Using Machine Learning to Predict Student Performance

1. Introduction

Having an educated population will help a country grow in productivity and life quality in the long run. While the Portuguese education system has improved from what it once was, the country's educational ranking is still one of the lowest in all the European nations. The early-school dropout rate among the Portuguese, at 40%, is significantly higher than the average of the European Union, at 15%. Most of the class data were centered between the student's demographics, their social and school statistics, and their performance scores, revolving around two core classes, Mathematics and the Portuguese language (their native language). Revolving the study around these two classes is important because these two classes provide students the fundamental knowledge needed to understand other subjects that are taught in school. Using automated methods to analyze such data and extract information from the data set can help the decision-maker to view important data that may be overlooked by humans. Automated methods would also help identify trends and patterns in data that will help the decision making as well as optimizing the method for a higher rate of success in classifying or predicting a feature.

Because of these automated methods, many more questions can be asked and answered: Which students are more likely to take on more classes? Which students will drop out of school? What courses are students excelling at and which courses are they performing poorly in? What factors of a student's life affect their performance the most? This study will focus on attempting to use factors that are deemed important in determining a student's performance in school. The results of predicting student performance will help decision-makers to distribute school resources. It could also help decision-makers implement new policies that help weaker students in their studies (e.g. remedial classes, tutors).

In this paper, we analyze real-world data, sourced from school reports and questionnaires. The data was taken from two sources due to the scarcity of data in the former - providing only grades and number of absences. The questionnaire provided more detailed information that included demographics (e.g. age, sex) and social and school attributes (e.g. alcohol consumption, mother's education, father's career). These two sources complemented each other and provided a profile about the student and their school life. The purpose of the paper is to use these profiles to predict student performance in school, using a binary classification (the student will either pass or fail).

Using this target binary classification, we compared five different machine learning algorithms (i.e. Decision Tree, Random Forest, kNN, Support Vector Machine, and Logistic Regression) to determine which model performed the best in predicting a student's performance in school.

2. Related Works

Cortez and Silva (2008) performed a similar study using Business Intelligence (BI)/Data Mining (DM) techniques to predict students' secondary school performance using four DM models (i.e. Decision Trees, Random Forest, Neural Networks, and Support Vector Machines). Their paper concluded that a student's future performance relies heavily on their previous achievements, and other features could impact a student's performance as well.

Ma et al. (2000) applied a DM approach in Singapore, selecting the weaker tertiary schools for remedial classes. Their data set included demographic attributes (e.g. sex, location) and a record of Singopoeran students from various tertiary schools' performance over the years. The solution proposed by the model outperformed the traditional method of assigning remedial classes.

In 2003 (Minaei-Bidgoli et al. 2003), online student grades from the Michigan State University were modeled using three classification approaches (i.e. binary: pass/fail; 3-level: low, middle, high; and 9-level: from 1 - lowest grade to 9 - highest score). The dataset used in this study had 227 individuals with online features (e.g. number of answers correct, how many attempts for an assignment). They attempted to predict how a student would perform on an assignment and the best results were obtained by a classifier ensemble (e.g. Decision Tree and Neural Network) with accuracy rates of 94% (binary), 72% (3-classes), and 62% (9- classes).

Kotsiantis et al. (2004) applied several more DM algorithms to predict the performance of computer science students participating in a long-distance university learning program. The data used here was that each student's demographic information (e.g. sex, age) and their performance in the class (e.g. grades, number of correct answers, pass or fail) were collected and recorded. The best performing method was the Naive Bayes method with an accuracy of 74%. In this study, similar to Cortez and Silva (2008), it was also found that past scholarly achievement had more impact in predicting future performance rather than demographic attributes.

Pardos et al. (2006) collected data from an online tutoring system regarding 8th grade Math tests in the United States. The authors used regression to predict the math test's score based on the student's skill. It was found that

Bayesian Networks performed the best and the results were a predictive error of 15%.

