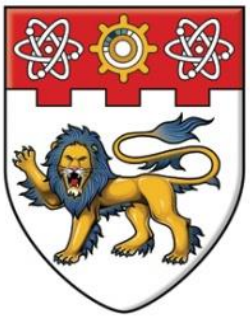

NANYANG TECHNOLOGICAL UNIVERSITY

CZ4032 Data Analytics and Mining

Project Report

Chin Jin Yao	U1321808B
Kaythi Myo Naing	U1322047F
Kerk Wei Yang	U1321927C
Leong Yong Fei	U1322484D
Lim Chin Meng	U1321429J

Date of Submission: 6th November 2015

SEMESTER 1 AY 2015/16

SCHOOL OF COMPUTER ENGINEERING

NANYANG TECHNOLOGICAL UNIVERSITY

Contents

Abstract

Education plays an important role in today's society, and much effort has been placed in improving the quality of education. With recent advances in Information Technology, there is an abundant amount of data available today. While some of these data holds valuable information that can be used to predict and improve students' academic performance, the raw data still needs to be analysed to extract interesting information such as trends and patterns.

In this project, six data mining algorithms are utilized to perform binary and grade classification tasks will be performed for the core subject Math in Portuguese secondary schools, which consists of 395 data samples containing student-related information. The algorithms include Naïve Bayes, Decision Tree, Random Forest, Neural Network, Support Vector Machine, and k-Nearest Neighbours.

The first experiment looked at using the wrapper approach to evaluate the relative importance of each input feature. Some similarities could be observed for the most significant and least significant attributes, and the impact of removing those attributes from the set of input features was examined as well. The second experiment focused on the subsets of attributes identified by the project team. The results show that a subset of attribute with more relevant attributes can indeed improve the predictive performance of the classifiers. The accuracies achieved for both the binary and grade classification tasks has surpassed that of a previous study as well.

The findings from this project has shown how various data mining algorithms can be applied to extract interesting patterns and trends from raw data, and at the same time, provide a deeper analysis of the available information. Furthermore, the information gained can provide ample opportunities for improving the quality of education and hopefully, the academic performance of students.

Problem Description

Motivation

In today's society, there is a great emphasis on education and many research has been carried out to identify methods and alternatives for improving the quality of education, and consequently the academic performance of students. With recent advances in Information Technology, there is an abundant amount of data available today. While some of these data holds valuable information that can be used to predict and improve students' academic performance, the raw data still needs to be analysed to extract interesting information such as trends and patterns. This can be done using automated data mining techniques, which can help to bring out important details that are often overlooked by human evaluators.

Related Work

Many studies have been done to leverage on data mining techniques in order to predict student performance. In particular, a previous study [1] aimed to utilize data mining techniques to predict secondary school student performance. The two main questions that the study tries to answer are "Is it possible to predict student performance", and "What are the factors that affect student performance". The study used student data from two different sources, namely student reports and questionnaires. As the student reports only contained information such as grades and number of absences, the questionnaire was used to complement it with several demographic, social, and school related information. The aim was to predict student performance and at the same time, identify the key factors that affect student performance. The study modelled two core subjects, Math and Portuguese, in Portuguese secondary schools under three data mining goals, which are binary classification, 5-level classification, and regression analysis. For each of these approaches, three input setups and four data mining algorithms, namely Decision Tree, Random Forest, Neural Network based on the Multilayer Perceptron, and Support Vector Machine, were used. In addition, a naïve classifier was used as the baseline. The study evaluated the predictive performance of the different data mining algorithms, and looked at the relative importance of the different factors as measured by the Decision Tree and Random Forest algorithms.

Problem Definition

The previous study was able to achieve good predictive performance by using different data mining algorithms and input setups, and at the same time, confirmed the authors' initial suspicion that student performance is highly affected by previous performances. However, there are various aspects that can be further improved. The previous study used about 30 attributes for its input setups, many of which proved to be irrelevant in predicting the student performance. Furthermore, the input setups were selected solely based on the authors' initial hypothesis that student performance is highly affected by previous performances. In fact, it is stated in the conclusion that automatic feature selection methods such as filtering or wrappers can be explored to select the more relevant factors, which will likely improve the performance of algorithms such as Neural Network and Support Vector Machine that are more sensitive to irrelevant features.

Approach

Methodology

In this project, the proposed approach is to use a smaller subset of attributes that are more likely to affect the student performance. Using the same data set [2] as the previous study, the project team hopes to achieve better predictive performance by focusing on the most relevant factors. The full set of attributes, together with their descriptions and data type is included in Appendix A. The project team has identified the various subsets of attributes that are more likely to affect the student performance, as shown in Table 1.

Table 1: Attribute Subsets

Subset	Attributes
Subset 1 (S1)	Pstatus (Parent's cohabitation status), Medu (Mother's education), Mjob (Mother's job), Fedu (Father's education), Fjob (Father's job), guardian (Student's guardian), famsize (Family size), famrel (Quality of family relationships), famsup (Family education support)
Subset 2 (S2)	sex (Student's gender), age (Student's age), traveltime (Travel time from home to school), studytime (Weekly study time), failures (Number of past class failures), activities (Extra-curricular activities), paidclass (Extra paid classes), internet (Internet access at home), higher (Intention to pursue higher education), romantic (In a romantic relationship), freetime (Free time after school), goout (Going out with friends), health (Current health status), absences (Number of school absences)
Subset 3 (S3)	Attributes in S1, G1 (First period grade), G2 (Second period grade)
Subset 4 (S4)	Attributes in S2, G1, G2
Subset 5 (S5)	G1 and G2 only

The subsets are determined by identifying the family-related factors and personal information related directly to the students. S1 contains all the attributes that are family-related, such as the education level and job of the parents. S2 contains all personal information of the student, such as the time spent studying, travel time from home to school, and the intention to pursue higher education. S3 contains all the family-related attributes in S1, together with the grades of the first period and second period. S4 contains all the student-related attributes in S2, together with the grades. S5 contains only the grades of the first period and second period. The assumption here is that student-related attributes are likely to be more relevant than family-related attributes, and that the previous performance such as first and second period grades will help to better determine the final grade. Based on the results from the previous study, it is highly likely that S5 will outperform the other subsets of attributes.

However, to justify the selection of the subsets, the project team will make use of the wrapper approach to identify the most relevant attributes and rank them based on how they affect the overall predictive performance. This will determine if the attributes chosen for the subsets are indeed the most relevant factors. In this project, only the binary classification and 5-level classification (denoted from here on as **grade classification**) tasks will be performed for the core subject Math, which consists of 395 data samples containing student-related information.

Algorithms

The previous study utilized four data mining algorithms, namely Decision Tree, Random Forest, Neural Network based on Multilayer Perceptron, and Support Vector Machine (denoted from here on as **SVM**) to perform both the binary classification and grade classification tasks. Rather than using the naïve classifier as the baseline, the project team will look at two other additional data mining algorithms to supplement the four existing algorithms.

The first data mining algorithm added is the Naïve Bayes algorithm, which is a simple probabilistic classifier based on applying Bayes' theorem with strong independence, or naïve, assumptions. The main assumption of the Naïve Bayes classifier is that the value of a particular feature is independent of the value of any other feature, given the class label. A Naïve Bayes classifier considers each of the given features to contribute independently to the final class label, regardless of any possible correlations between the different features. It can be trained very efficiently in a supervised learning setting, and works quite well for many situations. Furthermore, it only requires a small amount of training data to estimate the parameters necessary for the eventual classification.

The second data mining algorithm added is the k-Nearest Neighbour (denoted from here on as **k-NN**) algorithm. The main difference between the k-NN algorithm and the other data mining algorithms being used is that it is a lazy learner, which does not build models explicitly. It is extremely simple, as it only relies of the set of training records and a distance metric to identify the k nearest neighbours and use their class labels to determine the class label of the unknown record. One issue with the k-NN algorithm is that the k-value needs to be chosen adequately, and the prediction is made based on local information and can therefore be more susceptible to noise in the data.

The two data mining algorithms, namely the Naïve Bayes algorithm and k-NN algorithm, are added to provide a contrast to the existing algorithms used in the previous study. Furthermore, simple algorithms often work better in many real-world scenarios as compared to complicated algorithms such as Neural Network or Support Vector Machine. This would allow the project team to evaluate the advantages and disadvantages of the different data mining algorithms, and determine how a different subset of attributes affect the predictive performance of these algorithms.

Implementations

The project team developed a Java program that makes use of the Weka API to implement the data mining algorithms. Weka [3] is a collection of machine learning algorithms for data mining tasks, and it contains the various tools required for the experiments, such as data pre-processing, filtering data, and classification. Additionally, it comes with a set of well-documented Javadoc files for its Java API.

Firstly, to make use of the same data set as the previous study, some simple data pre-processing needs to be performed. As the final grade G3 is a numerical value, it needs to be converted to a nominal data type to suit the binary classification and grade classification tasks. As shown in Table 2, the approach used in the previous study was applied.

Table 2: Conversion of final grade G3 to its binary and grade equivalents

Final Grade (G3)	**0 -9**	**10 - 11**	**12 - 13**	**14 - 15**	**16 - 20**
Binary	Fail	Pass			
Grade	F	D	C	B	A

Before the relative importance of each attribute can be examined, the data mining algorithms need to be implemented and their predictive performance needs to be compared to those achieved in the previous study. Similar to the previous study, the 10-fold cross validation method will be used to assess how the predictive performance of each algorithm will generalize to an independent data set. For both the binary and grade classification tasks which produces discrete values as the final outputs, the set of algorithms utilized for the experiments will be evaluated using the Percentage of Correct Classification (denoted from here on as **PCC**). For such classifiers, a high PCC will imply that the performance of the classifier is good, especially since 10-fold cross validation is performed to ensure that the classifier does not favour the training data. The PCC can be calculated using the following equations:

$$\delta(i) = \begin{cases} 1, & y_i = \hat{y}_i \\ 0, & otherwise \end{cases} \quad (1)$$

$$PCC = \sum_{i=1}^{N} \delta(i) \,/\, \mathrm{N} * 100\% \quad (2)$$

Where y_i is the actual class label for the i-th data instance, $\hat{y}_i$ is the predicted class label, and N is the total number of data instances.

