Project ID for SYDE 522

Project Title:	Prediction of Academic Performance in Portuguese Secondary School Students
Project Member(s):	Chang Li, Maathusan Rajendram, Anastasia Santasheva, Evan Yeung
Summary of the Project:	This study aims to improve prediction accuracy of student performance. The models used are: SVM, Neural Network, Decision Tree, Naive Bayesian, and kNN. Dimensionality reduction, ensemble and voting were explored on each model. Genetic algorithms were investigated for Neural Networks.
Data Used:	649 rows for Portuguese course, 395 rows for Mathematics course 33 columns for both courses
Source of Data Used:	https://archive.ics.uci.edu/ml/datasets/Student+Performance
Results Achieved:	Results for three supervised approaches (binary and 5-level classification and regression) on the two datasets were recorded. For the Portuguese dataset the literature reports a validation accuracy of 93.0%, 76.1% and a RMSE of 1.32 respectively. We achieved a validation accuracy of 93.3%, 76.9% and a RMSE of 1.25. For the Mathematics dataset the literature reports a validation accuracy of 91.9%, 78.5% and a RMSE of 1.75 respectively. We achieved a validation accuracy of 92.4%, 79.7% and a RMSE of 1.69. Unseen test accuacuries for all cases have been included as well.

Prediction of Academic Performance in Portuguese Secondary School Students

Chang Li, Maathusan Rajendram, Anastasia Santasheva, Evan Yeung

Department of Systems Design Engineering

c299li@uwaterloo.ca, mrajendr@uwaterloo.ca, asantash@uwaterloo.ca, e6yeung@uwaterloo.ca

Abstract - *Education is a field that is constantly being revised. However, a significant amount of students are still failing courses, and even worse, dropping out of school. The aim of this study is to predict a student's final academic performance early on so educators can intervene and help prevent failure. Five supervised machine learning models were evaluated against three output variants (binary/five-level classification and regression) with two input setups. The impacts of dimensionality reduction, ensemble and voting were explored on each model. Genetic algorithms were investigated for Neural Networks optimization. Among these approaches the Decision Tree proved to have the highest prediction accuracy under almost all setups - achieving a 93.3% in its best case. The data used was collected from two public schools, and although it is limited to secondary education - in particular language and mathematics - these results can be extrapolated to other areas of the educational space.*

Keywords: Education, Performance, Decision Trees, Random Forest, SVM, Genetic Algorithm, Classification, Regression, Dimensionality Reduction, Ensemble, Voting.

1 Introduction

Education levels in Portugal have been lagging in comparison to the rest of Europe, quoting high student failure and high dropout rates as key issues [1]. Failures in the core classes of Portuguese and Mathematics are considered to be especially serious, as they form the pillars of understanding for subsequent courses. There are numerous factors that contribute to student's education and their ability to learn, including demographic and school performance over past years. This problem can in part be aided by a machine learning approach, as student performance can help education professionals introduce targeted corrective measures for students, such as remedial classes. In addition, the prediction of student achievement can help catch any potential downward cases early on and target support to bring them back on track [2]. Efforts to restore student retention rate have had some initial pay off. In 2013, Portugal reported that the portion of 15-year-olds with underachievement in Reading was 18.8%, while in 2016 it had improved to 17.2%, closer to the EU average. Similarly, 24.9% of 15 year olds were under-achieving in Mathematics in 2013, dropping to 23.8% in 2016, while the EU average was 22.1% [2]. Since interventions are primarily targeted towards at-risk individuals, improvements in accurately identifying these students can lead to further education improvements.

This paper analyzes a dataset [3] collected as questionnaires and grade reports from two Portuguese secondary schools between 2005-2006. Based on the review of related work it is subject to five main machine learning techniques to create a model which accurately predicts student performance. It explores and compares the effects of dimensionality reduction, ensemble, and voting techniques in developing a more suitable model. Overall results are recorded and main takeaways are discussed in the results section.

2 Background and Related Work

The basis of this report is on improving the findings obtained by Cortez and Silva (2008) [3] using the same dataset. Using Naive Bayes (NV), Support Vector Machine (SVM), Neural Networks (NN), Decision Trees (DT) and Random Forests (RF), the performance of students in both math and language courses were predicted at three different granularities (i.e. binary, five-level, and regression). It was found that different approaches performed better for different granularities and that high accuracy prediction is only possible knowing the student's previous academic performance. However, other relatively important features close to the root of the DT model include mother's job, absences, and frequency of going out. Table 1 shows the accuracy of their approaches using 10-fold validation. Based on their approaches, additional methods such as dimensionality reduction, bagging, boosting, and voting were applied to observe the impacts of these methods on the results. Other classification and regression models such as k Nearest Neighbour and genetic algorithms for optimization were considered.

Table 1: Best results of Cortez and Silva [3]

	Math		Language	
Granularity	**Best Classifier**	**Result**	**Best Classifier**	**Result**
Binary (PCC - %)	NV	91.9±0.0	DT	93.0±0.3
Five-level (PCC - %)	NV	78.5±0.0	DT	76.1±0.1
Regression (RMSE)	RF	1.75±0.01	NV & RF	1.32±0.00

In addition to Cortez and Silva's work, there are numerous studies on the topic of predicting student performance using machine learning methods. Several papers and their findings are highlighted below.

Kabakchieva (2013) [4] used two rule learners, a DT classifier, two Bayes classifier, and a Nearest Neighbour classifier to predict the final outcome of university students into five distinct categories given a student's previous academic performance (e.g. admittance exam score) and demographic data (e.g. gender). The DT classifier performed the best overall, achieving 65.84% accuracy for 10-fold cross-validation, and each classifier performed differently for the five categories. University admission score and number of failures in the first-year exams were the strongest predictors, consistent with Cortez and Silva's findings [3].

Yadav, Bharadwaj, and Pal (2012) [5] used ID3, C4.5, and CART DT algorithms to predict the end semester marks (first, second, third, or fail) of master students based on their lab work, attendance, assignment performance, seminar performance, and class test grade. Using 10-fold cross-validation, CART had the highest accuracy of 56.25%. Similarly, Huang and Fang (2013) [6] used multiple linear regression (MLR), MLP, radial basis function (RBF) network, and SVM to build regression models to predict course outcomes for dynamic physics based on cGPA, grades of math and physics courses, and course midterm grades. Results showed that SVM models produced the highest percentage of accurate predictions of 59.1% where a prediction is considered accurate if it has error $\pm 10\%$, and that the strongest predictor is the first midterm grade of the course.

One study that did not rely on past academic performance was Xing et al's work (2014) [7] where Genetic Programming Interpretable Classification Rule Mining (GP-ICRM) and activity theory were used to predict student outcome. Activity theory considers learnings as "the joint activity of a student, physical/symbolic tool(s), and another person(s) performing together as a working social system to achieve some outcome under constraints", and has the six elements: Tools, Object, Division of Labor, Community, Rules, and Subject. Student data from Virtual Math Team with Geogebra was mapped to elements of activity theory– for example, sum of all chat messages is a metric of Community– and GP-ICRM was used on the metrics to categorize students into five categories of performance. Compared to RF, Logistic Regression, MLP, and Naive Bayes, GP-ICRM had the highest overall prediction accuracy of 80.2%. Xing et al's work inspired the exploration of GA on NN for this study.

Lastly, a meta-analysis paper by Shahiri et al (2015) [8] conducted a systematic literature review on predicting student performance using data mining techniques to find the most important attributes in predicting student performance and the most effective prediction methods. NNs had the highest prediction accuracy, followed by DT, SVM, kNN, and NV. CGPA was the most important attribute in predicting student performance across all models. However, it is important to note that the meta-analysis does not state the validation method used in the studied literature and vast differences exist between the datasets used in each study.

3 Material and Methods

3.1 Dataset

Two datasets obtained from the University of Minho [3] were used for training and testing. The datasets provided student performances in a language (Portuguese) and mathematics course with 649 and 395 instances respectively, each containing 33 features such as test scores, habits, etc. A list of all the features can be found in Appendix B. The language dataset was initially used for training and validation. The best models discovered were then further validated on the mathematics dataset to determine if they generalized to obtaining similar results. Each dataset was split into a training set (80%) and a unseen testing set (20%) and randomized. The unseen set was tested on once using the best model determined through validation on the training set. These results are reported and compared with prior work in the results section.

3.2 Data Preprocessing

The original dataset contained nominal features (i.e. guardian, Mjob, Fjob, etc.) with more than two non-ordered discrete values, that needed to be converted to numerical values so they could be used as input for an machine learning model [9]. This was accomplished in two steps:

1. Integer Encoding
2. One-Hot Encoding

Integer encoding allowed the categorical data to be assigned to an integer value. Once the categorical data was converted and depending on whether the attribute was cardinal, one-hot encoding was then applied.