3. Methodology

Table 1: Pre-processed data of student features

Feature	Description
sex	student's sex (binary: female or male)
age	student's age (numeric: from 15 to 22)
school	student's school (binary: *Gabriel Pereira* or *Mousinho da Silveira*)
address	student's home address type (binary: urban or rural)
Pstatus	parent's cohabitation status (binary: living together or apart)
Medu	mother's education (numeric: from 0 to 4 *a*)
Mjob	mother's job (nominal *b*)
Fedu	father's education (numeric: from 0 to 4 *a*)
Fjob	father's job (nominal *b*)
guardian	student's guardian (nominal: mother, father or other)
famsize	family size (binary: ≤ 3 or > 3)
famrel	quality of family relationships (numeric: from 1 – very bad to 5 – excellent)
reason	reason to choose this school (nominal: close to home, school reputation, course preference or other)
traveltime	home to school travel time (numeric: 1 – < 15 min., 2 – 15 to 30 min., 3 – 30 min. to 1 hour or 4 – > 1 hour).
studytime	weekly study time (numeric: 1 – < 2 hours, 2 – 2 to 5 hours, 3 – 5 to 10 hours or 4 – > 10 hours)
failures	number of past class failures (numeric: n if $1 \leq n < 3$, else 4)
schoolsup	extra educational school support (binary: yes or no)
famsup	family educational support (binary: yes or no)
activities	extra-curricular activities (binary: yes or no)
paidclass	extra paid classes (binary: yes or no)
internet	Internet access at home (binary: yes or no)
nursery	attended nursery school (binary: yes or no)
higher	wants to take higher education (binary: yes or no)
romantic	with a romantic relationship (binary: yes or no)
freetime	free time after school (numeric: from 1 – very low to 5 – very high)

goout	going out with friends (numeric: from 1 – very low to 5 – very high)
Walc	weekend alcohol consumption (numeric: from 1 – very low to 5 – very high)
Dalc	workday alcohol consumption (numeric: from 1 – very low to 5 – very high)
health	current health status (numeric: from 1 – very bad to 5 – very good)
absences	number of school absences (numeric: from 0 to 93)
G1	first period grade (numeric: from 0 to 20)
G2	second period grade (numeric: from 0 to 20)
G3	final grade (numeric: from 0 to 20)

a 0 - none, 1 - primary education (4th grade), 2 - 5th to 9th grade, 3 - secondary education, or 4 - higher education.

b teacher, health care related, civil services (e.g. administrative or police), at home, or other.

3.1 Data Processing

The merged dataset contains 395 student profiles with 33 features, 16 of them being non-numerical data. There are no missing values within this dataset. We binarized all the features that contained two values by assigning the numeric values of 0 and 1 for easy processing. For example, male was replaced with 1 and female with 0 in the 'sex' column meanwhile all the yes and no values in different features were replaced by 1 and 0 respectively. We also converted the nominal values in 'Mjob', 'Fjob', 'reason', and 'guardian' by one-hot encoding it using panda's get_dummies method. We used the last three columns to create a scores column, which is the target variable. These last three columns - G1, G2, and G3 - represent the student's first-period score, second-period score, and final score, respectively. We decided to compile these scores together by averaging the three scores and assigning those values according to the letter grades mentioned in the paper. We ended up mapping these ordinal values into integers. There was class imbalance within the scores column with the majority assigned to 0 value with 178 rows. Hence, we upsampled the minority classes by using the SMOTE function to fix the class imbalance problem. For each model, we had to specify the random_state parameter to ensure that our results were consistent for the same code from run to run.

3.2 Exploratory Data Analysis (EDA)

We created a correlation heatmap of features that passed the threshold of 0.04 in Random Forest Feature Selection to see the correlations, which is feasible than creating it with all the features. The matrix shows that 'Fedu' and 'Medu' have a high positive correlation as well as 'Walc' and 'goout'.

3.3 Hyperparameter Selection and Feature Extraction

After processing the data into a format we can use, the next step was to select the appropriate hyperparameters for each model using a 10-fold cross-validation grid search. Before performing the grid search, we created 4 different feature extracted datasets to run grid searches for. As the control, for one dataset we did not include any feature extraction. The other feature extraction techniques we used were: Linear Discriminant Analysis (LDA), Principal Component Analysis (PCA), and Kernel Principal Component Analysis (KPCA). For each of the models we tried different test set sizes ranging from 20-30% of the overall dataset. We did not find any noticeable performance changes, so we opted for a test size of 30% (270 total examples) to protect against generalization error.