As mentioned earlier, the six data mining algorithms that will be used for the experiments are the four algorithms from the previous study, specifically Decision Tree, Random Forest, Neural Network based on Multilayer Perceptron, and SVM, as well as two additional algorithms which are Naïve Bayes and k-NN.

The Decision Tree algorithm is implemented using the J48 [4] class from the Weka library, which generates a pruned or unpruned C4.5 decision tree. C4.5 builds a decision tree from the given set of training data based on the concept of information entropy, where the attribute that provides the highest information gain is used to split the set of samples into smaller subsets.

The Random Forest algorithm is implemented using the RandomForest [5] class in the Weka library. The Random Forest is basically an ensemble of decision trees, where each decision tree makes an independent prediction and the most popular class label will be chosen as the final output of the Random Forest. As compared to the Decision Tree, the Random Forest will be less susceptible to noise and tends to perform better in most scenarios.

Similar to the previous study, the Neural Network used will be based on the popular Multilayer Perceptron. This is implemented using the MultilayerPerceptron [6] class available in the Weka library. The Neural Network is a feedforward network trained in a supervised manner with the error back-propagation algorithm based on the error correction learning rule. The weight matrix will be updated during the backpropagation process for each training pattern, and the training stops when convergence is achieved. For the implementation of the Multilayer Perceptron network in Weka, sigmoidal transfer functions are used by default as the class label is nominal.

As for the SVM, it is implemented using the SMO [7] class from the Weka library. The SMO class is basically an implementation of Sequential Minimal Optimization algorithm used for training a support vector classifier. The solution for the quadratic programming optimization problem required for training a SVM is obtained by breaking the problem into a series of smallest possible problems and solved analytically when using the Sequential Minimal Optimization algorithm.

The Naïve Bayes classifier is implemented using the NaïveBayes [8] class in the Weka library. It utilizes a simple probabilistic model, with the main assumption being that the value of a particular feature is independent of the value of any other feature, given the class label. It is relatively fast and efficient, and tends to work well for many situations.

Lastly, the k-NN classifier is implemented using the IBk [9] class from the Weka library. It is different from the first five classifiers as it is an instance-based classifier, and does not build an explicit model based on the training data. The full set of training data is used together with a distance metric to find the k nearest neighbours and the class labels of these neighbours are used to determine the class label of the unknown data instance. It can be more susceptible to noise as compared to classification algorithms which utilizes a global model that fits the entire input space. However, it is included for evaluative purposes against the other five classification algorithms.

The full set of 32 attributes are used as the input features to gauge the predictive performance of the above-mentioned classification algorithms using the default parameters for each of these algorithms, and compared against those achieved using a similar setup in the previous study. Table 3 shows the comparison of the PCC for both the binary and grade classification tasks.

Table 3: Comparison of PCC for both binary and grade classification tasks

Binary	**Classifiers**					
	Decision Tree	Random Forest	Neural Network	SVM	Naïve Bayes	k-NN
Previous Study	90.7 %	**91.2 %**	88.3 %	86.3 %	-	-
Current Project	**89.6 %**	88.6 %	88.1 %	87.9 %	87.9 %	66.1 %
Grade	**Classifiers**					
	Decision Tree	Random Forest	Neural Network	SVM	Naïve Bayes	k-NN
Previous Study	**76.7 %**	72.4 %	60.3 %	59.6 %	-	-
Current Project	**73.7 %**	66.6 %	53.9 %	51.1 %	67.3 %	26.3 %

The best algorithm for each classification task are shown in bold for both the previous study and the current project. Although using the classes from the Weka library together with the default parameters produce comparable results to those obtained in the previous study, it can still be improved by optimizing the parameters used for each of these algorithms. It should be noted that the k-NN algorithm performed less favourably for both the binary and grade classification tasks as compared to the other five algorithms.

First of all, the Decision Tree algorithm is optimized by using a confidence factor of 0.175 and having the value of the 'minNumObj' parameter set to 13. These optimal parameters are obtained by studying the implementation of the algorithm, and experimenting with the different values based on the findings of an analysis [10] done on decision trees in Weka. The confidence factor is used to regulate how the pruning of the decision tree will be performed, while the 'minNumObj' parameter determines the minimum number of instances for each leaf node. Figure 1 shows how different values used for the 'minNumObj' parameter affects the PCC for the Decision Tree algorithm while using the optimal value of 0.175 for the confidence factor.

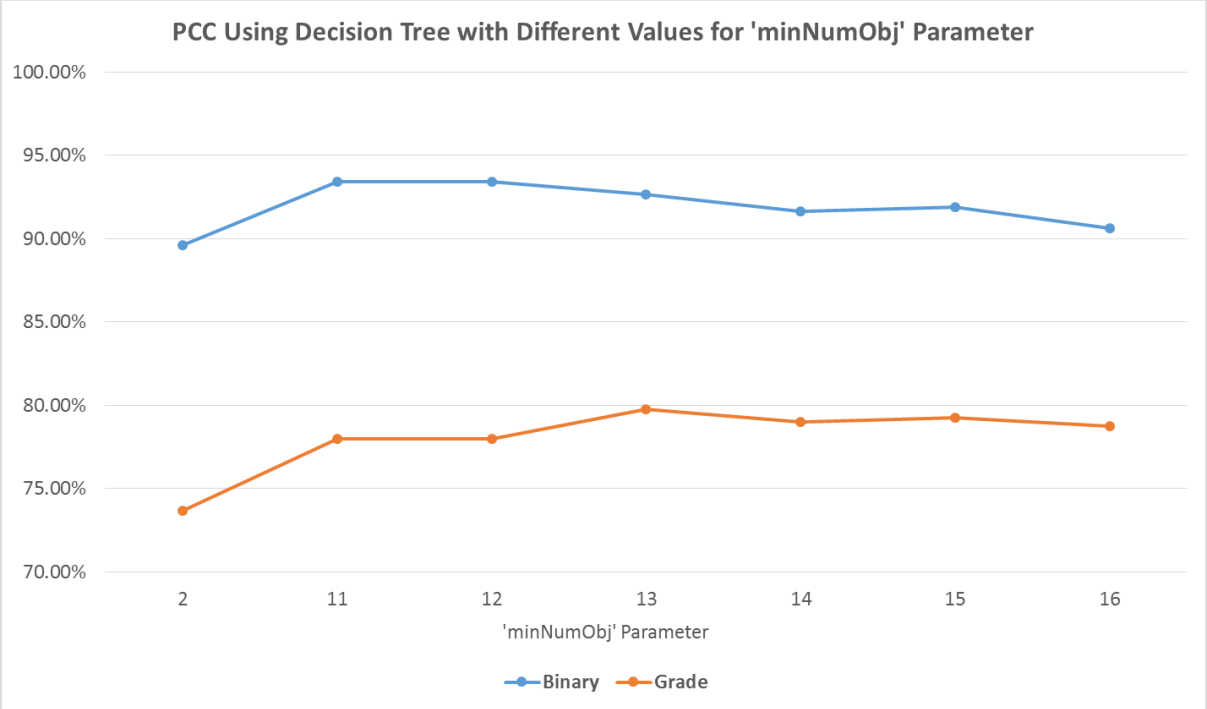

Figure 1: PCC for Decision Tree using different values for 'minNumObj' parameter

Next, the Random Forest algorithm is optimized by changing the number of features from 0 (unlimited) to 10, changing the maximum depth from 0 (unlimited) to 8, and increasing the number of trees from 100 to 200. The value 10 was chosen for the number of features to be used for the random subspace method after experimenting with different values based on the findings of a paper [11] on the influence of hyperparameters on the accuracy of random forests. As for the maximum depth, allowing the decision tree to grow to an unlimited depth could result in noise being captured and cause overfitting. In this case, the ideal value is 8, and that is roughly equal to $\log_2 N$, where N is the number of data samples which is 395. For the number of trees being used, using a larger number of trees in the random forest would mean that there are more unique combinations of data covered, thus reducing the error rate and biasness.

For both the Neural Network and SVM algorithms, it is hard to justify the choice of the optimal parameters as those are usually found through trial-and-error and depends on the dataset being used. When using the Multilayer Perceptron, the most commonly adjusted parameters are the number of hidden layers, number of hidden neurons in each hidden layer and the learning rate. Since the number of data instances is relatively small, there is no need for more than one hidden layer and an excessive number of hidden neurons as these will significantly increase the training time and tends to cause overfitting. Therefore, the Neural Network is optimized by using a single hidden layer with 5 hidden neurons. Other than that, the learning rate is changed to 0.09 to allow the weights to converge to their optimal values at a suitable rate. As for the SVM which is implemented based on the Sequential Minimal Optimization algorithm, the default implementation normalizes all attributes. After experimenting with the different values for the complexity constant and how the attributes are handled, the derived optimal setup involves using the default complexity constant of 1 and neither normalizing nor standardizing the attributes.

As for the Naïve Bayes classifier, the predictive performance is optimized by allowing the use of supervised discretization. For the dataset being used, after excluding the class label which is the final grade G3, there are still a total of 14 numerical attributes. Basically, discretization is the process of transforming a continuous-valued variable into a discrete one by creating a set of contiguous intervals, or equivalently a set of cut-off points, that spans the range of the variable's values. Supervised discretization methods will discretize a variable to a single interval if the variable has little or no correlation with the target variable, and this effectively removes the variable as an input to the classification algorithm. Therefore, discretization can help to significantly improve the predictive performance of algorithms that are sensitive to the dimensionality of the data, such as Naïve Bayes.

