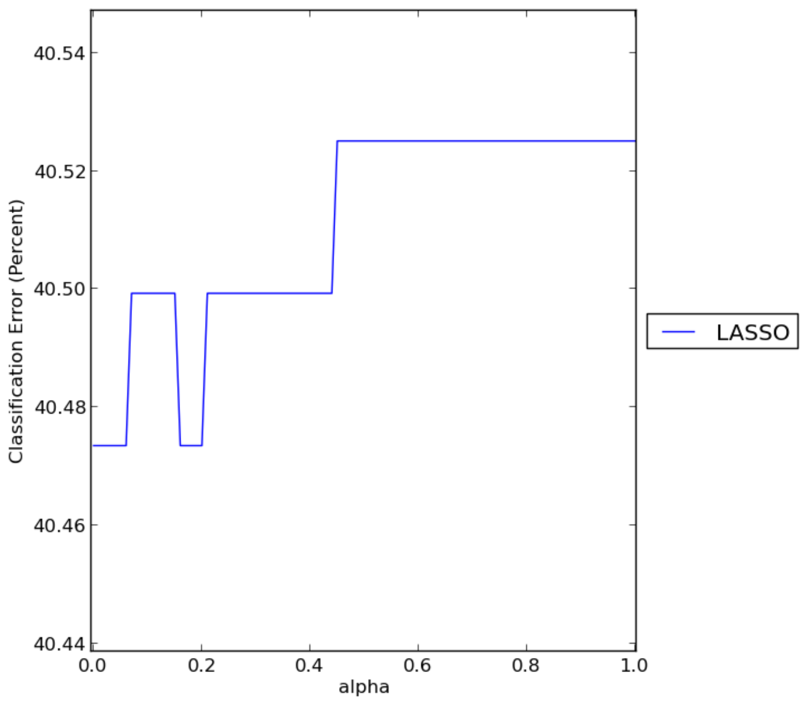


Figure 4.2: The effect on error rate of changing the alpha parameter on LASSO

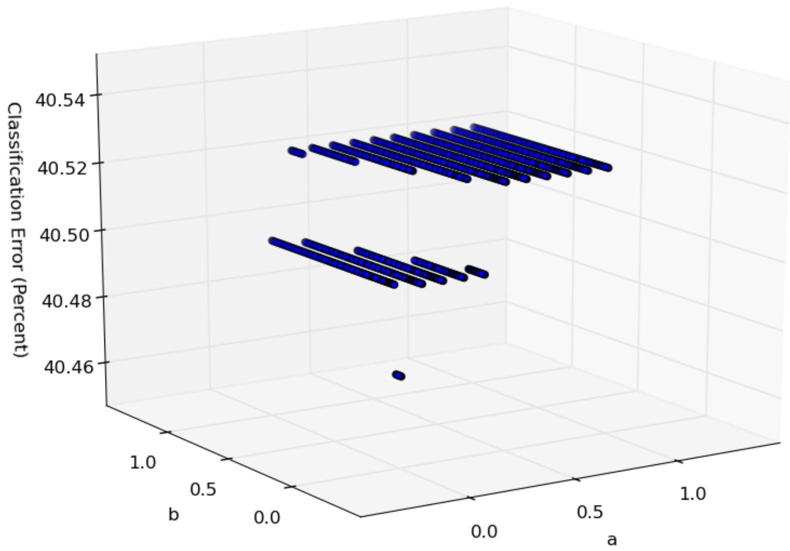


Figure 4.3: The effect on error rate of changing the a and b parameters on Elastic Net

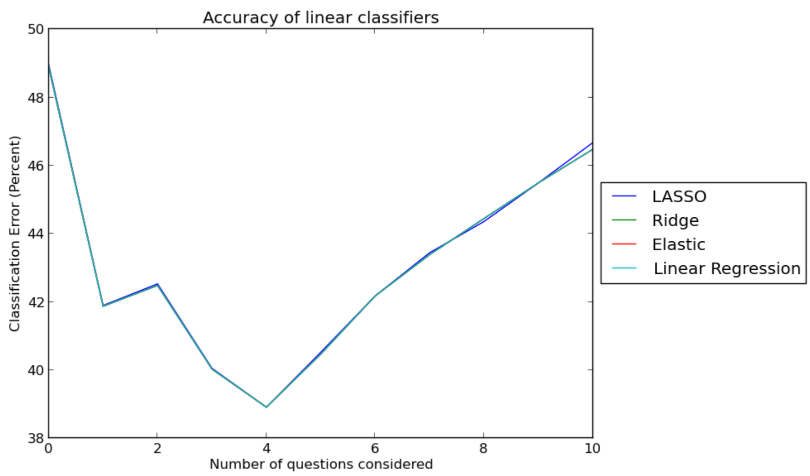


Figure 4.4: The error rates for linear models using the moving window data representation

then it should be able to have a more accurate prediction. One possible explanation is that the new data is not as relevant. Questions in Infandango are grouped - not strictly - by the type of questions they are: a question is likely to build on knowledge of the previous question. For this reason, results of questions within the same locality are