3.3.1 PCA and KPCA

For the Principal Component Analysis (PCA), it is recommended that one-hot encoded binary features should not be used in this algorithm. Due to this, we used only the continuous features in the PCA. We tested two

separate classifiers, one being regular PCA and the other being KPCA (Kernel). The result we got from them differed drastically because it appears that our dataset is not linear. For PCA, our output consisted of a graph displaying the separation of the classes. In this graph, it is evident that the classes are laid on top of one another, demonstrating that these classes are not linearly separable. For KPCA, we got more promising results because KPCA excels on non-linear dimensionality reduction. The graph produced shows the classes are well separated with the use of kernels, revealing that we have a suitable dataset for non-linear classifiers. Thus, the use of both methods of PCA has given us a lead on what classifier might perform best on the given dataset because it is not linear. So, we are inclined to believe that classifiers such as Support Vector Machine (SVM) might work best for our model of the dataset although this will not stop us from exploring many potential models.

3.3.2 LDA

The goal of performing Linear Discriminant Analysis (LDA) is to both increase the computational efficiency and reduce the chances of overfitting the model to the testing set. Additionally, LDA assumes that the data is normally distributed, so we decided to scale the data using the Normalizer from the sklearn API before performing LDA on the dataset. For this LDA, we chose to do it with the first two components so that it could be visualized, even though we could hypothetically use up to 3 (the sklearn API says that n_components should be strictly less than the number of classes - 1, which for us would be 4).

We performed each of these feature extraction methods for every model, and then ran a separate grid search for each feature extraction method as well as one without feature extraction where we only used a standard scaler on the data. Each grid search outputted the list of hyperparameters that led to the best accuracy score on the validation set and the value of the highest accuracy achieved. For each model, we chose the feature extraction (or no feature extraction) technique that yielded the best accuracy without giving us parameters that would lead to overfitting.

3.4 Feature Selection

After determining the best hyperparameters and feature extraction method, we used feature selection to find out which features should be used for our final model. The two feature selection techniques we used were Random Forest Feature Selection (RFFS) and Sequential Backwards Search (SBS).

3.4.1 RFFS

We used a RandomForest feature selector to determine which features were the most important towards determining the value of the target feature. RFFS is a supervised technique that requires knowledge of the target feature in order to determine the importance of each feature. For RFFS, we create a Random Forest classifier and fit it to our data. Once that was complete, we used our classifier to select the features in order of importance. Figure 1 shows a bar chart that shows the importance of some of the most prominent features.

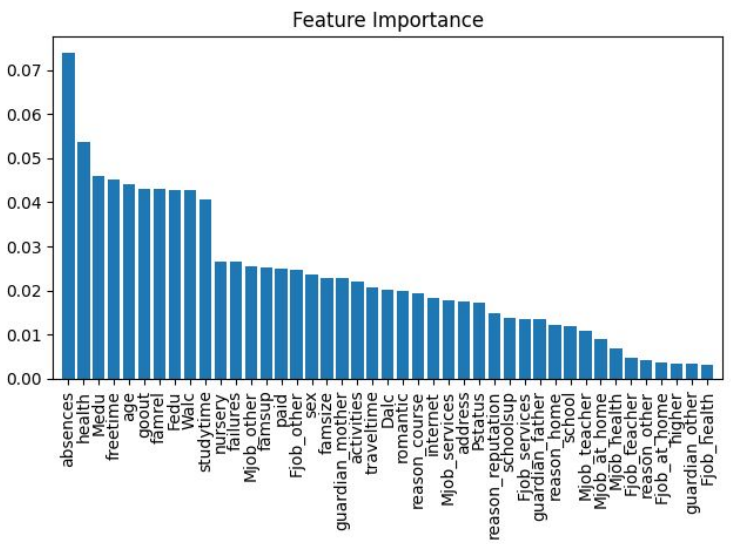


Figure 1. Feature Importance

3.4.2 SBS

The next feature selection technique we used was SBS. SBS is a greedy algorithm that is particularly effective at improving the performance of overfit models. For SBS, we create a classifier (we used an SVM) and specify a performance metric (for our purposes, we used accuracy). Then, we run the classifier all the features and go down to less and fewer features, measuring the performance loss caused by each feature along the way. This gives us a list of features sorted by their importance. Figure 2 shows how the accuracy of the classifier varies with the number of features used in the model:

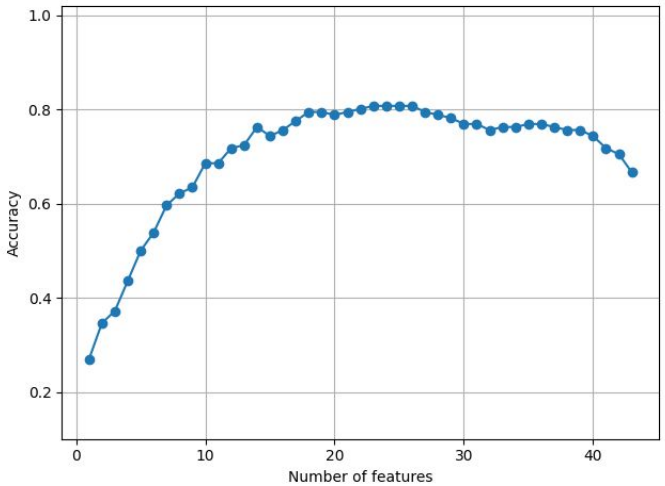


Figure 2. SBS

From these two feature selection techniques, we were able to create hard-coded lists of features sorted by their importance assigned by each respective feature selection method. Then, we took our model with the optimal hyperparameters and tracked the accuracy, precision, recall, and F1 score for each number of features.

3.5 Evaluation Metrics

As mentioned in the previous section, the four evaluation metrics we measured were accuracy, precision, recall, and F1 score. We chose to measure all of these metrics so we had a more full understanding of the performance of our models. For example, if we only measured accuracy and saw a 90% accuracy, we would think this is good, but if the precision was only 20% that would render our model essentially useless. Because of this, we needed to track several important metrics. Additionally, precision, recall, and F1 score were calculated using a weighted average, meaning that for each metric, the metric was calculated for each target class, and then the final output was the weighted average of all of those values. This parameter was required to be specified because we were not attempting to do binary classification.

3.6 Final Model Selection

After going through feature selection, we selected the number of features that yielded the best overall results returned by our performance metrics. We then created a final model and outputted all of its metrics. We also created a confusion matrix for each final model that allowed us to see the breakdown of how good our classifiers were at correctly classifying each letter grade. Finally, we measured how long it took to run the entire training, validation, and testing process. These graphics can be seen in the next section.

4. Evaluation

4.1 Models

We tested 6 different classifiers on the given dataset, Logistic Regression, Naive Bayes, Decision Tree, Random Forest, Support Vector Machine, and K-Nearest Neighbor. The first 3 performing poorly and the last 3 performing well.

4.1.1 Logistic Regression

We tested Logistic Regression through grid search and we found that we could not achieve accuracy over 0.54. In the end, we did not consider this model for predicting student grades. According to the Random Forest feature selection, the model performed the best with 40 features, however, SBS determined the model performed best with 33. In the end, we decided to instantiate the model with the parameters of 'l2', C=0.001, and 'newton-cg' for the solver. We found these parameters to be the most optimal using a grid search algorithm. Even after finding the optimal parameters and

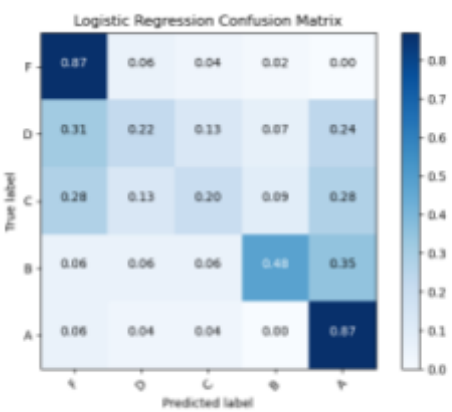


Figure 3. Logistic Regression Confusion Matrix

tuning the hyperparameters of the model, it results in the confusion matrix found to the left. According to this figure, the logistic regression model can predict an 'A' and 'F' very well, but the grades in between were not classified correctly. Because of this disparity, this model was ruled out from consideration because the accuracy and precision of the model will not be high. This may be because logistic regression classified between a passing grade and a failing grade. The total run time for logistic regression was 8 minutes and 7 seconds.