With regards to the k-NN algorithm, the best predictive performance can be achieved using a k value of 9 and at the same time, inversely weighting the neighbours. By inversely weighting the neighbours, it ensures that the closer neighbours will contribute more to the final output. However, as it is a lazy learner, the performance cannot be improved any further without removing the irrelevant features.

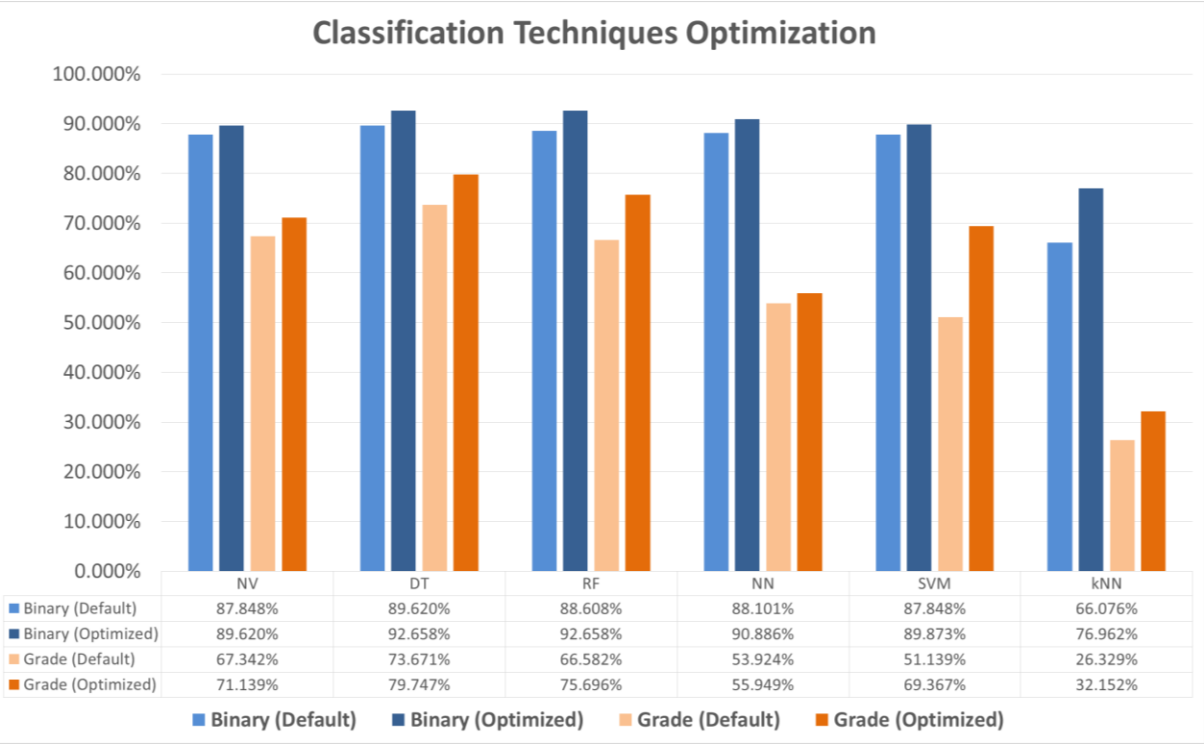

Figure 2: Optimization of classification techniques

Figure 2 shows the improvement in the predictive performance of the different classifiers for both the binary and grade classification tasks, after optimizing the parameters. The PCC obtained using the optimized classifiers will serve as the baseline for performing feature selection.

Experiments

Experiment 1: Feature Evaluation using the Wrapper Approach

The predictive performance of the classifiers will generally benefit from utilizing a smaller subset of attributes that are more relevant. In order to assess the relative importance of each attribute, the wrapper approach will be used. The PCC obtained from the optimized classifiers using the full set of 32 attributes will be used as the baseline. Each of the 32 attributes will be removed one at a time, and the overall predictive performance will be compared to the baseline performance.

```
public static int[] rankTechniques(RemovedAttribute baseline) {

    int numTechniques = Const.MODELS.length;
    int[] ranks = new int[numTechniques];

    for(int i = 0; i < numTechniques; i++) {

        int rank = 1;
        for(int j = 0; j < numTechniques; j++) {

            double accuracy1 = baseline.getAccuracy(Const.MODELS[i]);
            double accuracy2 = baseline.getAccuracy(Const.MODELS[j]);

            if(accuracy1 > accuracy2)
                rank++;
        }

        ranks[i] = (numTechniques - rank + 1);
    }

    return ranks;
}

public static double[] rankToWeight(int[] ranks) {

    int numTechniques = Const.MODELS.length;
    double[] weights = new double[ranks.length];

    for(int i = 0; i < weights.length; i++) {
        weights[i] = 1.0 + ((numTechniques - ranks[i]) * 0.1);
    }

    return weights;
}
```

Figure 3: Ranking the classification techniques and deriving the weight vector

As shown in Figure 3, in order to preserve the performance of the top classifiers, the classifiers are ranked according to their predictive performance for the baseline set of attributes and a weight vector is derived for calculating the overall accuracy gain/loss. This ensures that an improvement in the predictive performance of a good classifier will be favoured over that of a poorly performing classifier, when performing the feature subset selection. The ranking and the corresponding weight for each classifier is shown below in Table 4.

Table 4: Rank and corresponding weight for each classifier

Binary			Grade		
Rank	**Technique**	**Weight**	**Rank**	**Technique**	**Weight**
1	Decision Tree	1.4	1	Decision Tree	1.5
1	Random Forest	1.4	2	Random Forest	1.4
3	Neural Network	1.3	3	Naïve Bayes	1.3
4	SVM	1.2	4	SVM	1.2
5	Naïve Bayes	1.1	5	Neural Network	1.1
6	k-NN	1.0	6	k-NN	1.0

To assess the relative importance of each attribute, the overall accuracy gain/loss after removing each attribute can be calculated using the following equation:

$$\theta(attr) = \sum_{i=0}^{n} (acc(i) - baselineAcc(i)) * weight(i) \quad (3)$$

Where $acc(i)$ is the PCC of classifier i after removing attribute $attr$, $baselineAcc(i)$ is the PCC of classifier i using the baseline set of 32 attributes, and $weight(i)$ is the weight assigned to classifier i based on its rank.

The relative importance of each attribute can be determined by ranking the attributes based on their overall accuracy gain/loss. For an attribute that results in the most accuracy gain, or equivalently, the least accuracy loss, it is likely to be less important. This is due to the fact that when the predictive performance is improved by removing that particular attribute, it can be inferred that the attribute is causing confusion for the classifiers and it negatively affects the performance of the classifier. The complete set of ranking and overall accuracy gain/loss is included in Appendix B. The five most significant attributes and the five least significant attributes for both the binary and grade classification tasks are shown below in Table 5.

Table 5: Most significant and least significant attributes for both binary and grade classification tasks

Binary			
Most Significant		**Least Significant**	
Attribute	**Overall Accuracy Gain/Loss**	**Attribute**	**Overall Accuracy Gain/Loss**
G2	-39.64 %	school	-1.975 %
G1	-19.74 %	famsize	-2.759 %
goout	-9.620 %	studytime	-2.911 %
internet	-8.962 %	age	-3.823 %
romantic	-8.684 %	failures	-3.899 %
Grade			
Most Significant		**Least Significant**	
Attribute	**Overall Accuracy Gain/Loss**	**Attribute**	**Overall Accuracy Gain/Loss**
G2	-133.9 %	age	+3.089 %
G1	-20.91 %	traveltime	+2.785 %
nursery	-11.67 %	Dalc	+2.608 %
romantic	-9.873 %	famsize	+1.873 %
reason	-8.886 %	higher	+0.785 %

It can be observed from Table 5 that there are various similarities for the most significant attributes and least significant attributes for both the binary and grade classification tasks. In particular, the most significant attribute is always G2, while the second-most significant attribute is G1. Other than that, the 'romantic' attribute is also in the five most significant attributes for both cases. The results obtained as part of this experiment confirms the suspicion of the authors in the previous study that student performance is highly affected by previous performances. As for the least significant attributes, both 'age' and 'famsize' are included for both the binary and grade classification tasks. It can be seen that the overall accuracy actually increased by nearly 3% when the 'age' attribute was removed for the grade classification task. This implies that these two attributes are likely to be irrelevant in terms of predicting the final grade G3.

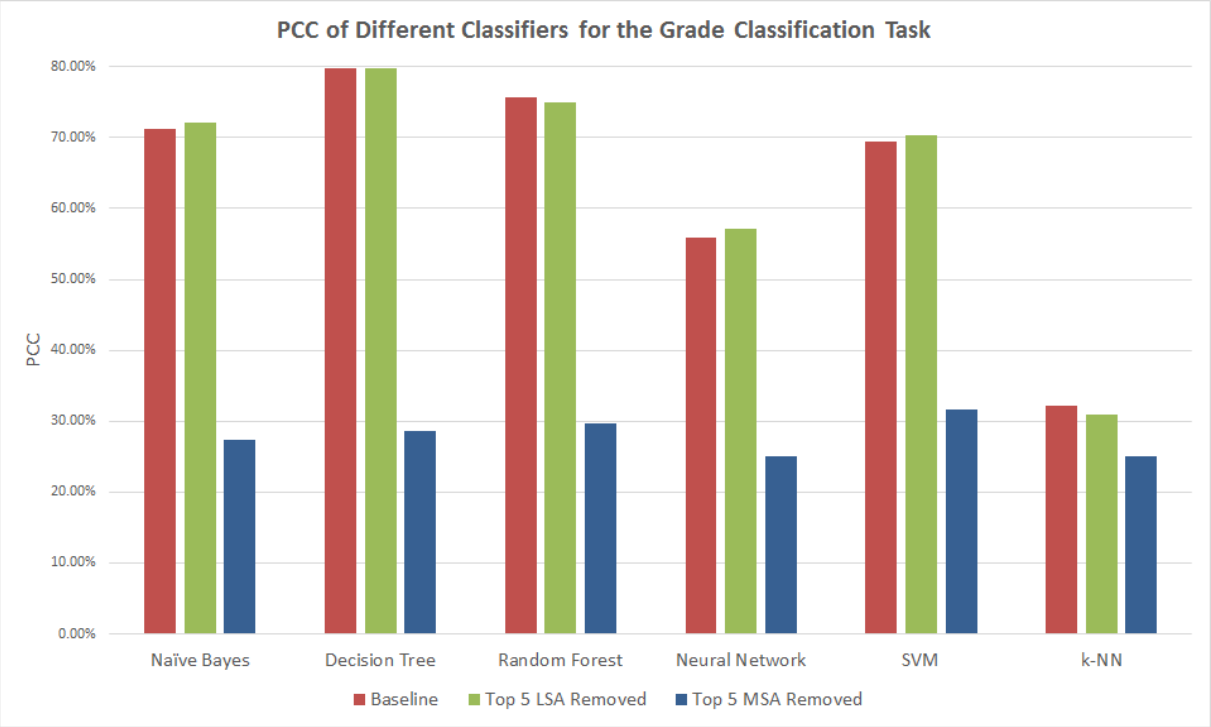

Figure 4: PCC of different classifiers for grade classification task with top five LSA and top five MSA removed