For the remaining ordinal and continuous data (i.e. G1, G2, taveltime, Dalc, etc.) values were normalized to a mean of zero and standard deviation of one. This normalization was done with Equation (1):

$$x' = \frac{x - \bar{x}}{\sigma} \tag{1}$$

Where x' is the normalized data, x is the original data, $\bar{x}$ is the mean and σ is the standard deviation.

There were 2 variations of input used for training the models:

1. *Input Setup A:* This includes all the features of the original dataset except G3 (the output).
2. *Input Setup B:* Same as A but adds the delta of G1 and G2 as a new feature to capture performance improvements or declines throughout the term.

Last, the output column (G3) was split into three supervised approaches:

1. *Binary classification:* G3 converted to binary values based on pass-fail criteria (pass if G3≥10, else fail).
2. *Five-level classification:* G3 converted based on Erasmus grade conversion system [10] (See Table 2 below).
3. *Regression:* original G3 value (grade is out of 20).

The sparseness of each dataset by the three approaches can be seen in Figure 1 and 2.

Table 2: The five-level classification system [10] (A-Excellent, B-Good, C-Satisfactory, D-Sufficient, F-Fail)

	ERASMUS Grading for France/Portugal				
	A	B	C	D	F
Grade (out of 20)	16-20	14-15	12-13	10-11	0-9

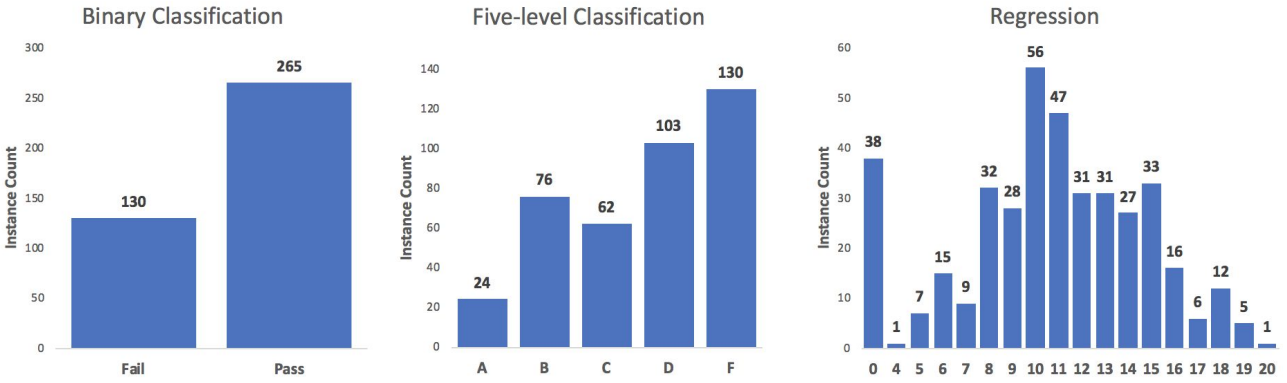

Figure 1: Sparseness of the language dataset.

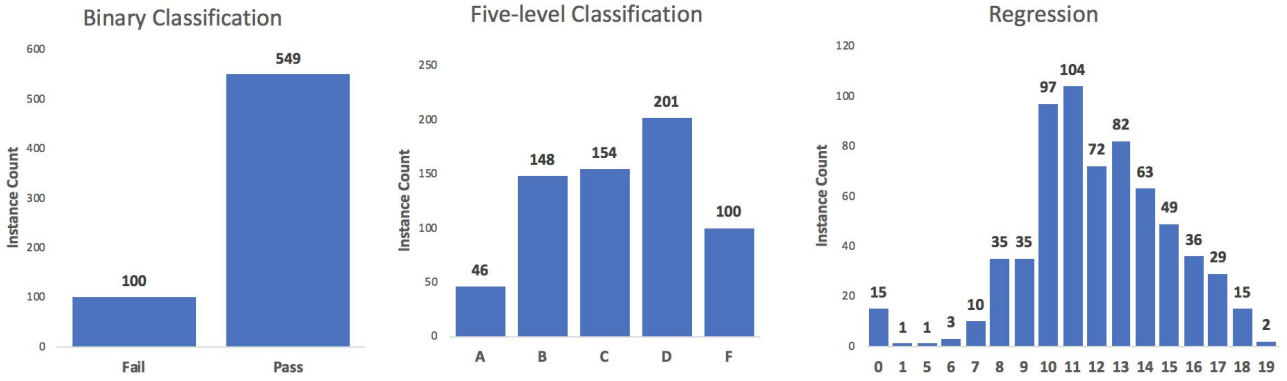

Figure 2: Sparseness of the mathematics dataset.

The combination of these input and output variants, in addition to the two original datasets (math and language), provided a total of 12 variant datasets to both train and test models against.

3.3 Dimensionality Reduction Methods

The effects of two dimensionality reduction methods were studied. They were Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). The impact of having no dimensionality reduction was also conducted and recorded to be used as a baseline for comparison with prior work.

PCA is a procedure to remove the redundancy present in a dataset by projecting it to a low dimensional vector space while achieving maximum variance. LDA projects a dataset onto a lower-dimensional space with maximum variance and good class-separability [11]. Both techniques are often used in dataset preprocessing to emphasize variation and bring out strong patterns [12].

3.4 Classification Methods

To keep consistency with the previous work, five main classification methods were validated against the three supervised approaches (binary and five-level classification and regression). They were:

- Support Vector Machine (SVM)
- Neural Network (NN)
- Decision Tree (DT) and Random Forest (RF)
- Naive Bayes (NV)
- k Nearest Neighbor (kNN)

Furthermore, ensemble and voting techniques were performed to investigate the improvements in accuracies, log loss and/or time. Ensemble techniques (e.g. bagging and boosting) were used to aggregate models of the same classifier while voting techniques (e.g. Voting Classifier - VC) were used to a aggregate models across classifiers.

3.5 Validation Methods

Two types of validation methods were used for each classifier – 10-fold cross validation and leave one out (LOO) cross validation. The average execution time for each iteration of the validation methods were also recorded to observe the impacts on the speed of the approach.

K-fold cross validation was used to keep consistent with prior work and helped compare any variations made to their approaches (e.g. effects of adding dimensionality reduction). LOO cross validation was considered since the dataset being used for training and validation was small (less than 1000 instances). However, due to the significant time it took to run LOO, it was usually left to the end once the tuning of hyperparameters was completed with k-fold.

3.6 Recording Results

Based on the configuration above, the results of each approach was evaluated using different techniques. For classification approaches Percentage of Correct Classification (PCC) was used, where a higher percentage was associated with better prediction results. For regression, Root Mean Square Error (RMSE), one of the most common metrics of evaluation, was used [13]. An RMSE value closer to zero indicated better fit of the model to the data. PCC was calculated using Equation (2) while RMSE used Equation (3):

$$PCC = [\frac{1}{N}\sum_{i=0}^{N-1} 1(y_i = \hat{y}_i)] \times 100\% \quad (2)$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2} \quad (3)$$

Where N is the number of samples, y_i is the true value at instance i and $\widehat{y_i}$ is its predicted value. In addition, error and log loss for different classifiers' hyperparameters were graphed. The trend in error helped indicate the change in prediction accuracy while log loss gave a sense of the confidence level of the model (where lower values were associated with a more confidently correct prediction). Finally, the average execution time for each iteration of k-fold was logged and recorded.

Overall, the best models were selected based on their prediction accuracy for 10-fold cross validation. If two models had very close prediction accuracies then the secondary factor - log loss - would be used to break the tie.

4 Results

4.1 Support Vector Machines

A Support Vector Machine (SVM) is a discriminative classifier that performs classification tasks to maximize the margin between different class labels. It is commonly used in educational performance predictions [14] and found to obtain high prediction accuracy [15]. The methodology outlined above was used to run experiments on SVM [16]. More specifically, Support Vector Classifier (SVC) was used for classification approaches and Support Vector Regression (SVR) for regression. With each approach, hyperparameters were tuned to obtain the highest accuracy while monitoring the log loss and execution time. Table 3 lists the parameters and their associated values used during

this process. Prior work [3] was limited to using a Gaussian kernel and solely tuning the hyperparameter Gamma.

Table 3: List of Parameters used for SVM tuning

Parameter	Value
C	{1, 10, 100, 1000, 10000}
Kernel	{'linear', 'poly', 'rbf', 'sigmoid'}
Gamma	$\{2^{-9}, 2^{-7}, 2^{-5}, 2^{-3}, 2^{-1}\}$

Through the validation process the linear kernel proved to consistently beat all other kernels for both classification and regression approaches. Furthermore, it was able to outperform the SVM prediction accuracies reported by Cortez and Silva [3] with regards to the binary classification and regression approaches. This was found to be consistent in both datasets. However, one pitfall with this variation was the noticeable increase in execution time. The linear kernel was found to take up to five times longer than the Gaussian kernel and increased at a similar rate as the C parameter increased.

Next, LDA was investigated with each approach and was found to have significant improvements in accuracy for 5 level classification. This in combination with a linear kernel for SVM provided a ~10% increase in prediction accuracies over prior work.