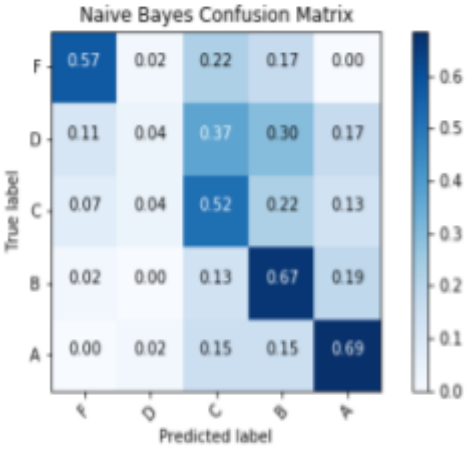


Figure 4. Naive Bayes Confusion Matrix

4.1.2 Naive Bayes

We tested Naive Bayes thoroughly but were not able to achieve accuracy above 0.52, thus we decided not to pursue this classifier. The model performed best at 33 features, even achieving a precision of 0.6, but struggled to push past that. We used LDA for feature extraction, which seemingly performed best, and Random Forest feature selection to create the best performing model in terms of metrics. We also employed the GaussianNB classifier, this was because the other viable classifiers, MultinomialNB and ComplementNB, could not process the negative numbers outputted by the feature selection. Additionally, the no feature extraction method performed poorly in all cases, which was why GaussianNB was selected in the end to be used for this dataset. The provided confusion matrix (Figure 4) illustrates this model's poor performance after all our preprocessing, therefore this type of classifier is not our best option for this dataset. This is likely a result of Naive Bayes being a simpler model that assumes that all of its features are independent, which they are not in the case of this dataset.

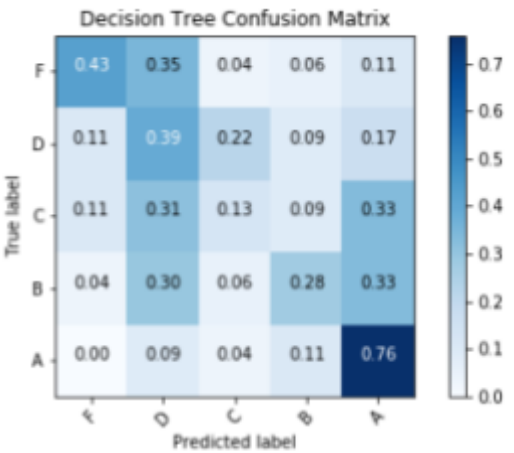


Figure 5. Decision Tree Confusion Matrix

4.1.3 Decision Tree

We tested Decision Tree thoroughly but were not able to achieve accuracy above 0.51, thus we decided not to pursue this classifier. The model performed best at 42 features, even achieving a precision of 0.62, but struggled to push past that. We used LDA for feature extraction, which seemingly performed best, and Random Forest feature selection to create the best performing model in terms of metrics. Additionally, we used the DecisionTreeClassifier for efficiency and convenience purposes and used a grid search on this classifier to produce the most ideal hyperparameters for the model. The resulting hyperparameters resulted in a depth of 4, a criterion of entropy, and a random_state of 0. The provided confusion matrix (Figure 5) illustrates this model's poor performance after all our preprocessing, therefore this type of classifier is clearly not our best option for this dataset.

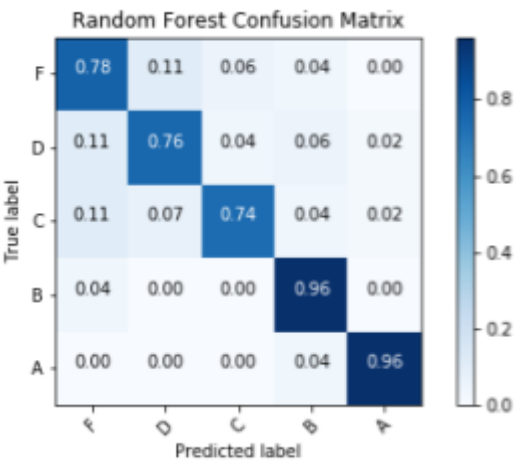


Figure 6a. Random Forest Confusion Matrix

4.1.4 Random Forest

For Random Forest, we decided to do a grid search on a RandomForestClassifier to determine the best hyperparameters for a model on this dataset. This was an extensive process because of a significant runtime, so multiple grid searches were needed to find these values. In the end, we produced n_estimators = 150, criterion = gini, max_depth = 17, and a random_state = 121 for our best

hyperparameters for this model. Next, we concluded that no feature extraction was necessary for this model, as this option performed significantly better than any feature extraction method during testing. Thus, we had to test other scalers other than just the standard scaler, so we tested MinMax, MaxAbs, and Normalizer, all of which performed worse than the standard scaler, so we stuck with the standard scaler for this model. Additionally, we noted that SBS feature selection was outperformed by Random Forest feature selection, thus we decided to use the latter for our model (images that detail this are included above). For RFFS, the best case was achieved when using all 43 features, producing a score of roughly 0.85 for accuracy, precision, recall and F1 score. This is coupled with a CV validation set accuracy of 0.758 (+- 0.023) which, although not perfect, shows us that our