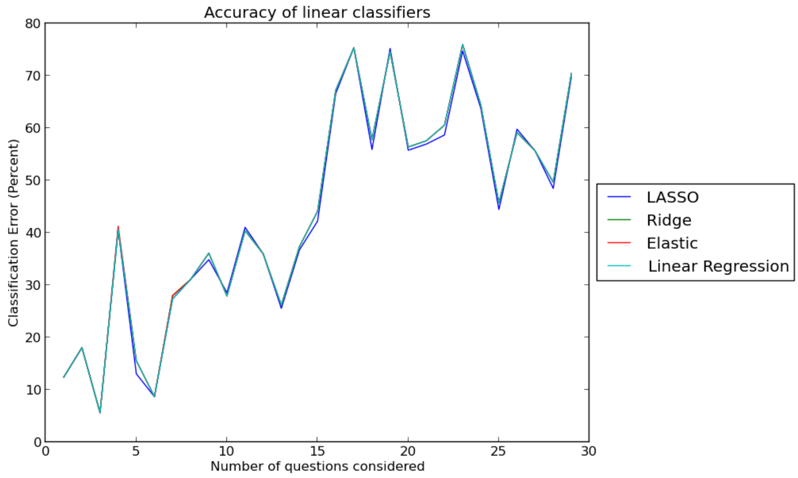
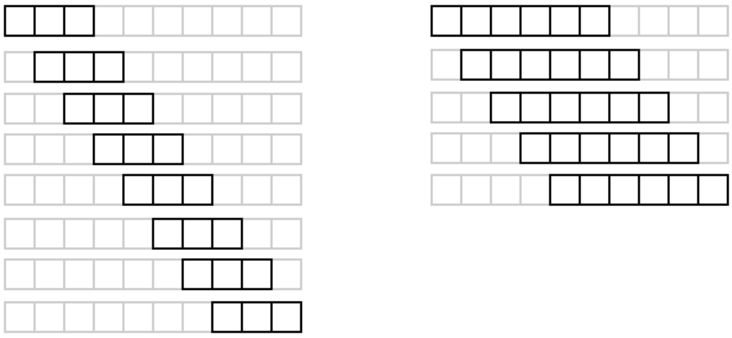


Figure 4.5: The error rates for linear models using the identity based data representation

likely to be useful in predicting the results of other questions in the same locality. As this locality is increased the questions become less and less likely to be relevant to the target question. Not only will these questions *n*ot add information, they may also obscure the relevant information from the training algorithms, hence the error rates increase. Another potential reason for the increasing error rate is that as the window size increases there is less usable data per observed datapoint (student) 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 4.5 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



(a) (b)

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

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, this is exactly what the identity based approach does. 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 **??**.

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	43.1508290455	44.066091954
Ridge Regression	43.1508290455	44.0876436782
LASSO	43.1844660458	43.5512452107
Elastic Net	43.1508290455	44.1091954023

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 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.

4.4.2.1 Kernel Functions

The simplest, linear Support Vector Machines attempt to find a maximum margin hyperplane using an equation which requires using 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,