The impacts of PCA were found to be minimal on SVM. This is likely due to the already low dimensionality of the dataset. However, with PCA, the same level of accuracy could be obtained with a slightly lower dimension (n=45 from n=58) as shown in Figure 3 below. This resulted in slight improvements in time and prediction confidence (log loss) both of which were secondary factors when considering the best classifier configuration.

Also shown in Figure 3 was the impact of varying the input setup between A (all features) and B (all features including delta of G1 and G2). This adjustment reduced the delta between the train and validation results (bottom of Figure 3). This was consistent with classification approaches with the decrease of log loss value regardless of SVMs configuration. However, input setup B did not have any improvements in prediction accuracy and in some cases reduced it. One theory behind this could be because the added feature is simply the difference between two existing features thus, the model gains no new information to learn from with this input setup.

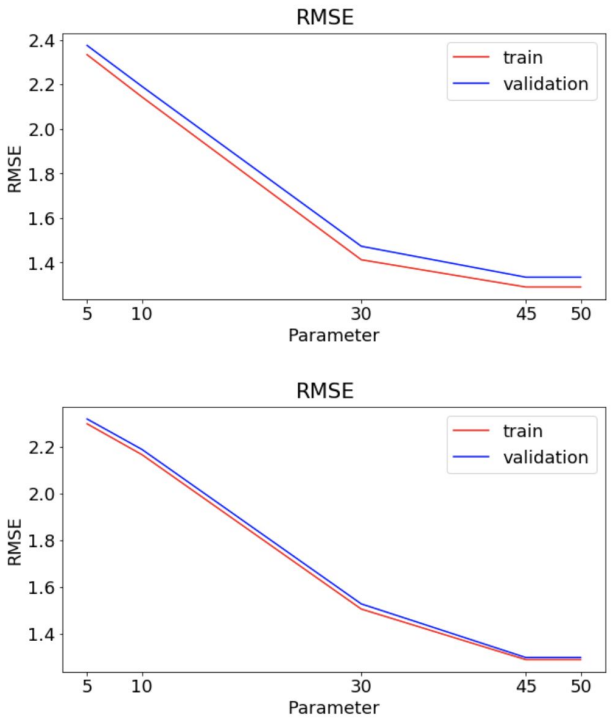

Figure 3: RMSE comparison of Input setup A (top) vs B (bottom) with PCA components varied on x-axis.

Finally, ensemble techniques were investigated with the best configuration of hyperparameters for SVM. Both bagging and Adaboost were used with their default configuration and found to have no improvement in prediction accuracies or log loss when compared to the best configuration of SVM.

The main takeaway from these findings was the impact of a linear kernel on the dataset. The prediction accuracies of all approaches in both datasets (math and language) were improved in varying levels with the adjustment to this parameter alone. Moreover, LDA helped improved the accuracies specifically for the five level classification approach.

4.2 Neural Network

Neural Networks were harnessed to identify whether a more accurate model could be established to predict students' grades. SciKit Learn's Multi-Layer Perceptron (MLP) classifier was used with the L-BFGS, a quasi-Newton solver for weight optimization. L-BFGS is a limited-memory version of BFGS which is well suited to problems with large numbers of variables. It finds the local minimum of an objective function with the help of the gradient of the objective function. As a quasi-Newton method, it approximates the Hessian matrix based on differences in the gradients instead of recomputing it [17]. The number of nodes in one hidden layer was varied between {0, 2, 4, 6, 8} in coordination with the Cortez &

Silva's paper [3]. The analysis was subject to 100 epochs of iteration. All other parameters of the MLP classifier were set to the default values provided by SciKit Learn. Notably, this included a ReLU activation function, the simplest non-linear activation function. An activation function was not specified by the Cortez and Silva paper, so ReLU was used for simplicity.

Over the short range of {0, 2, 4, 6, 8} nodes in a hidden layer, there was no observed impact on increasing the number of nodes. Overall, when 0 nodes in the hidden layer are used, the resulting analysis is similar to linear regression with the final value being passed through the activation function. In several cases, it was observed that this yielded results with the highest validation accuracy in comparisons to other numbers of nodes. In particular, the greatest validation accuracy was obtained for the binary classifier using PCA of size 45 with 0 nodes in the hidden layer for Portuguese (language studies). For five level classification, the highest validation accuracy was observed when using LDA with size five, as well as four nodes in the hidden layer. For regression, best results were observed with PCA of size 45, with four nodes in the hidden layer.

Overall, it can be consistently observed that the error increases when there are two nodes in the hidden layer. This is likely because 2 nodes does not adequately capture the data complexity. Instead, 4 nodes in the hidden layer appears to be more promising. Otherwise, the modified linear regression is favourable for binary classification.

To explore the data further, the number of nodes in the hidden layer was varied between (0, 40), which yielded more interesting results. For five-level classification, a notable improvement in the error and log loss was observed when a LDA with size 45 was used. At 20 nodes in the hidden layer, a validation accuracy of 66.7% was obtained (compared to 58.4% without LDA) and a decreasing log-loss value of 4.03 (vs. 5.64 without LDA). However, for the purposes of consistency with the Silva and Cortez, only 0-8 hidden nodes were considered for evaluating performance. This allows their findings to be used as a benchmark.

Varying the number of components used in PCA did not appear to have any influence on the accuracy of the final model obtained. However, in general, greater LDA values were found to yield higher validation accuracies in regression and five-level classifications. Overall, the neural network approach yielded the best results for the binary classification with a 93.07% validation accuracy. Without LDA, the highest attained accuracy is 67.73%, only a slight improvement on the results attained by Cortez and Silva. Accuracy improvements were also observed for the five-level classification (74.62%) and regression analysis (1.49) for the languages dataset. For the mathematics dataset, slight improvements on the validation accuracy are observed for the five-level classification (64.55%) and the regression analysis (2.03), while the binary classification performed slightly worse. Overall, results of both datasets benefited from LDA.

The MLPClassifier could not be run with Adaboost in the Scikit Learn framework, so this approach remained unexplored. When run with bagging, this consistently resulted in a slightly higher log loss, especially as the number of nodes in the hidden layer increased. This meant that as the number of nodes increased, the model more confidently made incorrect predictions, which is undesirable. The model was also tested with input setup B data, which yielded similar result trends, but overall had 2-3% lower test accuracies on unseen data. Input setup A was preferred.

4.3 Genetic Algorithm and Neural Network

In an attempt to improve the prediction accuracy with the MLP, a genetic algorithm (GA) was designed to optimize the the MLP topology. GAs are a population based optimization method that uses recombination and mutation to edit encodings of design solutions [18]. By running many generations of the population, and applying recombination and mutation to the best designs of each generation, it is hoped that the best design will be found faster than other approaches, such as enumeration of all possible designs [19].

To encode the MLP, a chromosome was built up of 10 integers. The first integer is a number from zero to nine representing the number of hidden layers in the MLP. The remaining are the number of nodes in that layer and range from 1 to ten nodes. This is a "weak" encoding as it focuses on encoding high level blocks of the network rather than specifying the details of each perceptron [19].

The fitness, or performance, of each network was evaluated via a 80/20 test/train split. K-fold was not used due to its high computational requirements. Selecting the best individuals of a population of size N was made by sampling the population N times taking three individuals each sample. The individual with the best fitness (accuracy) was moved into the next generation. Once selected, a 60% chance of recombination was applied to each pair of individuals. Recombination was done via two point crossover. This means two indices were selected on the first chromosome and everything between these points was swapped with the second chromosome. This is effectively swapping in and out different blocks of the network. The idea is to allow for the best parts of each network to be combined into a better final MLP. Finally, there was a 40% chance of each individual being eligible for mutation with a 5% chance of each integer in the chromosome of such an

individual to be mutated. The mutation was applied by uniformly sampling an integer from one to ten and replacing the chosen integer in the chromosome with it. The first integer, representing the number of layers, had a range of zero through nine instead. The mutation step helps to avoid the population getting stuck in local minimums.

The algorithm was run with a population size of 40 individuals for 70 generations (Figure 4). In all cases, the results from the MLP generated by the genetic algorithm were worse than that of a network chosen by hand. Results were close, usually within a few percent, but were consistently worse. The networks generated were also usually much larger, with seven, eight, or nine layers, than those chosen by hand.

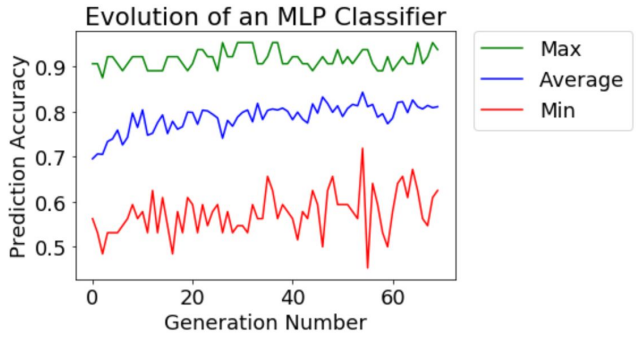

Figure 4: Average prediction accuracy of an MLP over 70 generations increased by over 10%. The most improvement is seen in early generations with later generations evolving slower and improving less.