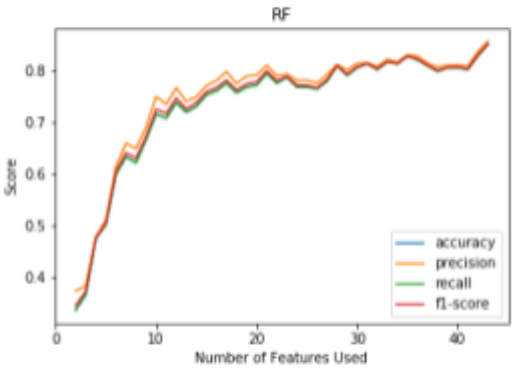


Figure 6c. RF of Random Forest

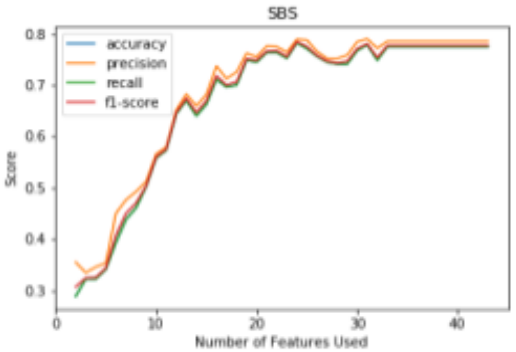


Figure 6b. SBS of Random Forest

model is not too overfit on this dataset. It is somewhat generalizable as the score is near our actual model's accuracy, but it is not ideal. Also, the provided Confusion Matrix (Figure 6a) illustrates that our model is efficient at predicting As and Bs (both having 96% chance of doing so), but struggles to consistently predict Cs, Ds and Fs (74%, 76%, 78% respectively chances of doing so). However, this is still a solid prediction of all 5 cases, and it is worth noting that this outperforms our other models by a significant margin. The random forest model was by far the most expensive in terms of run time, and since it had to be run in parts, we could not determine exactly how long the total run time was.

4.1.5 Support Vector Machine (SVM)

After the grid search cross validation, we determined that our SVM should have a C value of 10, a radial basis function (rbf) kernel, and a gamma value of 0.1. The overall cross validation accuracy was ~77% with a 2% margin of error.. These hyperparameters show us that the model is generalizable, but it is not over-generalized. The C value is high enough that the hyperplane is tight and can capture the complexity of the data, but it is not too tight to overfit our model to the training set. The gamma value of 0.1 is low enough that we are learning what we need to learn from each training example, but not too high that we are learning too much from each example and thus overfitting the model. The grid search also showed us that doing no feature extraction would yield the best results. Because of this, it was necessary to test several other scalers other than just the standard scaler, so we tested MinMax, MaxAbs, and Normalizer, all of which performed worse than the standard scaler, so we stuck with the standard scaler for this model. After performing feature selection, RFFS with 18 features lead to the best results with every metric at approximately 78%. The feature selection graphs (Figures 7b and 7c) tracking the various performance metrics can be seen on the right. This accuracy shows us that our model is not overfit because it does not perform significantly worse than it does on the validation set. The confusion matrix in Figure 7a shows