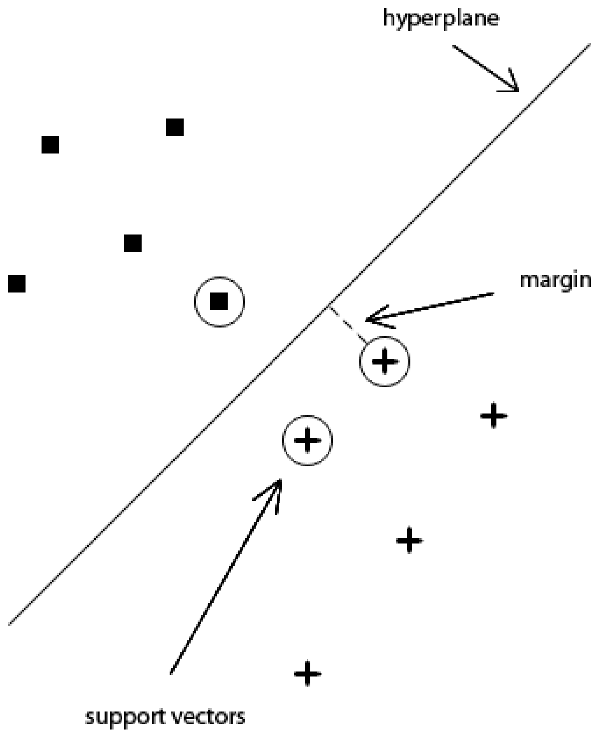


Figure 4.8: A maximum margin hyperplane that separates two classes in a two dimensional space

”degree”, allows the specification of the degree of the kernel. An appropriate way of finding good values for this parameter is to start at 1, and increase until the estimated error ceases to improve [12].

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.

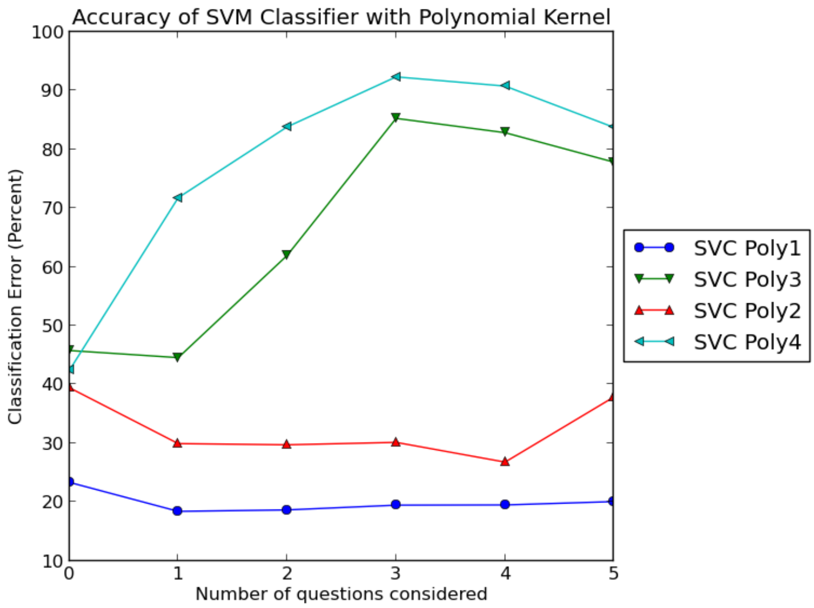


Figure 4.9: The performance of a SVM classifier using a polynomial kernel with different degree values

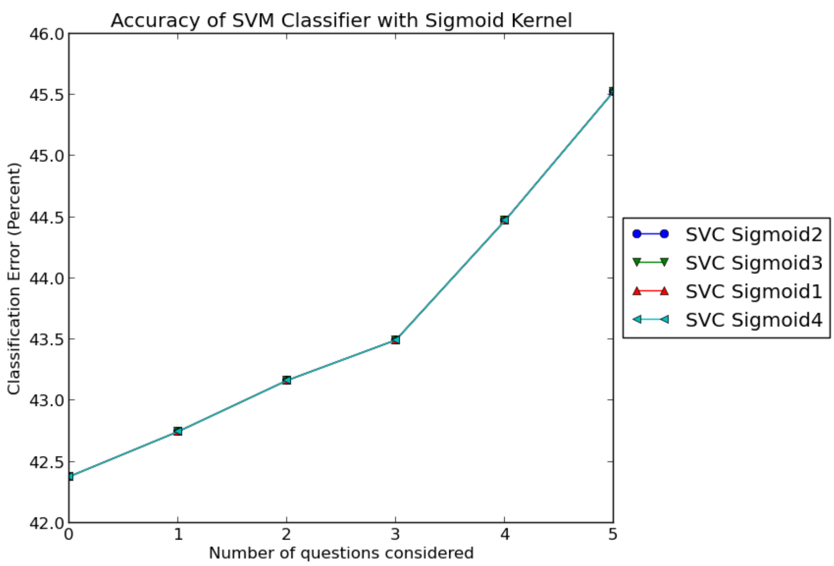


Figure 4.10: The performance of a SVM classifier using a sigmod kernel with different degree values