Additionally, the project team evaluated how the classifiers will perform when the five least significant attributes and the five most significant attributes are removed from the set of input features for the grade classification tasks. As shown from the results in Figure 4, it can be observed that removing the five least significant attributes improved the PCC for the Naïve Bayes classifier, Neural Network and SVM. However, the PCC remained the same or slightly decreased for the other classifiers. This is likely due to the fact that there still exists a large number of irrelevant features even after the five least significant attributes have been removed. On the other hand, removing the five most significant attributes caused the PCC for all six classifiers to decrease considerably. This highlights the importance of having the most relevant attributes in the set of input features for the classifiers to perform satisfactorily.

Experiment 2: Subset Evaluation

The main purpose of this experiment is to obtain a correlation between a properly chosen subset of attributes and the predictive performance of the various classifiers. The first step would be to investigate the relative importance of each attribute in the different subsets, based on the findings from the first experiment. Table 6 shows the subsets of attributes identified by the project team in the earlier part of this report, as well as the average ranking of each attribute in the subset.

Table 6: Attributes for each subset and their corresponding ranks

Subset	Attributes (Average Rank)
Subset 1 (S1)	Pstatus (14), Medu (19), **Mjob (8)**, Fedu (15), Fjob (23), guardian (21), famsize (30), famrel (19), famsup (17)
Subset 2 (S2)	sex (16), age (31), traveltime (20), studytime (24), failures (24), activities (13), paidclass (19), internet (13), higher (26), romantic (5), freetime (13), **goout (7)**, health (18), absences (20)
Subset 3 (S3)	Attributes in S1, G1 (2), **G2 (1)**
Subset 4 (S4)	Attributes in S2, G1 (2), **G2 (1)**
Subset 5 (S5)	G1 (2), **G2 (1)**

The attributes with the highest average rank in each subset is shown in bold. For both S1 and S2, the attributes are generally of relatively lower importance. As for S5, it contains G1 and G2, which are the second-most significant attribute and the most significant attribute respectively. Based on the rankings of the attributes in each subset, it is most probable that S1 and S2 will produce the worst predictive performance for most of the classifiers. By including highly relevant attributes such as G1 and G2, the classifiers should perform better when using S3 and S4. As for S5, since it only contains the top two most significant attributes, it should produce the best predictive performance for most of the classifiers.

Table 7: PCC using different subsets for binary and grade classification tasks

Subset	Classifiers (Binary Classification)					
	Naïve Bayes	Decision Tree	Random Forest	Neural Network	SVM	k-NN
Baseline	89.62%	**92.66%**	**92.66%**	90.89%	89.87%	76.96%
S1	73.42%	**74.18%**	66.33%	66.33%	**74.18%**	71.90%
S2	74.43%	**76.96%**	76.20%	70.89%	74.94%	71.65%
S3	90.38%	**92.91%**	90.63%	88.35%	92.15%	78.99%
S4	91.39%	**93.17%**	92.15%	87.09%	90.89%	77.98%
S5	90.13%	93.17%	**94.18%**	93.17%	92.91%	93.92%
Subset	**Classifiers (Grade Classification)**					
	Naïve Bayes	Decision Tree	Random Forest	Neural Network	SVM	k-NN
Baseline	71.14%	**79.75%**	75.70%	55.95%	69.37%	32.15%
S1	22.03%	20.25%	21.27%	22.03%	23.54%	**24.05%**
S2	28.86%	27.85%	**29.87%**	25.32%	26.58%	27.85%
S3	75.44%	**80.76%**	73.92%	67.09%	76.46%	38.48%
S4	74.43%	**80.51%**	77.98%	70.13%	75.70%	41.01%
S5	76.46%	79.75%	**80.51%**	80.00%	80.00%	80.25%

Table 7 shows the PCC for each classifier using the different subsets as the input features for both the binary and grade classification tasks. The best PCC for each subset, as well as the baseline set of features, are shown in bold.

For both the binary and grade classification tasks, it can be observed that S1 and S2 produced the worst predictive performance for the various classifiers. This provides the confirmation to the initial conjecture that S1 and S2 will produce the worst predictive performance for most of the classifiers due to the rankings of the attributes contained in those subsets.

For the binary classification task, the best predictive performance of 94.18% is obtained using the Random Forest algorithm on S5. Generally, for the subsets that include the two most significant attributes, namely G1 and G2, the predictive performance tends to be much better. For the grade classification task, the best predictive performance of 80.76% is achieved using the Decision Tree algorithm on S3. Similarly, the subsets that include G1 and G2 do perform significantly better than the other subsets. This proves the correlation between a well-chosen subset of attributes and the optimal predictive performance of the various classifiers.

One other noteworthy finding obtained from this experiment is the predictive performance of the k-NN algorithm when using highly relevant features such as those in S5. The PCC of the k-NN algorithm using S5 is 93.92% for the binary classification task, and 80.25% for the grade classification task. This demonstrates how a lazy learner, such as k-NN in this case, can perform substantially well when given the right set of input features.

Furthermore, it should also be noted that simpler algorithms such as Decision Tree or Random Forest can perform just as well, if not better than, complicated algorithms such as Neural Network or SVM.

Conclusion

In this project, the project team identified the various shortcomings of a previous study and developed a set of experiments to improve on the results. Besides implementing the four data mining algorithms used in the previous study, the project team proposed the use of two additional data mining algorithms, namely Naïve Bayes and k-NN, to further examine the different choices of algorithms for a classification task. Extensive effort have been devoted to the implementation of the different data mining algorithms to ensure that its performance will be comparable to that of the previous study.

The first experiment looked at using the wrapper approach to assess the relative importance of each attribute in the set of input features available. The different classification algorithms were ranked based on their accuracy for the baseline set of input features, and a weight vector was derived to be used for the calculation of the overall accuracy gain/loss when evaluating each attribute. The findings from the first experiment showed that there are some attributes that are indeed more relevant than the other attributes. The project team also assessed the impact of removing the least significant attributes and the most significant attributes from the set of input features on the predictive performance of the various classifiers.

The second experiment focused on how the choice of input features are correlated with the eventual predictive performance of the various classifiers. In particular, S5, which includes the two most relevant attributes G1 and G2, tends to provide the most optimal predictive performance for both the binary and grade classification tasks. The project team also examined how the different classification algorithms performed using different subsets of attributes. The PCC of 94.18% obtained using the Random Forest algorithm on S5 for the binary classification task is more than 2% higher than the PCC of 91.9% achieved using the Naïve classifier in the previous study. At the same time, the PCC of 80.76% attained using the Decision Tree algorithm on S3 for the grade classification task is also more than 2% higher than the PCC of 78.5% obtained using the Naïve classifier in the previous study.

With the increased emphasis on education, the ability to accurately predict and identify factors related to student performance will provide ample opportunities for improving the quality of education and hopefully, the academic performance of students. The project has shown how various data mining algorithms can be applied to extract interesting patterns and trends from raw data, and at the same time, provide a deeper analysis of the available information. As the number of data samples used in this project is relatively small, there is a potential for more accurate results to be obtained when a larger dataset is used. Furthermore, the achievable accuracy and training time required for each of these algorithms with respect to size of the dataset can be studied. More research can also be done to explore the use of an ensemble of different classifiers to improve the predictive performance, as the classifiers used in this project are evaluated and utilized independently.

References

[1] P. Cortez and A. Silva, 'USING DATA MINING TO PREDICT SECONDARY SCHOOL STUDENT PERFORMANCE'. [Online]. Available: http://www3.dsi.uminho.pt/pcortez/student.pdf. [Accessed: 05- Nov- 2015].

[2] 'UCI Machine Learning Repository: Student Performance Data Set'. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Student+Performance. [Accessed: 05- Nov- 2015].

[3] 'Weka 3 - Data Mining with Open Source Machine Learning Software in Java'. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/index.html. [Accessed: 05- Nov- 2015].

[4] 'J48'. [Online]. Available: http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html. [Accessed: 05- Nov- 2015].

[5] 'RandomForest'. [Online]. Available: http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/RandomForest.html. [Accessed: 05- Nov- 2015].

[6] 'MultilayerPerceptron'. [Online]. Available: http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/MultilayerPerceptron.html. [Accessed: 05- Nov- 2015].

[7] 'SMO'. [Online]. Available: http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/SMO.html. [Accessed: 05- Nov- 2015].

[8] 'NaiveBayes'. [Online]. Available: http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/NaiveBayes.html. [Accessed: 05- Nov- 2015].