Over the course of the algorithm running, the average accuracy of the population improved up to ten percent, but a wide range between the best and worst individual still existed. This is thought to be due to the way that the sample was being evaluated. To avoid overfitting the validation set, a random 20% of the training set was withheld as the validation set and the individual network performance was evaluated on it. Different individuals getting easier or harder validation sets could have led to "worse" individuals being selected over "better" ones. In the future, k-fold validation should be attempted instead. In addition, the lack of large improvements over time and relatively high accuracy of the first generation of individuals shows that the architecture of the MLP may not matter too much. Many different architectures produced similar results, as seen by the close comparison of the results of this algorithm and the hand tuned MLPs.

The population at the end of 70 generations also looked fairly homogenous, with only a few different individuals. A better solution may have been found if more variation was injected into the population. This could be achieved by a larger population or a higher chance of mutations. Different methods of recombination may also be explored to allow for generating more diverse MLPs in the next generation.

4.4 Decision Trees and Random Forest

Decision trees (DT) computed based on an optimized version of the CART algorithm [20] were explored. When building a decision tree, training data is split at every node based on some criterion (features in the dataset) to maximize the decrease in impurity, measured by the gini index where zero means perfect classification, or sum of squared errors (SSE) in case of regression. The splitting stops when a stopping criteria has been reached, such as a node becoming pure, the tree reaching the max-depth specified by the user, etc. [21] DTs have been used extensively in this area of research. Cortez and Silva [3] have shown that DTs are the best classifiers for binary and five-level classification of Portuguese performance on this dataset, and Kabakchieva [4] showed that DT outperforms NV and kNN for predicting university student outcome given past academic performance at five-level classification.

DTs can create large trees that capture small variations in the training data that generalize poorly. To avoid overfitting, the validation set error is observed at different tree depths, and it can be seen that as the tree grows larger, the validation error decreases initially and then increases while the training error continues to decrease. For example, as shown in Figure 5, the validation error increases as the tree's max depth increases beyond three while the training error continues to decrease. This diverging behaviour of training the validation errors shows that beyond a depth of three, the DT model begins to overfit the data, thus the DT with a depth of three is selected to be the best model. The process is then repeated with PCA and LDA with varying number of components, and both setup A and B to increase the model's accuracy.

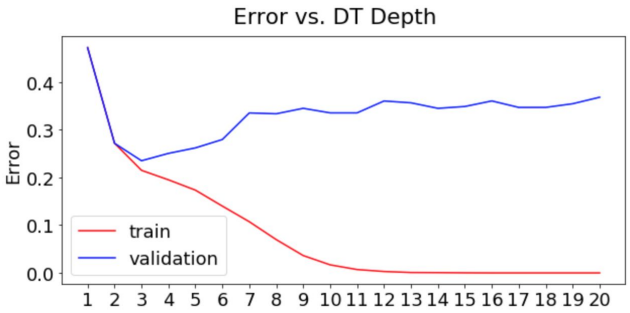

Figure 5: Training and validation accuracy for different tree depth.

To improve the prediction of DTs, Random Forests (RF) models were used. RF is essentially ensembling applied to DTs. In RF, multiple DTs are trained on subsamples of the original data generated by bootstrap aggregation where samples are drawn with replacement to create subsamples.

The RF's output is the average of the individual DT outputs. In addition to the max depth of each tree, RF also has the hyperparameter n_estimators which is the number of DTs in the RF. To find the best hyperparameter, a grid search cross validation algorithm was applied and the best model was selected with 3-fold cross validation on training data. In the first iteration of grid search, the grid search parameters considered were 1 to 40 with incremented by 2 for max depth, and 1 to 300 incremented by 10 for n_estimators. In the second iteration, values around the optimal parameters were searched with increments of 1 for both parameters. The final RF with the best hyper parameters is then validated with 10-fold cross validation on the training set.

Using the method above, attempts were made to first replicate the results from Cortez and Silva [3]. For the DT classifier, the results for binary and five-level classification of Portuguese courses were successfully replicated, as shown in Table 4 and 5, and better results were obtained for all others – 2-3% increases in PCC for classification predictions and reductions of 0.23-0.14 in RMSE for regression predictions. For RF, better results were obtained for regression predictions – a reduction in RMSE by 0.04 and 0.07 for Portuguese and math respectively – and accuracies of other prediction tasks were replicated exactly. Results from DT and RF classifiers show that DT is better at binary and five-level classification, whereas the RF classifier had lower RMSE for regression. By using setup B where the difference between G1 and G2 is added as a predictor, the RF regressor's RMSE decreased by 0.0551 for outcome in math, illustrating that improvements or declines in performance throughout the term is a useful predictor in final outcome. Applying both LDA and PCA resulted in worse accuracies for all predictions for both DT and RF.

By graphing the DT models, one can visually see the important attributes that determines a student's final outcome. For example, Figure 6 shows the DT model for binary classification of Portuguese class performance, including the gini impurity index at each node. It can be seen that the most important predictor is G2, followed by G1 and how often the student goes out. This result supports Cortez and Silva's [3] finding where frequency of going out is a strong non academic predictor of performance. Simple rules can be inferred from the DT classifiers. For example, G2 < 8.5 AND goes out > 1.5 (i.e. very low) leads to high chances of failure. Using these rules, educators can identify students who are at risk of failure and offer timely support.

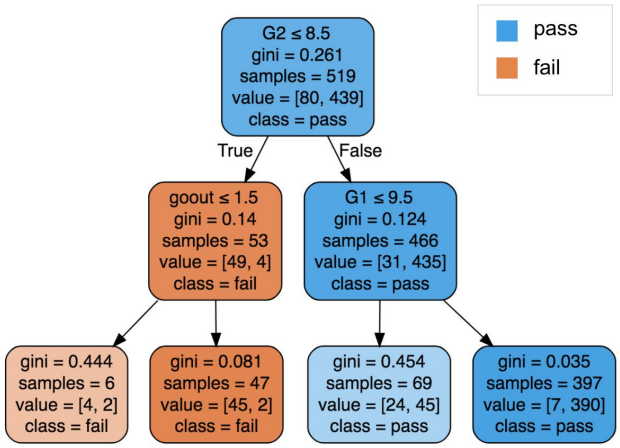

Figure 6: Visualized DT for binary classification of Portuguese class performance.

4.5 Naive Bayesian

Next, a Naive Bayes classifier was used to predict the students' grades. This classifier is based off of Bayes rule in which the probability of each class being the correct outcome is calculated based on the prior probability of that class and the likelihood of the observed predictor with that class. It is naive as it assumes independent probability between each of the predictors [22]. Because a NV approach cannot do regression, the regression was approximated by attempting to predict the grade as one of 20 classes.

The results of Cortez and Silva [3] could not be replicated with a basic NV classifier. It was found that prediction accuracies were significantly worse. This may be due to the reduction of dataset size, as the work done in this paper uses only 80% of the full dataset for validation. This could also be because of unknown differences in methodology. Cortez and Silva did not focus on the NV classifier and used it as a baseline [3]. Without knowing the specifics of their model, it is possible that a different methodology was applied that led to their increased accuracy.

Applying LDA was found to give the best results with the NV classifier. In all cases, LDA was able to significantly improve the accuracy of the prediction. For the binary classification, LDA with one component was found to be best while in the five-level data, one or two components were used for the math and Portuguese datasets, respectively. In one case, accuracy improved by over three times compared to a standard NV classifier. PCA was also found to significantly improve the predictions, although by not quite as much as LDA. It is concluded that dimensionality reduction is clearly beneficial for prediction with an NV classifier.

When input setup B was used, very slight improvements were seen in the language and math regression predictions and the math binary prediction. However, improvements were small (less than two percent increase in accuracy for the math binary) leading to an inconclusive result on whether input setup B would be best to use overall. In an attempt to further improve results, bagging and boosting were applied to the best NV classifier but were found to not improve the prediction accuracy.

4.6 K-Nearest Neighbours

An additional algorithm, k Nearest Neighbours was applied to the dataset. This classification algorithm works by using each of the features as one dimension of an n-dimensional vector. Each example in the dataset is embedded into this n-dimensional space and stored in the model. To classify a new example, the embedding for the example is calculated and the K nearest data points, using the L2 norm as a distance function, from the labelled dataset are found [23]. The most common label for these data points is the predicted class [24]. This algorithm was not attempted by Cortez and Silva [3] but was used to predict grades by Jie Xu et al. They found it to be worse performing than linear regression, logistic regression, or random forest techniques [25] but an attempt is still made here as it is a common classification technique [26].