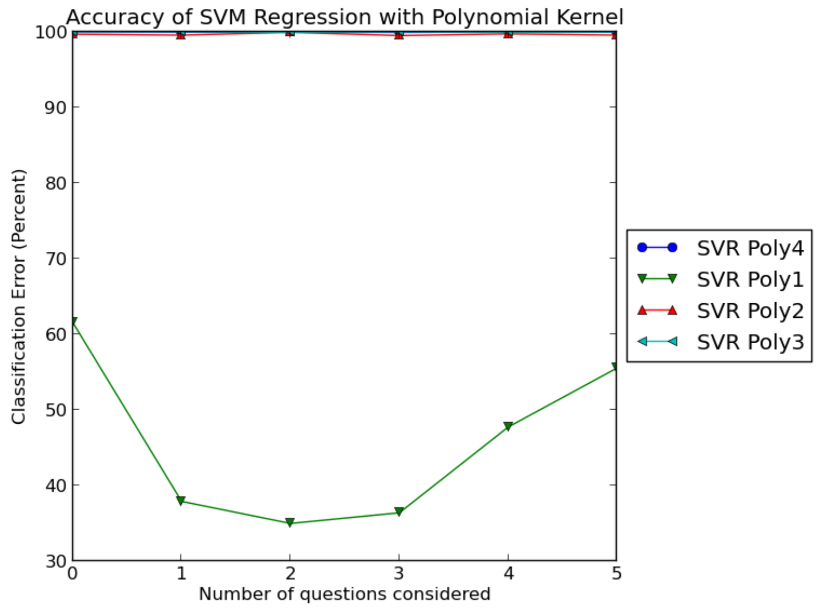


Figure 4.11: The performance of a SVM regression using a polynomial kernel with different degree values

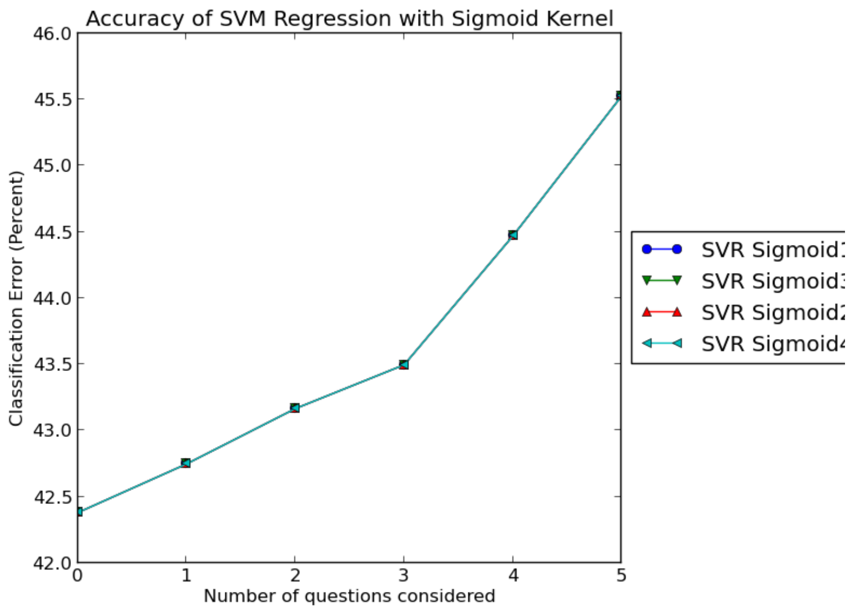


Figure 4.12: The performance of a SVM regression using a sigmoid kernel with different degree values

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 approach. Figure 4.15 and Table **??** do the same for the identity based approach. The moving window approach has three similar, high performing models: SVC Linear, SVC Poly, SVC RBF. These models show a very similar performance in the identity based approach, although as with all models their performance is more erratic.