[9] 'IBk'. [Online]. Available: http://weka.sourceforge.net/doc.dev/weka/classifiers/lazy/IBk.html. [Accessed: 05- Nov- 2015].

[10] S. Drazin and M. Montag, 'Decision Tree Analysis using Weka'. [Online]. Available: http://ww.samdrazin.com/classes/een548/project2report.pdf. [Accessed: 05- Nov- 2015].

[11] S. Bernard, L. Heutte and S. Adam, 'Influence of Hyperparameters on Random Forest Accuracy', *Multiple Classifier Systems*, pp. 171-180, 2009.

Appendix A: Full Set of Attributes

Attribute	Description	Data Type	Possible Values
sex	Student's Gender	Binary	Male, Female
age	Student's Age	Numeric	[15, 22]
school	Student's School	Binary	Gabriel Pereira, Mousinho da Silveria
address	Student's Home Address Type	Binary	Urban, Rural
Pstatus	Parent's Cohabitation Status	Binary	Living Together, Living Apart
Medu	Mother's Education	Numeric	[0, 4]: 0 – None, 1 – Primary Education (4th Grade), 2 – 5th to 9th Grade, 3 – Secondary Education, 4 – Higher Education
Mjob	Mother's Job	Nominal	Teacher, Healthcare-related, Civil Services, At Home, Other
Fedu	Father's Education	Numeric	[0, 4]: 0 – None, 1 – Primary Education (4th Grade), 2 – 5th to 9th Grade, 3 – Secondary Education, 4 – Higher Education
Fjob	Father's Job	Nominal	Teacher, Healthcare-related, Civil Services, At Home, Other
guardian	Student's Guardian	Nominal	Mother, Father, Other
famsize	Family Size	Binary	≤ 3, > 3
famrel	Quality of Family Relationships	Numeric	[1, 5]: 1 – Very Bad, 5 - Excellent
reason	Reason for Choosing School	Nominal	Close to Home, School Reputation, Course Preference, Other
traveltime	Home to School Travel Time	Numeric	[1, 4]: 1 – Less than 15 minutes, 2 – 15 to 30 minutes, 3 – 30 minutes to 1 hour, 4 – More than 1 hour
studytime	Weekly Study Time	Numeric	[1, 4]: 1 – Less than 2 hours, 2 – 2 to 5 hours, 3 – 5 to 10 hours, 4 – More than 10 hours
failures	Number of Past Class Failures	Numeric	[1, 4]: 1 – 1 failure, 2 – 2 failures, 3 – 3 failures, 4 – More than 4 failures

Attribute	Description	Data Type	Possible Values
schoolsup	Extra Educational Support from School	Binary	Yes, No
famsup	Extra Educational Support from Family	Binary	Yes, No
activities	Extra-curricular Activities	Binary	Yes, No
paidclass	Extra Paid Classes	Binary	Yes, No
internet	Internet Access at Home	Binary	Yes, No
nursery	Attended Nursery School	Binary	Yes, No
higher	Intention to Pursue Higher Education	Binary	Yes, No
romantic	In a Romantic Relationship	Binary	Yes, No
freetime	Free Time After School	Numeric	[1, 5]: 1 – Very Low, 5 – Very High
goout	Time Spent Going Out with Friends	Numeric	[1, 5]: 1 – Very Low, 5 – Very High
Walc	Weekend Alcohol Consumption	Numeric	[1, 5]: 1 – Very Low, 5 – Very High
Dalc	Weekday Alcohol Consumption	Numeric	[1, 5]: 1 – Very Low, 5 – Very High
health	Current Health Status	Numeric	[1, 5]: 1 – Very Bad, 5 – Very Good
absences	Number of School Absences	Numeric	[0, 93]
G1	First Period Grade	Numeric	[0, 20]
G2	Second Period Grade	Numeric	[0, 20]
G3	Final Grade	Numeric	[0, 20]

Appendix B: Feature Evaluation Results

Binary Classification								
Rank	Attribute	Naïve Bayes	Decision Tree	Random Forest	Neural Network	SVM	k-NN	Overall
1	G2	87.34%	88.35%	86.08%	83.04%	82.03%	74.68%	-39.65%
2	G1	88.61%	90.13%	90.89%	84.30%	87.34%	75.95%	-19.75%
3	goout	89.62%	92.66%	92.15%	84.81%	89.87%	75.95%	-9.62%
4	internet	90.13%	92.66%	92.41%	86.33%	88.86%	74.94%	-8.96%
5	romantic	89.11%	92.66%	92.15%	87.60%	89.37%	74.43%	-8.68%
6	Fedu	89.62%	92.91%	92.66%	85.82%	89.37%	75.70%	-8.10%
7	Walc	89.62%	92.91%	92.15%	86.58%	88.35%	76.96%	-7.77%
8	traveltime	89.62%	92.66%	91.65%	85.82%	90.38%	76.71%	-7.65%
9	schoolsup	90.38%	92.66%	92.66%	85.57%	90.38%	74.94%	-7.49%
10	Mjob	89.87%	92.66%	93.17%	85.82%	90.63%	74.43%	-7.22%
11	reason	90.13%	92.66%	92.66%	86.58%	89.62%	75.19%	-7.11%
12	paid	90.13%	92.66%	91.65%	85.82%	90.89%	76.46%	-6.73%
13	sex	89.87%	92.66%	92.15%	86.33%	90.63%	76.20%	-6.20%
14	address	90.38%	92.66%	91.39%	87.34%	90.38%	75.95%	-5.95%
15	health	89.62%	92.66%	92.66%	87.60%	89.62%	75.70%	-5.85%
16	activities	90.13%	92.66%	92.41%	86.58%	90.38%	75.95%	-5.80%
17	freetime	89.62%	93.17%	92.15%	86.58%	90.38%	76.20%	-5.75%
18	Fjob	89.87%	92.66%	92.15%	88.10%	91.14%	73.92%	-5.57%
19	Medu	89.62%	92.66%	92.41%	87.34%	89.87%	76.46%	-5.47%
20	nursery	89.62%	92.66%	92.15%	87.60%	90.63%	75.70%	-5.34%
21	Pstatus	90.13%	92.66%	92.41%	86.58%	89.87%	77.22%	-5.14%
22	famsup	90.13%	92.66%	92.66%	87.09%	90.13%	75.95%	-5.09%
23	higher	89.87%	92.66%	92.15%	87.34%	89.62%	77.22%	-5.09%
24	famrel	89.62%	92.66%	92.66%	88.10%	90.38%	75.44%	-4.53%
25	absences	89.11%	92.66%	92.15%	88.61%	89.87%	76.71%	-4.48%
26	Dalc	89.62%	92.66%	92.41%	87.85%	90.13%	76.71%	-4.25%
27	guardian	89.62%	92.66%	92.91%	87.85%	90.63%	75.44%	-4.20%
28	failures	90.13%	92.66%	92.15%	87.85%	90.89%	75.95%	-3.90%
29	age	89.62%	92.66%	92.91%	87.60%	90.38%	76.46%	-3.82%
30	studytime	89.62%	92.66%	92.41%	88.10%	91.39%	76.20%	-2.91%
31	famsize	91.14%	92.66%	92.91%	86.58%	90.13%	77.47%	-2.76%
32	school	90.13%	92.66%	93.17%	88.35%	90.13%	76.71%	-1.98%

Grade Classification								
Rank	Attribute	Naïve Bayes	Decision Tree	Random Forest	Neural Network	SVM	k-NN	Overall
1	G2	51.14%	56.71%	51.90%	43.54%	50.13%	28.86%	-133.90%
2	G1	68.35%	78.99%	69.62%	54.94%	65.82%	29.87%	-20.91%
3	nursery	70.63%	79.75%	72.15%	52.66%	68.61%	30.63%	-11.67%
4	romantic	71.90%	79.75%	73.17%	51.65%	69.11%	29.87%	-9.87%
5	reason	71.90%	79.75%	72.91%	54.43%	68.10%	29.37%	-8.89%
6	Mjob	70.38%	79.75%	75.95%	53.17%	68.10%	28.61%	-8.76%
7	Pstatus	71.39%	79.75%	72.91%	54.43%	70.38%	28.35%	-7.82%
8	freetime	71.14%	79.75%	74.68%	54.18%	68.10%	30.63%	-6.41%
9	school	72.15%	79.75%	73.17%	55.95%	68.10%	32.41%	-3.49%
10	activities	71.65%	79.75%	71.65%	58.99%	69.11%	30.63%	-3.49%
11	goout	71.14%	79.75%	73.67%	56.71%	70.38%	29.87%	-3.06%
12	famsup	72.41%	79.75%	74.18%	54.68%	71.39%	29.37%	-2.23%
13	famrel	71.14%	79.75%	73.67%	56.20%	69.87%	31.90%	-2.20%
14	guardian	70.89%	80.51%	71.90%	55.44%	70.63%	33.67%	-2.03%
15	absences	71.14%	80.00%	71.39%	58.99%	69.87%	31.90%	-1.95%
16	Walc	71.14%	79.75%	74.43%	59.49%	67.34%	30.63%	-1.82%
17	studytime	71.14%	79.75%	74.43%	57.98%	68.86%	30.63%	-1.67%
18	sex	71.90%	79.75%	72.91%	56.71%	71.39%	30.38%	-1.42%
19	Medu	71.14%	79.75%	73.42%	60.00%	68.61%	30.89%	-0.91%
20	failures	71.90%	79.75%	73.92%	55.95%	71.39%	30.38%	-0.84%
21	health	71.14%	79.75%	74.43%	57.98%	69.87%	30.38%	-0.71%
22	internet	72.15%	79.75%	73.42%	57.47%	71.14%	29.62%	-0.61%
23	address	71.39%	79.75%	73.67%	58.99%	69.11%	31.14%	-0.48%
24	Fedu	71.14%	79.75%	73.17%	59.75%	69.62%	30.89%	-0.33%
25	schoolsup	72.41%	79.75%	73.92%	57.22%	70.63%	30.38%	0.30%
26	paid	71.39%	79.75%	73.42%	57.72%	69.62%	33.17%	0.41%
27	Fjob	72.15%	79.75%	74.18%	55.70%	71.14%	31.65%	0.53%
28	higher	71.14%	79.75%	71.90%	60.25%	69.87%	32.91%	0.79%
29	famsize	72.41%	79.75%	74.18%	57.72%	70.13%	31.65%	1.87%
30	Dalc	71.14%	79.75%	74.18%	61.27%	70.13%	30.13%	2.61%
31	traveltime	71.14%	80.25%	74.18%	58.99%	69.62%	32.66%	2.79%
32	age	71.14%	79.75%	74.43%	58.48%	70.89%	32.41%	3.09%