To find the best model, the number of neighbours, K, was varied from 1 to 225 at increasing intervals (Figure 7). In general, it was found that around 30 neighbours was usually optimal. This may be due to outliers in the dataset that caused smaller sets of neighbours to predict the wrong class and a large number of neighbours pulling in unrelated examples. The kNN classifier was unable to outperform other classifiers in most cases. Applying dimensionality reduction with LDA helped to improve prediction accuracies by a moderate amount. PCA improved accuracies a small amount. The best results were found with a data projected onto a single LDA component.

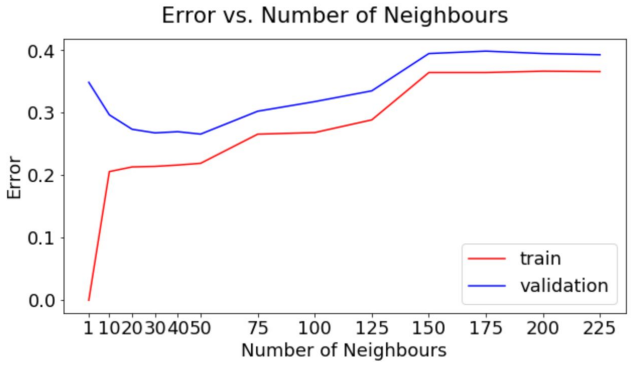

Figure 7: Training and validation accuracy as the number of neighbours increases for five level language classification.

Using input setup B had little effect on the kNN prediction accuracy. It resulted in a less than one percent increase in prediction accuracy on the language binary dataset and a 2.5 percent increase in accuracy when predicting on the math five level dataset. Since it is only effective in a couple of cases, that seem unrelated, it is not known whether using the extra predictor would be useful in the general case.

Finally, ensemble methods were tried. AdaBoost with kNN is not supported in SciKit Learn, the library used to build this model. Therefore, only bagging was tested. It was found that bagging had no effect or worsened prediction accuracy. This may be due to the smaller size of the dataset. Bagging only leads to removing examples that would have been useful to a classifier and, as such, one good classifier was able to perform better than ten worse ones.

4.7 Voting Classifier

With the best configuration determined for each of the aforementioned classifiers, (Table 4, 5, and 6) the top two to three were selected for each granularity and dataset and used as input with a VC. The use of VC was inspired by the improved results obtained when combining top predictors in recent works [27] as well as, the use of it in similar applications [28].

In most cases DT and RF were the top two methods but occasionally, SVM and NN would be used. However, the configureation requirements for nonlinear function methods (NN and SVM) conflicted with tree based ones (DT and RF). For example, both SVM and NN had noticeable improvements in accuracy due to the impact of LDA, but LDA caused disimprovement with any tree-based method. Since at least one tree-based method (often both) was in the VC, its overall accuracy got negatively impacted. Therefore, VC's were setup using either only nonlinear function methods or tree based ones. Tree based VC's were found to always outperform any combination of the top nonlinear function based VC's.

A voting classifier has two voting parameters - hard and soft. Hard uses predicted class labels for majority rule voting whereas soft predicts the class label based on the argmax of the sums of the predicted probabilities [29]. Both parameters were validated to determine their impacts on the prediction accuracy. Weights were also applied with each configuration where the highest values were applied to the best predicting classifier, and lowest to the worst among the set. Last, the impact of no weights was investigated, in which case uniform weights were applied. Soft voting and no weights was the best VC combination in all approaches (binary, five level and regression) and on both datasets.

As alluded to above, any dimensionality reduction techniques were found to consistently have negative prediction impacts on tree based methods which extended to any VC that included them. This was consistent regardless of the approach, dataset or other classifiers included.

One noticeable variant that caused prediction improvement was using the delta of G1 and G2 as a new feature (Input setup B). This aligned with the results found for tree based methods, since they benefited from this setup as well. Therefore, it makes sense a VC comprised of them would improve.

Finally, the ensemble techniques - bagging and boosting - were applied and found to have no improvement in prediction accuracies and log loss.

Table 4: Binary classification validation results (bold was best metric score; highlighted was best model overall)

	Mathematics			**Language**		
Classifier	**PCC (%)**	**Log Loss**	**Time (s)**	**PCC (%)**	**Log Loss**	**Time (s)**
SVM	91.72	**0.19**	0.06	92.29	**0.23**	0.10
NN	87.34	0.37	0.01	93.07	0.24	0.01
DT	**92.43**	0.30	**0.002**	**93.27**	0.23	**0.003**
RF	92.10	0.49	0.02	92.49	0.20	0.05
NV	86.06	0.26	0.01	89.40	0.29	0.01
KNN	85.80	1.42	0.01	89.78	0.55	0.02
VC (DT | RF)	92.10	0.22	0.43	93.26	0.19	0.05

Table 5: Five-level classification validation results (bold was best metric score; highlighted was best model overall)

	Mathematics			**Language**		
Classifier	**PCC (%)**	**Log Loss**	**Time (s)**	**PCC (%)**	**Log Loss**	**Time (s)**
SVM	69.58	0.72	0.01	73.03	0.68	0.02
NN	64.55	0.87	0.02	74.62	0.24	0.03
DT	**79.73**	1.08	**0.003**	**76.88**	0.82	**0.003**
RF	72.80	0.81	0.42	73.04	0.72	0.72
NV	71.80	**0.60**	0.01	71.30	0.74	0.01
KNN	72.50	0.82	0.01	73.40	1.13	0.01
VC (DT | RF)	79.14	0.63	0.47	76.86	0.67	0.64

Table 6: Regression validation results (bold was best metric score; highlighted was best model overall)

	Mathematics		**Language**	
Classifier	**RMSE**	**Time (s)**	**RMSE**	**Time (s)**
SVM	2.0224	0.1033	**1.2464**	0.1767
NN	2.0282	0.0580	1.4884	0.0652
DT	1.7058	**0.0023**	1.3157	**0.0029**
RF	**1.6866**	0.2761	1.2807	0.2750
NV	2.5622	0.0137	1.3992	0.0208
KNN	2.6127	0.0395	1.4111	0.0142
VC (DT | RF)	2.3498	0.2107	1.4802	0.1987

4.8 Comparison of Approaches

To recapitulate, five main machine learning models (SVM, NN, DT & RF, NB and kNN) were tested against three variants of the output G3 (binary/five-level classification and regression) with two input setups (setup A and B). The impacts of dimensionality reduction, ensemble and voting were explored on each model and their results have been highlighted. Finally, genetic algorithms were investigated; specifically on Neural Networks.

Dimensionality reduction techniques were found to have a positive impact on the predictions of nonlinear function methods whereas using input setup B improved tree-based methods. For majority of the approaches ensemble and voting techniques came close to top performing predictors but did not do better.

Overall, tree based methods still outperformed nonlinear function methods which resonates with the key findings from Cortez and Silva [3]. The Decision Tree proved to have the highest prediction accuracy under almost all setups. Furthermore, with the added feature of the delta between G1 and G2, tree-based methods were able to beat the best results from prior work. As a final test, Table 7 below, shows the top performing classifier for each approach, and its accuracy on the 20% unseen dataset.

Table 7: Best Results on Unseen Test Set

	Mathematics		**Language**	
Granularities	**Best Classifier**	**Result**	**Best Classifier**	**Result**
Binary (PCC - %)	DT	89.87	DT	95.38
Five-level (PCC - %)	DT	68.35	DT	76.50
Regression (RMSE)	RF	1.67	SVM	1.09

5 Summary and Conclusion

This paper attempted to replicate the results obtained by Silva and Cortez [3], and improve their accurracies by exploring an additional classifier kNN, and the effects of dimensionality reduction, ensemble, genetic algorithms, and voting on the classifiers and regressors they used. Comparing the 10-fold cross validation, the accuracy was improved for binary classification of math results, five-level classification of both math and language results, and regression of both math and language results. Table 8 shows a side by side comparison of these accuracies. For binary and five-level classification, DTs were the strongest predictors, whereas RF and SVM were the best regressors for math and Portuguese respectively.

There are limitations in comparing the validation results directly. First, in this paper, the data was split into 80/20 train/test datasets, where 10-fold cross validation was performed using the training set only. Silva and Cortez used 100% of the data in their 10-fold cross validation, thus the training sets used to build the classifiers and regressors during validation were larger. Second, this paper used Python and SciKit Learn's implementation of the classifiers and regressors, whereas Silva and Cortez used R [3]. Differences in implementation and default parameters could exist between the two languages.

Table 8: Comparison of 10-fold validation results (best accuracy is bolded)

	Mathematics		**Language**	
Granularities	**Current Results**	**Prior Results**	**Current Results**	**Prior Results**
Binary (PCC - %)	**92.43**	91.9±0.0	93.27	**93.0±0.3**
Five-level (PCC - %)	**79.73**	78.5±0.0	**76.88**	76.1±0.1
Regression (RMSE)	**1.69**	1.75±0.01	**1.25**	1.32±0.00

DT and RF classifiers were consistently strong performers. For simple classification tasks such as binary classification, DT performed very well because G2 and G1 were such strong predictors of a student's final outcome. The ability to visualize decisions trees allows one to extract simple IF-THEN rules that enable educators to identify students who are at risk and take timely action to prevent failure. In addition, it was found that the frequency of going out is a strong non academic related predictor of student failure. This information can be given to supporting staff and students who are looking to improve their grades with the aim of recommending preventative or remedial support.