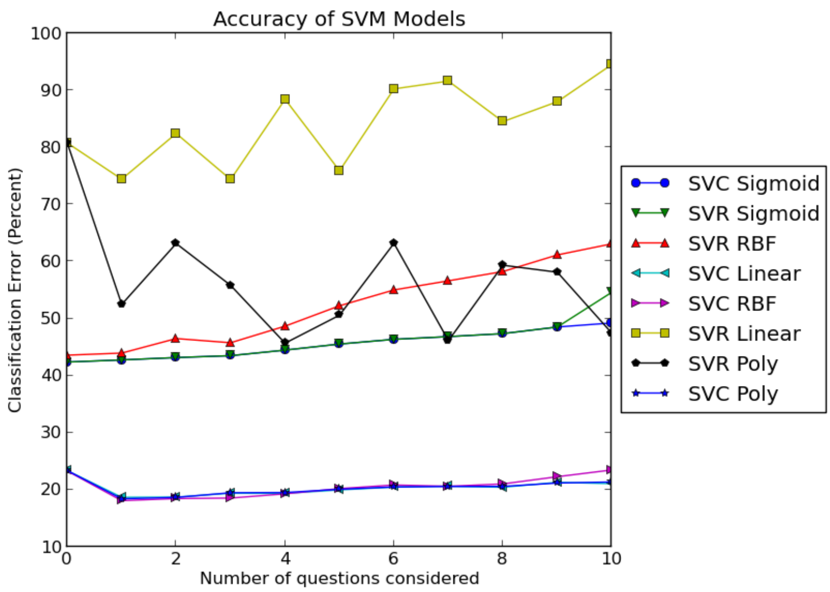


Figure 4.13: The performance of various SVM models

4.5 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	20.3041791411
SVC Poly	20.3313952482
SVC RBF	20.5499438422
SVC Sigmoid	45.4603396387
SVR Sigmoid	45.9489389873
SVR RBF	52.2508272032
SVR Poly	56.6047985482
SVR Linear	84.1041319495

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

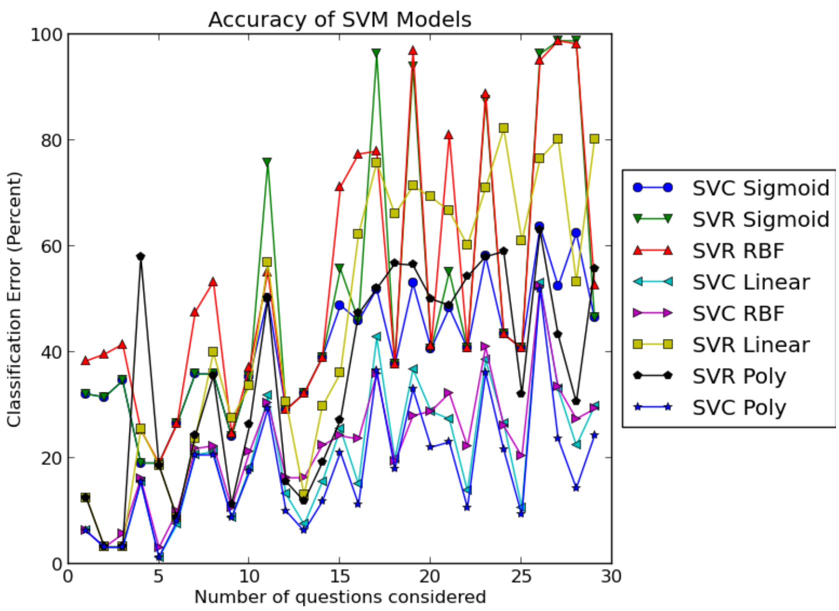


Figure 4.15: The performance of various SVM models

Models	Average Error
SVC Linear	20.5459770115
SVC Poly	17.8424329502
SVC RBF	22.3659003831
SVR Poly	35.5555555556
SVC Sigmoid	40.6082375479
SVR Linear	46.1206896552
SVR Sigmoid	49.8970306513
SVR RBF	53.43151341

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

Chapter 5

Implementation

5.1 Introduction

Following the design process, the two elements were designed separately and then integrated with Infandango. This step 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 Visual Design

The final design is relatively simple and as the system is web based, web technologies like Javascript and CSS were considered. The choice between implementations strategies was based on the ease of integration with Infandango: although Infandango does already use Javascript like the Scriptaculous library[?], CSS is more heavily used. On the lab exercise page Infandango displays a single score for an exercise with a background colour determined by the score. Since this is already implemented using CSS, a similar approach was used for this implementation.

5.3 Prediction Model

The majority of the implementation for the model has already been performed when choosing the appropriate model: loading data, parsing data, training model and performing a prediction. The final remaining step is how to allow Infandango access to the model, a problem for which two solutions were created.

When Infandango first starts, load the data and train the model. This object would then passed around to the appropriate place, staying in memory and being used whenever it is needed. A problem with this model is changing the model during runtime is more or less impossible without restarting Infandango. Another problem is that the data used for training would also need to be stored somewhere, which could require a separate database running alongside the active database. Finally, this could potentially cause some coupling between the code used for training the model and the Infandango startup procedure.