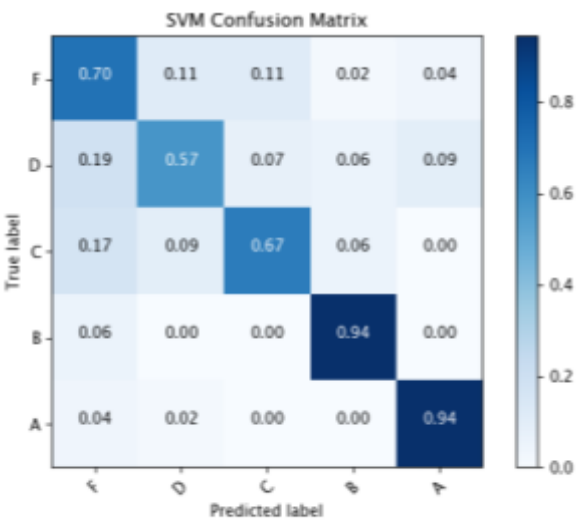


Figure 7a. SVM Confusion Matrix

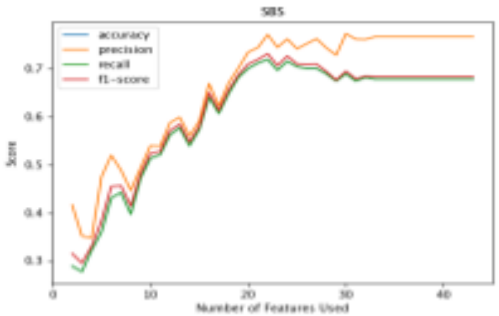


Figure 7b. SBS of SVM

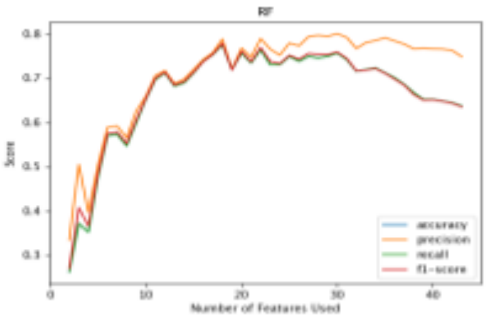
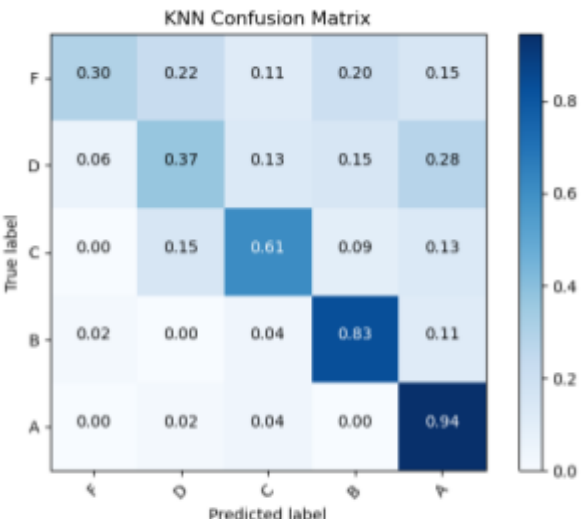


Figure 7c. RF of SVM

how good the SVM was at classifying each grade. It predicts A and B students very well, with accuracies of 94%, but that percentage goes down with B, C, and D students. The SVM could be useful in classifying A and B students from those with worse grades, but for our purposes, it does not have consistent performance across all 5 classes. The total run time for the SVM model was 9 minutes 23 seconds.

4.1.6 K-Nearest Neighbors

We were able to achieve a cross-validation accuracy of 0.767 with a margin of error of 0.02. The model performed the best with 36 features. However, after performing a grid search of the model, we achieved the optimal parameters, using 'ball_tree, 10 leaves, 'manhattan', 17 neighbors, and 'distance'. Using the Random Forest feature selection, we achieved a final accuracy of 0.61 with a final precision of 0.72. Because the model had 42 features that could have been used to predict the target feature, considering 17 neighbors would not be too complex. According to RF, the kNN model performed the best with 36 features with all the scores (i.e. accuracy, precision, recall, F1 score) higher than 0.58. Figure 8a shows that although the model is precise and accurate with predicting scores, the accuracy drops