References

[1] M. C. Pereira and H. Reis, "Grade retention during basic education in Portugal: determinants and impact on student achievement", *Bank of Portugal*, Economic Bulletin, June 2014.

[2] Directorate-General for Education and Culture. "Education and Training Monitor 2017, Portugal," *Education and Training*, European Union, Luxembourg: Publications Office of the European Union, 2017.

[3] P. Cortez and A. Silva. Using Data Mining to Predict Secondary School Student Performance. In A. Brito and J. Teixeira Eds., *Proceedings of 5th Future Business*

Technology Conference (FUBUTEC 2008) pp. 5-12, Porto, Portugal, April, 2008, EUROSIS, ISBN 978-9077381-39-7.
[4] D. Kabakchieva, "Predicting Student Performance by Using Data Mining Methods for Classification," *Cybernetics and Information Technologies*, vol. 13, no. 1, Jan. 2013.
[5] S. K. Yadav, B. Bharadwaj, and S. Pal, "Data Mining Applications: A comparative Study for Predicting Student's performance," *International Journal of Innovative Technology & Creative Engineering,* pp. 13–19, Feb. 2012.
[6] S. Huang and N. Fang, "Predicting student academic performance in an engineering dynamics course: A comparison of four types of predictive mathematical models," *Computers & Education*, vol. 61, pp. 133–145, 2013.
[7] W. Xing, R. Guo, E. Petakovic, and S. Goggins, "Participation-based student final performance prediction model through interpretable Genetic Programming: Integrating learning analytics, educational data mining and theory," *Computers in Human Behavior*, vol. 47, pp. 168–181, 2015.
[8] A. M. Shahiri, W. Husain, and N. A. Rashid, "A Review on Predicting Student's Performance Using Data Mining Techniques," *Procedia Computer Science*, 23-Dec-2015. Available: https://www.sciencedirect.com/science/article/pii/S1877050915036182. [Accessed: 18-Apr-2018].
[9] Chung-Chian Hsu, "Generalizing self-organizing map for categorical data," in *IEEE Transactions on Neural Networks*, vol. 17, no. 2, pp. 294-304, March 2006. doi: 10.1109/TNN.2005.863415
[10] Keele.ac.uk. (2018). *Notes on the grading system in use at Keele University as relevant for ERASMUS Students*. [online] Available at: https://www.keele.ac.uk/media/keeleuniversity/get/getpdfs/Grade-Eraspost-2004-1.pdf [Accessed 20 Apr. 2018].
[11] A. J. Izenman, "Linear Discriminant Analysis," *Springer Texts in Statistics Modern Multivariate Statistical Techniques*, pp. 237–280, 2013.
[12] P. Xanthopoulos, P. M. Pardalos, and T. B. Trafalis, "Linear Discriminant Analysis," SpringerBriefs in Optimization Robust Data Mining, pp. 27–33, Feb. 2012.
[13] K. Grace-Martin , "Assessing the Fit of the Model," *Applied Logistic Regression*, pp. 143–202, 2005.
[14] I. Lykourentzou, I. Giannoukos, and V. Nikolopoulos, "Dropout prediction in e-learning courses through the combination of machine learning techniques," *Computers & Education*, May 2009.
[15] A. Tekin, "Early Prediction of Students' Grade Point Averages at Graduation: A Data Mining Approach," *Eurasian Journal of Educational Research*, vol. 14, no. 54, pp. 207–226, 2014.
[16] "sklearn.svm.SVC¶," *sklearn.svm.SVC - scikit-learn 0.19.1 documentation*. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. [Accessed: 21-Apr-2018].
[17] A. S. Lewis and M. Overton. "Nonsmooth optimization via BFGS," 2009. Available: https://cs.nyu.edu/~overton/papers/pdffiles/bfgs_inexactLS.pdf [Accessed April 23, 2018]
[18] D. Whitley, "A genetic algorithm tutorial", *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.
[19] G. Miller, P. Todd and S. Hegde, "Designing Neural Networks Using Genetic Algorithms", in *Third International Conference on Genetic Algorithms*, George Mason University, 1989, pp. 379-384.
[20] "1.10. Decision Trees," *1.10. Decision Trees - scikit-learn 0.19.1 documentation.* [Online]. Available: http://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart. [Accessed: 18-Apr-2018].
[21] S. L. Crawford, "Extensions to the CART algorithm," *International Journal of Man-Machine Studies*, vol. 31, no. 2, pp. 197–217, 1989.
[22] H. Zhang, "Exploring Conditions for the Optimality of Naive Bayes", *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 19, no. 02, pp. 183-198, 2005.
[23] S. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques", in *Emerging Artificial Intelligence Applications in Computer Engineering*, I. Maglogiannis, K. Karpouzis, M. Wallace and J. Soldatos, Ed. Amsterdam: IOS Press, 2018, pp. 11-12.
[24] "1.6. Nearest Neighbors — scikit-learn 0.19.1 documentation", *Scikit-learn.org*, 2018. [Online]. Available: http://scikit-learn.org/stable/modules/neighbors.html. [Accessed: 19- Apr- 2018].
[25] J. Xu, K. Moon and M. van der Schaar, "A Machine Learning Approach for Tracking and Predicting Student Performance in Degree Programs", *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 5, pp. 742-753, 2017.
[26] K. Hechenbichler and K. Schliep, *Weighted k-Nearest-Neighbor Techniques and Ordinal Classification*. 2004.
[27] Y. Zhang, H. Zhang, J. Cai, and B. Yang, "A Weighted Voting Classifier Based on Differential Evolution," *Abstract and Applied Analysis*, vol. 2014, pp. 1–6, 2014.
[28] S. Kotsiantis, K. Patriarcheas, and M. Xenos, "A combinational incremental ensemble of classifiers as a technique for predicting students' performance in distance education," *Knowledge-Based Systems*, vol. 23, no. 6, pp. 529–535, 2010.
[29] "sklearn.ensemble.VotingClassifier¶," *sklearn.ensemble.VotingClassifier - scikit-learn 0.19.1 documentation*. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html. [Accessed: 21-Apr-2018].

Appendix A - Source Code

Import Packages

```
# standard useful packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# validation & normalization methods
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, KFold, LeaveOneOut

# accuracy, MSE, log loss & timer methods
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import log_loss
from time import time

# dim reduction & classification methods
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import VotingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.svm import SVR
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, VotingClassifier, BaggingClassifier

# make matplotlib to show plots inline
%matplotlib inline
```

Set Configuration

```
# 1. set dataset
ENABLE_POR_DATA = True     # set Portuguese course dataset
ENABLE_MAT_DATA = False    # set Math course dataset

# 2. set input setup
ENABLE_INPUT_SETUP_B = True   # adds the delta of G1 and G2 as a new column (GDelta)

# 3. set supervised approach for G3
ENABLE_BINARY_TARGET = False       # sets G3 to binary
ENABLE_5LEVEL_TARGET = False        # set G3 to five-level scale
ENABLE_REGRESSION_TARGET = True   # set G3 to current state for regression

# 4. set dimensionality reduction method - set both to false for none
ENABLE_PCN = True
ENABLE_LDA = False

# 5. set validation type
ENABLE_KFOLD = True
ENABLE_LOO = False

# 6. set final test
ENABLE_TEST = False
```

Load Dataset

```
# import data from csv
if (ENABLE_POR_DATA):
    dataframe = pd.read_csv('student-por-train.csv', usecols = range(0,33))
    dataframe_test = pd.read_csv('student-por-test.csv', usecols = range(0,33))
elif (ENABLE_MAT_DATA):
    dataframe = pd.read_csv('student-mat-train.csv', usecols = range(0,33))
    dataframe_test = pd.read_csv('student-mat-test.csv', usecols = range(0,33))

# shuffle dataset
dataframe = dataframe.sample(frac=1)

# find col length
num_cols = dataframe.shape[1]
```