The second option is training the model before hand and storing/loading this model to be used when necessary. A problem with this approach is 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 problem of the previous approach: the model can be changed easily, providing it uses the same loading interface; only the model needs to be stored, not all the data used for training; any model and training procedure can be used, as long as the same call can be made in order to make a prediction. The final implementation uses the training procedures discussed in chapter 4 to train the model and serialises it using the standard python module, pickle.

5.4 System Integration

Infandango uses Django to generate the web pages. Django allows for the insertion of variables into HTML, which creates dynamic content. Each page has a method which generates the variables which can be used for that page. 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 and would be very difficult to change. Instead, Django offers ”Context Processors” which is a way of hooking methods into the call to load any page, allowing these variables to be used by any page without telling every page explicitly. This also makes changing/removing the functionality much simpler.

5.5 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 how the control flow of this process.

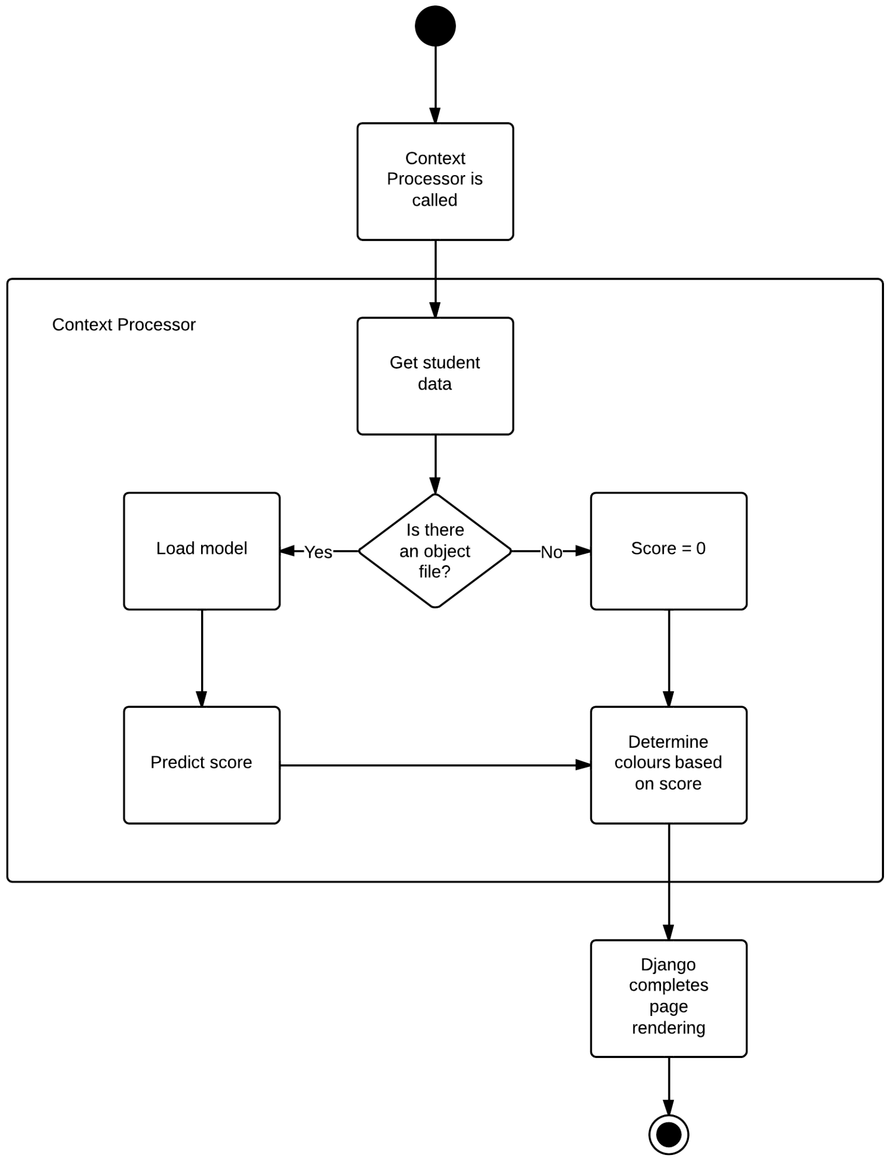


Figure 5.1: A flowchart of the context processor function and how it interacts with Django

Chapter 6

Evaluation

Chapter 7

Conclusion

Bibliography

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

[2] http://pybrain.org/.

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

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

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

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

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

[8] 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.

[9] 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.

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

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

[12] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2011.