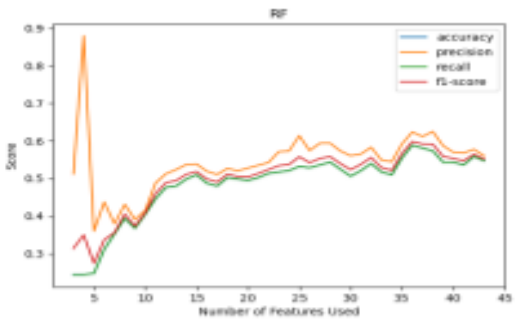


Figure 8c. RF of KNN

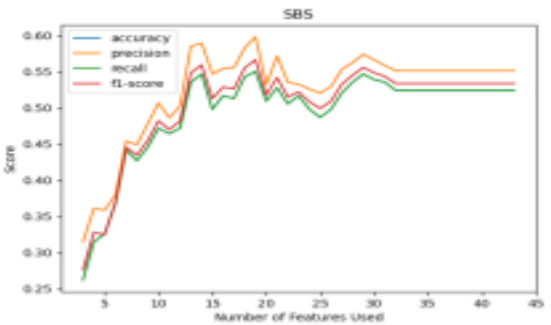


Figure 8b. SBS of SVM

off when the student is predicted to perform poorly. The model can predict passing grades extremely well, however scores below 'C' have less than a 50% chance to be predicted correctly. Although kNN performed better than some of the other models due to this imbalance of classification, the model cannot be considered. The total run time for kNN was 14 minutes and 3 seconds.

5. Threats to Validity

Although remedying the class imbalance was important for getting representative data to train and test on, it also causes a possible issue with the generation of synthetic data. To make it so the number of training examples of each class are the same, the SMOTE API call had to create synthetic data. In doing so, we are creating data that was not pulled naturally from the population we are studying. Hence, there is the potential for training our model on inaccurate, non representative data which will make it harder for the model to classify new unseen data. Additionally, because we are specifying a random state, that means we have the potential to generate infinite possibilities of synthetic data, which could cause better or worse performance, entirely dependent on which random state we use. Therefore, the random state we selected could be causing our model to either overperform or underperform based on this synthetic data.

Another topic that must be addressed when considering our models is the potential for generalization error. While we included validation sets to make sure we avoided this, they still did not perform as well as our model's sets, thus leaving some, but not much, trouble in generalizing our best models. Additionally, our data was sourced from only two schools in Portugal, leaving us to question whether or not these models are applicable towards other regions in the world. Regardless, to the best of our ability we constructed models with generalization error and overfitting in mind and feel as though our models can be generalized.

6. Conclusion/Future Work

Our paper presents multiple approaches to classifying this given dataset, these being six different models. Three of these models, Logistic Regression, Decision Tree, and Naive Bayes were unable to classify this data sufficiently and thus are not worth considering in our final solution. On the other hand, SVM, Random Forest, and KNN are all worth considering in contention for the best model on this data. Acknowledging that Random Forest has the best accuracy by about 5-10 percent more than SVM and KNN, we conclude that the best classifier for our data is Random Forest. Additionally, because it is an ensemble learning method, it is less likely than the other single classifier to suffer from generalization error. Although it is the most computationally expensive algorithm, Random Forest can still be run on a local machine and produce results on the scale of hours (assuming the grid search is used to full capacity) instead of days as a deep learning approach might. Since minimizing runtime is not one of our goals in this paper, we can discount this shortcoming. Something else to consider is that the confusion matrices for almost all of the models show that it is easier to classify As and Bs than it is to classify Cs, Ds, and Fs. If this model were to be used more heavily in education research, it should be noted that the model is significantly better at classifying higher performing students.

For future work, one could take a similar approach to us, except without using the combined dataset. By separating the Portugese and Math grades into different datasets, there is potential for creating a classifier that performs better than the ones we created. Since our datasets were combined, there might have been more variance and less consistency in the data.

References

[1] Cortez, P., & Silva, A. (2008, January). Using Data Mining to Predict Secondary School Student Performance. Retrieved November 21, 2020, from http://www3.dsi.uminho.pt/pcortez/student.pdf

[2] Ma Y.; Liu B.; Wong C.; Yu P.; and Lee S., 2000. Targeting the right students using data mining. In Proc. of 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Boston, USA, 457–464.

[3] Minaei-Bidgoli B.; Kashy D.; Kortemeyer G.; and Punch W., 2003. Predicting student performance: an application of data mining methods with an educational web-based system. In Proc. of IEEE Frontiers in Education. Colorado, USA, 13–18.

[4] Kotsiantis S.; Pierrakeas C.; and Pintelas P., 2004. Predicting Students' Performance in Distance Learning Using Machine Learning Techniques. Applied Artificial Intelligence (AAI), 18, no. 5, 411–426.

[5] Pardos Z.; Heffernan N.; Anderson B.; and Heffernan C., 2006. Using Fine-Grained Skill Models to Fit Student Performance with Bayesian Networks. In Proc. of 8th Int. Conf. on Intelligent Tutoring Systems. Taiwan.