Preprocessing

```
# helper functions for preprocessing
def convertToBinary(df, num_cols):
    df.loc[(df.G3 < 10), 'G3'] = 0
    df.loc[(df.G3 >= 10), 'G3'] = 1

    G3 = df['G3'].values
    return G3

def convertToFiveLevel(df, num_cols):
    df.loc[(df.G3 <= 9), 'G3'] = 0
    df.loc[(df.G3 > 9) & (df.G3 <= 11), 'G3'] = 1
    df.loc[(df.G3 > 11) & (df.G3 <= 13), 'G3'] = 2
    df.loc[(df.G3 > 13) & (df.G3 <= 16), 'G3'] = 3
    df.loc[(df.G3 > 16), 'G3'] = 4

    G3 = df['G3'].values
    return G3

def convertToInputSetupB(df, num_cols):
    df.insert(num_cols-1, 'GDelta', df.G2 - df.G1, allow_duplicates=True)

    return df

def oneHotEncode(df):
    df = df.drop(labels='G3', axis=1)
    cols_to_transform = [
                         'school',
                         'sex',
                         'address',
                         'famsize',
                         'Pstatus',
                         'Mjob',
                         'Fjob',
                         'reason',
                         'guardian',
                         'famsup',
                         'schoolsup',
                         'paid',
                         'activities',
                         'nursery',
```

```
                        'higher',
                        'internet',
                        'romantic',
                        ]
    hot_encoded_df = pd.get_dummies(df, columns = cols_to_transform)

    attributes = hot_encoded_df.values
    return attributes

def normalizeData(train_data, val_data):
    scaler = StandardScaler().fit(train_data)
    train_data = scaler.transform(train_data)
    val_data = scaler.transform(val_data)

    return train_data, val_data

# labels of Y
labels = []

# switch to input setup B
if (ENABLE_INPUT_SETUP_B):
    dataframe = convertToInputSetupB(dataframe, num_cols)
    dataframe_test = convertToInputSetupB(dataframe_test, num_cols)

# split one-hot encoded attributes (X) and G3 (Y)
X = oneHotEncode(dataframe)
X_tst = oneHotEncode(dataframe_test)

# selects supervised approach for G3
if (ENABLE_BINARY_TARGET):
    Y = convertToBinary(dataframe, num_cols).astype('int')
    Y_tst = convertToBinary(dataframe_test, num_cols).astype('int')
    labels = [0, 1]
elif (ENABLE_5LEVEL_TARGET):
    Y = convertToFiveLevel(dataframe, num_cols).astype('int')
    Y_tst = convertToFiveLevel(dataframe_test, num_cols).astype('int')
    labels = [0, 1, 2, 3, 4]
elif (ENABLE_REGRESSION_TARGET):
    Y = dataframe['G3'].values.astype('int')
    Y_tst = dataframe_test['G3'].values.astype('int')
```

Dimensionality Reduction

```
def pcaReduction(train_data, val_data, n_comp):
    pca = PCA(n_components=n_comp)
    train_data = pca.fit_transform(train_data)
    val_data = pca.transform(val_data)

    return train_data, val_data

def ldaReduction(train_data, train_target, val_data, n_comp):
    lda = LinearDiscriminantAnalysis(n_components=n_comp)
    train_data = lda.fit_transform(train_data, train_target)
    val_data = lda.transform(val_data)

    return train_data, val_data
```

Validation Methods

```
# method to calculate accuracy (PCC) & RMSE
def calcMetric(actual, predicted):
    if(ENABLE_REGRESSION_TARGET): return (mean_squared_error(actual, predicted))**(0.5) # calculates RMSE
    else: return accuracy_score(actual, predicted, normalize = True) # calculates PCC

def kFoldValidation(n_comp, penalty, n_splits=10):
    kFold = KFold(n_splits=n_splits)

    # run on validation data
    val_results = []
    train_results = []
    log_loss_results = []
    time_log = []

    for train_index, val_index in kFold.split(X):
        #start timer, return avg time below
        start = time()

        X_train, X_val = X[train_index], X[val_index]
        Y_train, Y_val = Y[train_index], Y[val_index]

        # normalize data
        X_train, X_val = normalizeData(X_train, X_val)

        # reduce dimensionality
        if (ENABLE_PCN): X_train, X_val = pcaReduction(X_train, X_val, n_comp)
        elif (ENABLE_LDA): X_train, X_val = ldaReduction(X_train, Y_train, X_val, n_comp)

        # build classifier for each set
        clf = buildClf(X_train, Y_train, penalty)

        predicted = clf.predict(X_val)
        val_accuracy = calcMetric(Y_val, predicted)
        predicted = clf.predict(X_train)
        train_accuracy = calcMetric(Y_train, predicted)

        # log loss calculation - for classification only
        if(not ENABLE_REGRESSION_TARGET): log_loss_results.append(log_loss(Y_val, clf.predict_proba(X_val),
labels=labels))

        val_results.append(val_accuracy)
        train_results.append(train_accuracy)
        time_log.append(time()-start)

    return np.mean(train_results, axis = 0), np.mean(val_results, axis = 0), np.mean(log_loss_results, axis =
0), np.mean(time_log, axis = 0)

def looValidation(n_comp, penalty):
    loo = LeaveOneOut()

    # run on validation data
    val_results = []
    train_results = []
    Y_test_prob = []
    log_loss_value = 0
    time_log = []
```

```
    for train_index, val_index in loo.split(X):
        #start timer, return avg time below
        start = time()

        X_train, X_val = X[train_index], X[val_index]
        Y_train, Y_val = Y[train_index], Y[val_index]

        # normalize data
        X_train, X_val = normalizeData(X_train, X_val)

        # reduce dimensionality
        if (ENABLE_PCN): X_train, X_val = pcaReduction(X_train, X_val, n_comp)
        elif (ENABLE_LDA): X_train, X_val = ldaReduction(X_train, Y_train, X_val, n_comp)

        # build classifier for each set
        clf = buildClf(X_train, Y_train, penalty)

        predicted = clf.predict(X_val)
        val_accuracy = calcMetric(Y_val, predicted)
        predicted = clf.predict(X_train)
        train_accuracy = calcMetric(Y_train, predicted)

        # save probability for log loss calculation
        if(not ENABLE_REGRESSION_TARGET): Y_test_prob.append(clf.predict_proba(X_val)[0])

        test_results.append(test_accuracy)
        val_results.append(val_accuracy)
        train_results.append(train_accuracy)
        time_log.append(time()-start)

    # log loss calculation - for classification only
    if(not ENABLE_REGRESSION_TARGET): log_loss_value = log_loss(Y, Y_test_prob, labels=labels)

    return np.mean(train_results, axis = 0), np.mean(val_results, axis = 0), log_loss_value, np.mean(time_log,
axis = 0)
```

Build Genetic Algorithm

```
import random
from deap import base, creator, tools
# First parameter is the number of hidden layers
# Remaining parameters are number of nodes in that layer
def createNNIndividual(params):
    num_hidden_layers = params[0] - 1
    hidden_layers = tuple(params[1:num_hidden_layers + 1])
    model = MLPClassifier(solver='lbfgs', hidden_layer_sizes=tuple(params))
    return model

# because of the way this is encoded, it works better if layers and nodes in layer are the same
MAX_LAYERS = 10
MAX_NODES_PER_LAYER = MAX_LAYERS

# Add creators
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

# Build toolbox
toolbox = base.Toolbox()
```

```
toolbox.register("attr_int", random.randint, 1, MAX_NODES_PER_LAYER)
# add 2 as first int is the num_layers - 1
toolbox.register("individual", tools.initRepeat,
                 creator.Individual, toolbox.attr_int, MAX_LAYERS + 2)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

def evaluateIndividual(nn_params):
    n_comp = 40
    X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.2)

    # normalize data
    X_train, X_val = normalizeData(X_train, X_val)

    # reduce dimensionality
    if (ENABLE_PCN): X_train, X_val = pcaReduction(X_train, X_val, n_comp)
    elif (ENABLE_LDA): X_train, X_val = ldaReduction(X_train, Y_train, X_val, n_comp)

    # build classifier for each set
    clf = buildClf(X_train, Y_train, nn_params)

    predicted = clf.predict(X_val)
    val_accuracy = calcMetric(Y_val, predicted)

    return val_accuracy

toolbox.register("evaluate", evaluateIndividual)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutUniformInt, low=1, up=MAX_NODES_PER_LAYER, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

def mainGA():
    POP_SIZE=40
    CXPB = 0.6
    MUTPB = 0.4
    NGEN = 70

    cols = ['gen', 'min_val', 'max_val', 'avg', 'std']
    all_results = pd.DataFrame(columns=cols)
    all_populations = []

    # create population
    pop = toolbox.population(n=POP_SIZE)
    all_populations.append(list(map(toolbox.clone, pop)))

    # Evaluate the entire population
    fitnesses = list(map(toolbox.evaluate, pop))
    for ind, fit in zip(pop, fitnesses):
        ind.fitness.values = (fit,)

    # Extracting all the fitnesses of pop
    fits = [ind.fitness.values[0] for ind in pop]

    # Variable keeping track of the number of generations
    g = 0

    # Begin the evolution
    while max(fits) < 100 and g < NGEN:
        # A new generation
        g = g + 1
```

```
        # Select the next generation individuals
        offspring = toolbox.select(pop, len(pop))
        # Clone the selected individuals
        offspring = list(map(toolbox.clone, offspring))

        # Apply crossover and mutation on the offspring
        for child1, child2 in zip(offspring[::2], offspring[1::2]):
            if random.random() < CXPB:
                toolbox.mate(child1, child2)
                del child1.fitness.values
                del child2.fitness.values

        for mutant in offspring:
            if random.random() < MUTPB:
                toolbox.mutate(mutant)
                del mutant.fitness.values

        # Evaluate the individuals with an invalid fitness
        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = (fit,)

        # The population is entirely replaced by the offspring
        pop[:] = offspring
        fits = [ind.fitness.values[0] for ind in pop]

        all_populations.append(list(map(toolbox.clone, pop)))

        # Save metrics
        length = len(pop)
        mean = sum(fits) / length
        sum2 = sum(x*x for x in fits)
        std = abs(sum2 / length - mean**2)**0.5

        generation_results = pd.DataFrame([[g, min(fits), max(fits), mean, std]], columns=cols)
        all_results = all_results.append(generation_results)

    return all_populations, all_results

populations, results = mainGA()
```