Appendix C: Source Code

RemovedAttribute.java

```
public class RemovedAttribute implements Comparable<RemovedAttribute> {

	private String _attrName = null;

	private HashMap<String, Double> _accuracies = null;
	private double _accuracyGainLoss = 0;

	public RemovedAttribute(String attrName) {

		_attrName = attrName;
		_accuracies = new HashMap<>();
	}

	public String getAttributeName() {
		return _attrName;
	}

	public void setAttributeName(String attrName) {
		_attrName = attrName;
	}

	public void addAccuracy(String technique, double accuracy) {
		_accuracies.put(technique, accuracy);
	}

	public double getAccuracy(String technique) {
		return _accuracies.get(technique);
	}

	public HashMap<String, Double> getAccuracies() {
		return _accuracies;
	}

	public double getAccuracyGainLoss() {
		return _accuracyGainLoss;
	}
```

```
	public void calcAccuracyGainLoss(RemovedAttribute baselineAttr,
	                                 double[] weights) {

		_accuracyGainLoss = 0;

		int currAttr = 0;
		HashMap<String, Double> baselineAccuracies =
			baselineAttr.getAccuracies();
		for(String technique : Const.MODELS) {

			double baselineAccuracy = baselineAccuracies.get(technique);
			double selfAccuracy = getAccuracy(technique);

			double gainLoss = selfAccuracy - baselineAccuracy;

			if(weights != null)
				_accuracyGainLoss += (weights[currAttr++] * gainLoss);
			else
				_accuracyGainLoss += gainLoss;
		}
	}

	@Override
	public String toString() {

		StringBuilder sb = new StringBuilder();

		sb.append(String.format("%-18s", _attrName)).append(" | ");
		for(String key : Const.MODELS) {

			sb.append(String.format("%-13s",
					String.format("%5.3f %%",
					_accuracies.get(key))));
			sb.append(" | ");
		}
		sb.append(String.format("%-13s",
				String.format("%+5.3f %%", _accuracyGainLoss)));

		return sb.toString();
	}

	@Override
	public int compareTo(RemovedAttribute other) {

		double otherAccuracyGainLoss = other.getAccuracyGainLoss();

		if(otherAccuracyGainLoss > _accuracyGainLoss)
			return 1;
		else if(otherAccuracyGainLoss < _accuracyGainLoss)
			return -1;
		else
			return 0;
	}
}
```

Const.java

```
public class Const {

	public static final String MATH_BINARY_FILENAME =
			"data\\student_math_binary.arff";
	public static final String MATH_GRADE_FILENAME =
			"data\\student_math_grade.arff";

	public static final String ATTR_EVAL_RESULTS_FILENAME =
			"data\\student_math_attrEvalResults.txt";

	public static final String GOOD_BAD_ATTR_FILENAME =
			"data\\student_math_goodBadAttr.txt";

	public static final String SUBSETS_FILENAME =
			"data\\subsets.txt";
	public static final int NUM_SUBSETS = 5;

	public static final String SUBSETS_EVAL_RESULTS_FILENAME =
			"data\\student_math_subsetEvalResults.txt";

	public static final int TOP_N = 5;

	public static final int CROSS_VALIDATION_FOLDS = 10;

	public static final String[] MODELS = {
		"Naive Bayes",
		"Decision Tree",
		"Random Forest",
		"NN (MLP)",
		"SVM",
		"k-NN (k = 9)"
	};
}
```

FileIOHelper.java

```
/**
 * Utility class to help with file I/O operations
 */
public class FileIOHelper {

	/**
	 * Appends the given contents to an existing file (if any)<br>
	 * Creates a new file and stores the contents if no such file exists
	 *
	 * @param content   Contents to be saved to file
	 * @param fileName  Name of file
	 */
	public static void appendToFile(String content, String fileName) {

		saveToFile(content, fileName, false);
	}

	/**
	 * Stores the given contents in a file
	 *
	 * @param content   Contents to be saved to file
	 * @param fileName  Name of file
	 * @param bAppend   Indicates whether to append or truncate
	 */
	public static void saveToFile(String content, String fileName,
	                              boolean bAppend) {

		try {
			FileWriter fw = new FileWriter(new File(fileName), bAppend);
			fw.write(content);
			fw.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	/**
	 * Returns a BufferedReader to the given input file
	 *
	 * @param filename  File name
	 * @return   BufferedReader to the given input file
	 */
	public static BufferedReader readDataFile(String filename) {

		BufferedReader inputReader = null;
		try {
			inputReader = new BufferedReader(new FileReader(filename));
		} catch (FileNotFoundException ex) {
			System.err.println("File not found: " + filename);
		}
		return inputReader;
	}
}
```

WekaHelper.java

```
public class WekaHelper {

	private static Evaluation classify(Classifier model, Instances trainingSet,
			Instances testingSet) {

		Evaluation evaluation = null;
		try {
			evaluation = new Evaluation(trainingSet);
			model.buildClassifier(trainingSet);
			evaluation.evaluateModel(model, testingSet);
		} catch (Exception e) {
			e.printStackTrace();
		}

		return evaluation;
	}

	/**
	 * Returns the accuracy of the predictions<br>
	 * [0, 100]%
	 *
	 * @param predictions The set of predictions
	 * @return Accuracy of predictions
	 */
	private static double calculateAccuracy(
			ArrayList<NominalPrediction> predictions) {

		double correct = 0;

		for (int i = 0; i < predictions.size(); i++) {
			NominalPrediction np = predictions.get(i);
			if (np.predicted() == np.actual()) {
				correct++;
			}
		}

		return 100 * (correct / predictions.size());
	}

	/**
	 * Splits the data into training set and testing set
	 */
	private static Instances [][] crossValidationSplit(Instances data,
			int numberOfFolds) {

		Instances [][] split = new Instances[2][numberOfFolds];

		for (int i = 0; i < numberOfFolds; i++) {
			split[0][i] = data.trainCV(numberOfFolds, i);
			split[1][i] = data.testCV(numberOfFolds, i);
		}

		return split;
	}
```

```
public static RemovedAttribute calcAccuracy(Instances dataInstances) {

	RemovedAttribute attr = new RemovedAttribute("None");

	// Set last attribute as class index
	dataInstances.setClassIndex(
			dataInstances.numAttributes() - 1);

	calcAccuracies(dataInstances, attr);
	return attr;
}

public static void removeAttrAndCalc(Instances dataInstances,
			ArrayList<RemovedAttribute> attrList) {

	int numAttributes = dataInstances.numAttributes();
	Remove removeAttrFilter = new Remove();

	for (int attrIndex = 0; attrIndex < numAttributes - 1; attrIndex++)
	{

		String removedAttrName =
				dataInstances.attribute(attrIndex).name();
		System.out.println("Removed attribute '" + removedAttrName +
				"'.. Calculating accuracies after removing
				attribute....");
		RemovedAttribute removedAttr = new
			RemovedAttribute(removedAttrName);

		// Remove one of the attributes
		removeAttrFilter.setAttributeIndices(
				String.valueOf(attrIndex + 1));
		Instances filteredData = null;
		try {
			removeAttrFilter.setInputFormat(dataInstances);
			filteredData = Filter.useFilter(
					dataInstances, removeAttrFilter);
			filteredData.setClassIndex(
				filteredData.numAttributes() - 1);
		} catch (Exception e) {
			e.printStackTrace();
		}

		calcAccuracies(filteredData, removedAttr);
		attrList.add(removedAttr);
	}
}
```

```
private static void calcAccuracies(Instances dataInstances,
                                   RemovedAttribute attr) {

      // Separate data instances into training and testing splits
      Instances[][] split = crossValidationSplit(dataInstances,
                  Const.CROSS_VALIDATION_FOLDS);
      Instances[] trainingSet = split[0];
      Instances[] testingSet = split[1];

      Classifier[] models =
            ClassificationModels.getInstance().getModels();

      // Run for each model
      for (int currModel = 0; currModel < models.length; currModel++) {

            // Collect all the predictions for current model
            ArrayList<NominalPrediction> predictions = new ArrayList<>();

            // For each training-testing split pair,
            // train and test the classifier
            for (int i = 0; i < trainingSet.length; i++) {

                  Evaluation validation = classify(
                              models[currModel], trainingSet[i],
                              testingSet[i]);

                  for(Object o : validation.predictions()) {
                        if(o instanceof NominalPrediction)
                              predictions.add((NominalPrediction) o);
                  }
            }

            // Calculate overall accuracy of current classifier on all
            // splits
            double accuracy = calculateAccuracy(predictions);

            // Store accuracy
            attr.addAccuracy(Const.MODELS[currModel], accuracy);
      }
}
```

```
public static int[] rankTechniques(RemovedAttribute baseline) {

	int numTechniques = Const.MODELS.length;
	int[] ranks = new int[numTechniques];

	for(int i = 0; i < numTechniques; i++) {

		int rank = 1;
		for(int j = 0; j < numTechniques; j++) {

			double accuracy1 =
				baseline.getAccuracy(Const.MODELS[i]);
			double accuracy2 = b
				aseline.getAccuracy(Const.MODELS[j]);

			if(accuracy1 > accuracy2)
				rank++;
		}

		ranks[i] = (numTechniques - rank + 1);
	}

	return ranks;
}

public static double[] rankToWeight(int[] ranks) {

	int numTechniques = Const.MODELS.length;
	double[] weights = new double[ranks.length];

	for(int i = 0; i < weights.length; i++) {
		weights[i] = 1.0 + ((numTechniques - ranks[i]) * 0.1);
	}

	return weights;
}

public static String getColumnHeaders() {

	StringBuilder sb = new StringBuilder();

	sb.append(String.format("%n%-18s", "Removed Attribute")).
		append(" | ");
	for(String model : Const.MODELS) {

		sb.append(String.format("%-13s", model));
		sb.append(" | ");
	}
	sb.append(String.format("%-13s", "Gain/Loss"));

	return sb.toString();
}

public static void printLine(StringBuilder sb, String contents) {

	sb.append(contents).append("\n");
	System.out.println(contents);
}
```

```
	public static String[] concat(String[] a, String[] b) {
		int aLen = a.length;
		int bLen = b.length;
		String[] c = new String[aLen + bLen];
		System.arraycopy(a, 0, c, 0, aLen);
		System.arraycopy(b, 0, c, aLen, bLen);
		return c;
	}
}
```