Classifier Selection

```
# method to build classifier
def buildClf(train_data, train_target, penalty):
    if(ENABLE_REGRESSION_TARGET):
        model = SVR(C=penalty, kernel='linear')
#         model = tree.DecisionTreeRegressor(max_depth = penalty)
    else:
#         model = SVC(C=penalty, probability=True, kernel='linear')
#         model = AdaBoostClassifier(base_estimator=SVC(C=penalty, probability=True, kernel='linear'))
#         model = BaggingClassifier(base_estimator=SVC(C=penalty, probability=True, kernel='linear'))
        model = getVotingClassifier(penalty)
#         model = GaussianNB()
#         model = MLPClassifier(solver = 'lbfgs', hidden_layer_sizes=penalty)
#         model = DecisionTreeClassifier(max_depth = None, max_features = penalty, criterion = "entropy")
```

```
#         model = RandomForestClassifier(n_estimators = penalty, max_features=450, criterion = "entropy")
#         model = KNeighborsClassifier(n_neighbors=penalty, p=1)
#         model = createNNIndividual(params)

    model.fit(train_data, train_target)
    return model
```

Final Test Method

```
# get accuracy/RMSE, logloss and time
def runTestSet(X_train, Y_train, X_test, Y_test, n_comp, penalty):

    start = time()
    test_log_loss = 0

    # normalize data
    X_train, X_test = normalizeData(X_train, X_test)

    # reduce dimensionality
    if (ENABLE_PCN): X_train, X_test = pcaReduction(X_train, X_test, n_comp)
    elif (ENABLE_LDA): X_train, X_test = ldaReduction(X_train, Y_train, X_test, n_comp)

    # build classifier, predict and get accuracy
    clf = buildClf(X_train, Y_train, penalty)
    predicted = clf.predict(X_test)
    test_accuracy = calcMetric(Y_test, predicted)

    # log loss calculation - for classification only
    if(not ENABLE_REGRESSION_TARGET): test_log_loss = log_loss(Y_test, clf.predict_proba(X_test))

    # execution time calculation
    test_time = time()-start

    return test_accuracy, test_log_loss, test_time
```

Obtain Accuracy, Log Loss & Error

```
# input reduced dimension - this can be ignored if none selected
n_comp = 1

# input parameter iterations - can tune other params in classifer selection method above
if(not ENABLE_TEST):
    hyperparams = [5, 10, 30, 45, 50]
else: hyperparams = [10] # input best parameter for test

# results array format: [[train], [validation], [log loss]]
results = [[],[],[]]

# calculate train error, test error, log loss & time for specific param
for penalty in hyperparams:

    if (ENABLE_KFOLD): train_res, val_res, log_loss_val, time_val = kFoldValidation(penalty, n_comp, 10)
    elif (ENABLE_LOO): train_res, val_res, log_loss_val, time_val = looValidation(n_comp, penalty)

    # save error, RMSE, log loss for each penalty for graph
    if (not ENABLE_REGRESSION_TARGET):
        results[0].append(1-train_res)
```

```
        results[1].append(1-val_res)
        results[2].append(log_loss_val)
    elif (ENABLE_REGRESSION_TARGET):
        results[0].append(train_res)
        results[1].append(val_res)

    print ("-----C={}-----".format(penalty))
    print ("Time: {} seconds".format(time_val))
    print ("-----Train-----")
    print ("Accuracy/RMSE: {}".format(train_res))
    print ("-----Validation-----")
    print ("Accuracy/RMSE: {}".format(val_res))
    print ("Log Loss: {}\n".format(log_loss_val))

# run best model on unseen test set
if(ENABLE_TEST):
    test_accuracy, test_log_loss, test_time = runTestSet(X, Y, X_tst, Y_tst, n_comp, hyperparams[0])

    print ("-----Test-----")
    print ("Accuracy/RMSE: {}".format(test_accuracy))
    print ("Log Loss: {}".format(test_log_loss))
    print ("Time: {} seconds".format(test_time))

# create error and log loss graph for penalty iterations - classification only
if(not ENABLE_REGRESSION_TARGET and len(hyperparams) > 1):
    f, axarr = plt.subplots(2, sharex=False)
    f.suptitle('Error and Log Loss', y = 0.92)
    f.set_size_inches(10, 10)

    # subplot 1: error plot
    axarr[0].set_ylabel('Error')
    axarr[0].plot(hyperparams, results[0], color='r', label='train')
    axarr[0].plot(hyperparams, results[1], color='b', label='validation')
    axarr[0].set_xticks(hyperparams)
    axarr[0].legend()

    # subplot 2: log loss plot
    axarr[1].set_ylabel('Log Loss')
    axarr[1].plot(hyperparams, results[2], color='g', label='log loss')
    axarr[1].set_xticks(hyperparams)
    axarr[1].set_xlabel('Parameter')
    plt.show()

# create RMSE graph for penalty iterations - regression only
if(ENABLE_REGRESSION_TARGET and len(hyperparams) > 1):
    f, axarr = plt.subplots(sharex=False)
    f.suptitle('RMSE', y = 0.95)
    f.set_size_inches(10, 5)

    # subplot 1: RMSE plot
    axarr.set_ylabel('RMSE')
    axarr.plot(hyperparams, results[0], color='r', label='train')
    axarr.plot(hyperparams, results[1], color='b', label='validation')
    axarr.set_xticks(hyperparams)
    axarr.set_xlabel('Parameter')
    axarr.legend()
    font = {'weight' : 'normal', 'size'   : 18}
    plt.rc('font', **font)
    plt.show()
```

Appendix B - Feature Set

Attributes (bold shows categorization type)

1. school - student's school (**binary**: 'GP' - Gabriel Pereira or 'MS' - Mousinho da Silveira)
2. sex - student's sex (**binary**: 'F' - female or 'M' - male)
3. age - student's age (**numeric-cont**: from 15 to 22)
4. address - student's home address type (**binary**: 'U' - urban or 'R' - rural)
5. famsize - family size (**binary**: 'LE3' - less or equal to 3 or 'GT3' - greater than 3)
6. Pstatus - parent's cohabitation status (**binary**: 'T' - living together or 'A' - apart)
7. Medu - mother's education (**numeric-cont**: 0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education)
8. Fedu - father's education (**numeric-cont**: 0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education)
9. Mjob - mother's job (**nominal-disc**: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other')
10. Fjob - father's job (**nominal-disc**: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other')
11. reason - reason to choose this school (**nominal-disc**: close to 'home', school 'reputation', 'course' preference or 'other')
12. guardian - student's guardian (**nominal-disc**: 'mother', 'father' or 'other')
13. traveltime - home to school travel time (**numeric-cont**: 1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour)
14. studytime - weekly study time (**numeric-cont**: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours)
15. failures - number of past class failures (**numeric-cont**: n if 1<=n<3, else 4)
16. schoolsup - extra educational support (**binary**: yes or no)
17. famsup - family educational support (**binary**: yes or no)
18. paid - extra paid classes within the course subject (Math or Portuguese) (**binary**: yes or no)
19. activities - extra-curricular activities (**binary**: yes or no)
20. nursery - attended nursery school (**binary**: yes or no)
21. higher - wants to take higher education (**binary**: yes or no)
22. internet - Internet access at home (**binary**: yes or no)
23. romantic - with a romantic relationship (**binary**: yes or no)
24. famrel - quality of family relationships (**numeric-cont**: from 1 - very bad to 5 - excellent)
25. freetime - free time after school (**numeric-cont**: from 1 - very low to 5 - very high)
26. goout - going out with friends (**numeric-cont**: from 1 - very low to 5 - very high)
27. Dalc - workday alcohol consumption (**numeric-cont**: from 1 - very low to 5 - very high)
28. Walc - weekend alcohol consumption (**numeric-cont**: from 1 - very low to 5 - very high)
29. health - current health status (**numeric-cont**: from 1 - very bad to 5 - very good)
30. absences - number of school absences (**numeric-cont**: from 0 to 93)
31. G1 - first period grade (**numeric-cont**: from 0 to 20)
32. G2 - second period grade (**numeric-cont**: from 0 to 20)
33. G3 - final grade (**numeric-cont**: from 0 to 20, output target)