m

ClassificationModels.java

```
public class ClassificationModels {

	private static ClassificationModels _instance = null;
	private static Classifier[] _models = null;

	// Classifiers
	private NaiveBayes _naiveBayes = null;
	private J48 _decisionTree = null;
	private RandomForest _randomForest = null;
	private MultilayerPerceptron _neuralNetwork = null;
	private SMO _svm = null;
	private IBk _kNN = null;

	/** Determines if the default or optimized classification models
	*  will be used **/
	private boolean _bOptimized = true;

	private ClassificationModels() {

		_naiveBayes = new NaiveBayes();
		if(_bOptimized) {
			// Default:		No supervised discretization
			// Optimizied:	Supervised discretization
			_naiveBayes.setUseSupervisedDiscretization(true);
		}

		_decisionTree = new J48();
		if(_bOptimized) {
			// Default:		Confidence factor of 0.25
			//			Mininum number of objects is 2
			// Optimized:	Confidence factor of 0.175
			//			Mininum number of objects is 13
			_decisionTree.setConfidenceFactor(0.175f);
			_decisionTree.setMinNumObj(13);
		}

		_randomForest = new RandomForest();
		if (_bOptimized) {
			// Default:		Number of features is 0 (unlimited)
			//			Maximum depth is 0 (unlimited)
			//			Number of trees is 100
			// Optimized:	Number of features is 10
			//			Maximum depth is 8
			//			Number of trees is 200
			_randomForest.setNumFeatures(10);
			_randomForest.setMaxDepth(8);
			_randomForest.setNumTrees(200);
		}
```

```
            _neuralNetwork = new MultilayerPerceptron();
            if (_bOptimized) {
                  // Default:         1 Hidden layer with [(# of attributes + #
                                      of classes)/2] neurons
                  //                  Learning rate of 0.3
                  // Optimized:       1 Hidden layer with 5 neurons
                  //                  Learning rate of 0.09
                  _neuralNetwork.setHiddenLayers("5");
                  _neuralNetwork.setLearningRate(0.09);
            }

            _svm = new SMO();
            if(_bOptimized) {
                  // Default:         All attributes normalized
                  // Optimized:       No normalization/standardization
                  _svm.setFilterType(new SelectedTag(
                              SMO.FILTER_NONE, SMO.TAGS_FILTER));
            }

            // Using k-NN with k-value of 9, and weight neighbours using
            // 1/distance
            _kNN = new IBk();
            if (_bOptimized) {
                  // Default:         k value is 1
                  //                  No distance weighting performed
                  // Optimized:       k value is 9
                  //                  Neighbours are inversely weighted, i.e.
                                      1/distance
                  _kNN.setKNN(9);
                  _kNN.setDistanceWeighting(new SelectedTag(
                              IBk.WEIGHT_INVERSE, IBk.TAGS_WEIGHTING));
            }

            _models = new Classifier[] {
                        _naiveBayes, _decisionTree,
                        _randomForest, _neuralNetwork,
                        _svm, _kNN
            };
      }

      public static ClassificationModels getInstance() {

            if(_instance == null) {
                  _instance = new ClassificationModels();
            }

            return _instance;
      }

      public Classifier[] getModels() {

            return _models;
      }
}
```

WekaAttributeEvaluator.java

```
public class WekaAttributeEvaluator {

	private static RemovedAttribute _binaryBaseline = null;
	private static ArrayList<RemovedAttribute> _binaryAttrList = null;

	private static RemovedAttribute _gradeBaseline = null;
	private static ArrayList<RemovedAttribute> _gradeAttrList = null;

	private static StringBuilder _sbAttrEval = new StringBuilder();
	private static StringBuilder _sbGoodBadAttr = new StringBuilder();

	public static void main(String[] args) {

		BufferedReader binaryDataFile = FileIOHelper
				.readDataFile(Const.MATH_BINARY_FILENAME);
		Instances binaryDataInstances = null;
		try {
			binaryDataInstances = new Instances(binaryDataFile);
		} catch (IOException e) {
			e.printStackTrace();
		}

		if (binaryDataInstances != null) {

			System.out.println(
					"Calculating baseline accuracies for binary
					data....");
			_binaryBaseline =
					WekaHelper.calcAccuracy(binaryDataInstances);
			int[] ranks = WekaHelper.rankTechniques(_binaryBaseline);
			double[] weights = WekaHelper.rankToWeight(ranks);

			StringBuilder sb = new StringBuilder();
			sb.append("Binary Data (Technique/Rank/Weight): ");
			for(int i = 0; i < Const.MODELS.length; i++) {
				sb.append("(");
				sb.append(Const.MODELS[i]).append("/");
				sb.append(ranks[i]).append("/").append(weights[i]);
				sb.append("), ");
			}
			sb.delete(sb.lastIndexOf(","), sb.length());
			sb.append("\n");

			WekaHelper.printLine(_sbAttrEval, sb.toString());

			_binaryAttrList = new ArrayList<>();
			WekaHelper.removeAttrAndCalc(binaryDataInstances,
				_binaryAttrList);

			WekaHelper.printLine(_sbAttrEval,
				WekaHelper.getColumnHeaders());
			WekaHelper.printLine(_sbAttrEval, _binaryBaseline.toString());

			// Calculate accuracy gain/loss
			for(RemovedAttribute ra : _binaryAttrList) {
				ra.calcAccuracyGainLoss(_binaryBaseline, weights);
			}
```

```
		Collections.sort(_binaryAttrList);
		for(RemovedAttribute ra : _binaryAttrList) {
			WekaHelper.printLine(_sbAttrEval, ra.toString());
		}

		// Save TOP_N Least Significant Attributes to the file
		// i.e. Removed attributes resulting in most accuracy gain
		for(int i = 0; i < Const.TOP_N; i++) {

			_sbGoodBadAttr.append(
				_binaryAttrList.get(i).getAttributeName());
			_sbGoodBadAttr.append(",");
		}
		_sbGoodBadAttr.deleteCharAt(_sbGoodBadAttr.lastIndexOf(","));
		_sbGoodBadAttr.append("\n");

		// Save TOP_N Most Significant Attributes to the file
		// i.e. Removed attributes resulting in most accuracy loss
		for(int i = _binaryAttrList.size() - 1;
				i > (_binaryAttrList.size() - 1 - Const.TOP_N);
				i--) {

			_sbGoodBadAttr.append(
				_binaryAttrList.get(i).getAttributeName());
			_sbGoodBadAttr.append(",");
		}
		_sbGoodBadAttr.deleteCharAt(_sbGoodBadAttr.lastIndexOf(","));
		_sbGoodBadAttr.append("\n");
}

BufferedReader gradeDataFile = FileIOHelper
			.readDataFile(Const.MATH_GRADE_FILENAME);
Instances gradeDataInstances = null;
try {
		gradeDataInstances = new Instances(gradeDataFile);
} catch (IOException e) {
		e.printStackTrace();
}

if (gradeDataInstances != null) {

		System.out.println(
				"\n\nCalculating baseline accuracies for grade
				data....");
		_gradeBaseline = WekaHelper.calcAccuracy(gradeDataInstances);
		int[] ranks = WekaHelper.rankTechniques(_gradeBaseline);
		double[] weights = WekaHelper.rankToWeight(ranks);

		StringBuilder sb = new StringBuilder();
		sb.append("Grade Data (Technique/Rank/Weight): ");
		for(int i = 0; i < Const.MODELS.length; i++) {
			sb.append("(");
			sb.append(Const.MODELS[i]).append("/");
			sb.append(ranks[i]).append("/").append(weights[i]);
			sb.append("), ");
		}
```

```
                sb.delete(sb.lastIndexOf(","), sb.length());
                sb.append("\n");

                WekaHelper.printLine(_sbAttrEval, sb.toString());

                _gradeAttrList = new ArrayList<>();
                WekaHelper.removeAttrAndCalc(gradeDataInstances,
                        _gradeAttrList);

                WekaHelper.printLine(_sbAttrEval,
                        WekaHelper.getColumnHeaders());
                WekaHelper.printLine(_sbAttrEval, _gradeBaseline.toString());

                // Calculate accuracy gain/loss
                for(RemovedAttribute ra : _gradeAttrList) {
                        ra.calcAccuracyGainLoss(_gradeBaseline, weights);
                }

                Collections.sort(_gradeAttrList);
                for(RemovedAttribute ra : _gradeAttrList) {
                        WekaHelper.printLine(_sbAttrEval, ra.toString());
                }

                // Save TOP_N Least Significant Attributes to the file
                // i.e. Removed attributes resulting in most accuracy gain
                for(int i = 0; i < Const.TOP_N; i++) {

                        _sbGoodBadAttr.append(
                                        _gradeAttrList.get(i).getAttributeName());
                        _sbGoodBadAttr.append(",");
                }
                _sbGoodBadAttr.deleteCharAt(_sbGoodBadAttr.lastIndexOf(","));
                _sbGoodBadAttr.append("\n");

                // Save TOP_N Most Significant Attributes to the file
                // i.e. Removed attributes resulting in most accuracy loss
                for(int i = _gradeAttrList.size() - 1;
                                i > (_gradeAttrList.size() - 1 - Const.TOP_N);
                                i--) {

                        _sbGoodBadAttr.append(
                                        _gradeAttrList.get(i).getAttributeName());
                        _sbGoodBadAttr.append(",");
                }
                _sbGoodBadAttr.deleteCharAt(_sbGoodBadAttr.lastIndexOf(","));
                _sbGoodBadAttr.append("\n");
        }

        FileIOHelper.saveToFile(_sbAttrEval.toString(),
                        Const.ATTR_EVAL_RESULTS_FILENAME, false);
        FileIOHelper.saveToFile(_sbGoodBadAttr.toString(),
                        Const.GOOD_BAD_ATTR_FILENAME, false);
    }
}
```

WekaSubsetEvaluator.java

```
public class WekaSubsetEvaluator {

	private static String[] _binaryLSA = null;
	private static String[] _binaryMSA = null;

	private static String[] _gradeLSA = null;
	private static String[] _gradeMSA = null;

	private static String[][] _subsetsAttr = null;

	private static StringBuilder _sbSubsetEval = new StringBuilder();

	public static void main(String[] args) {

		BufferedReader goodBadAttrFile =
			FileIOHelper.readDataFile(Const.GOOD_BAD_ATTR_FILENAME);
		try {

			_binaryLSA = goodBadAttrFile.readLine().split(",");
			_binaryMSA = goodBadAttrFile.readLine().split(",");

			_gradeLSA = goodBadAttrFile.readLine().split(",");
			_gradeMSA = goodBadAttrFile.readLine().split(",");

		} catch (IOException e) {
			e.printStackTrace();
		}

		BufferedReader subsetsFile =
				FileIOHelper.readDataFile(Const.SUBSETS_FILENAME);
		try {
			_subsetsAttr = new String[Const.NUM_SUBSETS][];
			for(int i = 0; i < Const.NUM_SUBSETS; i++) {
				_subsetsAttr[i] = subsetsFile.readLine().split(",");
			}

		} catch (IOException e) {
			e.printStackTrace();
		}

		BufferedReader binaryDataFile =
				FileIOHelper.readDataFile(Const.MATH_BINARY_FILENAME);
		Instances binaryDataInstances = null;
		try {
			binaryDataInstances = new Instances(binaryDataFile);
		} catch (IOException e) {
			e.printStackTrace();
		}
```

```
if (binaryDataInstances != null) {

	// Calculate baseline accuracy for binary data
	RemovedAttribute binaryBaseline =
			WekaHelper.calcAccuracy(binaryDataInstances);
	WekaHelper.printLine(_sbSubsetEval,
		WekaHelper.getColumnHeaders());
	WekaHelper.printLine(_sbSubsetEval,
		binaryBaseline.toString());

	// Remove Top 5 Least Significant Attributes
	String attrIndicesToRemove =
			findAttributeIndices(binaryDataInstances,
				_binaryLSA);
	Instances filteredBinaryData = removeUnwantedAttr(
			binaryDataInstances, attrIndicesToRemove);

	// Calculate accuracy after removing top 5 least significant
	// attributes
	RemovedAttribute binaryWithoutLSA =
			WekaHelper.calcAccuracy(filteredBinaryData);
	binaryWithoutLSA.setAttributeName("Binary Top 5 LSA");
	binaryWithoutLSA.calcAccuracyGainLoss(binaryBaseline, null);
	WekaHelper.printLine(_sbSubsetEval,
		binaryWithoutLSA.toString());

	// Remove Top 5 Most Significant Attributes
	attrIndicesToRemove = findAttributeIndices(
			binaryDataInstances, _binaryMSA);
	filteredBinaryData = removeUnwantedAttr(
			binaryDataInstances, attrIndicesToRemove);

	RemovedAttribute binaryWithoutMSA =
			WekaHelper.calcAccuracy(filteredBinaryData);
	binaryWithoutMSA.setAttributeName("Binary Top 5 MSA");
	binaryWithoutMSA.calcAccuracyGainLoss(binaryBaseline, null);
	WekaHelper.printLine(_sbSubsetEval,
		binaryWithoutMSA.toString());

	// Iterate through the different subsets
	for(int i = 0; i < Const.NUM_SUBSETS; i++) {

		// Keep only attributes specified in subset
		String attrIndicesToKeep = findAttributeIndices(
				binaryDataInstances,
				WekaHelper.concat(
					_subsetsAttr[i],
					new String[]{"BinaryData"}));

		filteredBinaryData = keepSpecifiedAttr(
				binaryDataInstances, attrIndicesToKeep);
```

```
                RemovedAttribute subset =
                        WekaHelper.calcAccuracy(filteredBinaryData);
                subset.setAttributeName("Binary !(Subset " + (i+1) +
                        ")");
                subset.calcAccuracyGainLoss(binaryBaseline, null);
                WekaHelper.printLine(_sbSubsetEval, subset.toString());
        }
}

BufferedReader gradeDataFile =
                FileIOHelper.readDataFile(Const.MATH_GRADE_FILENAME);
Instances gradeDataInstances = null;
try {
        gradeDataInstances = new Instances(gradeDataFile);
} catch (IOException e) {
        e.printStackTrace();
}

if (gradeDataInstances != null) {

        // Calculate baseline accuracy for 5-grade data
        RemovedAttribute gradeBaseline =
                        WekaHelper.calcAccuracy(gradeDataInstances);
        WekaHelper.printLine(_sbSubsetEval,
                        "\n" + WekaHelper.getColumnHeaders());
        WekaHelper.printLine(_sbSubsetEval, gradeBaseline.toString());

        // Remove Top 5 Least Significant Attributes
        String attrIndicesToRemove =
                        findAttributeIndices(gradeDataInstances,
                                _gradeLSA);
        Instances filteredGradeData = removeUnwantedAttr(
                        gradeDataInstances, attrIndicesToRemove);

        // Calculate accuracy after removing top 5 least significant
        // attributes
        RemovedAttribute gradeWithoutLSA =
                        WekaHelper.calcAccuracy(filteredGradeData);
        gradeWithoutLSA.setAttributeName("Grade Top 5 LSA");
        gradeWithoutLSA.calcAccuracyGainLoss(gradeBaseline, null);
        WekaHelper.printLine(_sbSubsetEval,
                gradeWithoutLSA.toString());

        // Remove Top 5 Most Significant Attributes
        attrIndicesToRemove = findAttributeIndices(
                        gradeDataInstances, _gradeMSA);
        filteredGradeData = removeUnwantedAttr(
                        gradeDataInstances, attrIndicesToRemove);

        RemovedAttribute gradeWithoutMSA =
                        WekaHelper.calcAccuracy(filteredGradeData);
        gradeWithoutMSA.setAttributeName("Grade Top 5 MSA");
        gradeWithoutMSA.calcAccuracyGainLoss(gradeBaseline, null);
        WekaHelper.printLine(_sbSubsetEval,
                gradeWithoutMSA.toString());
```

```
		// Iterate through the different subsets
		for(int i = 0; i < Const.NUM_SUBSETS; i++) {

			// Keep only attributes specified in subset
			String attrIndicesToKeep = findAttributeIndices(
					gradeDataInstances,
					WekaHelper.concat(
							_subsetsAttr[i],
							new String[]{"Grade"}));

			filteredGradeData = keepSpecifiedAttr(
					gradeDataInstances, attrIndicesToKeep);

			RemovedAttribute subset =
				WekaHelper.calcAccuracy(filteredGradeData);
			subset.setAttributeName("Grade !(Subset " + (i+1) +
				")");
			subset.calcAccuracyGainLoss(gradeBaseline, null);
			WekaHelper.printLine(_sbSubsetEval, subset.toString());
		}
	}

	FileIOHelper.saveToFile(_sbSubsetEval.toString(),
			Const.SUBSETS_EVAL_RESULTS_FILENAME, false);
}

private static String findAttributeIndices(
		Instances dataInstances, String[] attrNameLst) {

	StringBuilder sb = new StringBuilder();
	for (int attrIndex = 0; attrIndex < dataInstances.numAttributes();
			attrIndex++) {

		for (String attrName : attrNameLst) {
			if(dataInstances.attribute(attrIndex).name().
				equals(attrName)) {
				sb.append(attrIndex + 1).append(",");
			}
		}
	}
	sb.deleteCharAt(sb.lastIndexOf(","));
	return sb.toString();
}
```

```
	private static Instances removeUnwantedAttr(
			Instances dataInstances, String removeLst) {

		// Filter data to get rid of unwanted columns
		Remove removeAttrFilter = new Remove();
		removeAttrFilter.setAttributeIndices(removeLst);
		Instances filteredData = null;
		try {
			removeAttrFilter.setInputFormat(dataInstances);
			filteredData = Filter.useFilter(dataInstances,
					removeAttrFilter);
		} catch (Exception e) {
			e.printStackTrace();
		}

		return filteredData;
	}

	private static Instances keepSpecifiedAttr(
			Instances dataInstances, String retainLst) {

		// Filter data to get rid of unwanted columns
		Remove keepAttrFilter = new Remove();
		keepAttrFilter.setAttributeIndices(retainLst);
		keepAttrFilter.setInvertSelection(true);
		Instances filteredData = null;
		try {
			keepAttrFilter.setInputFormat(dataInstances);
			filteredData = Filter.useFilter(dataInstances,
					keepAttrFilter);
		} catch (Exception e) {
			e.printStackTrace();
		}

		return filteredData;
	}
}
```