1 Classification vs Regression

- This project is a supervised machine learning problem of classification. Rather than finding the continuous answers, we are looking for discrete answers. Specifically, we want our classifier to predict between two types of students: those who pass and those who fail.

2 Exploring the Data

- Total number of students: 395
- Number of students who passed: 265
- Number of students who failed: 130
- Graduation rate of the class: 67.09%
- Number of features: 30

3 Preparing the Data

- (IPython Notebook)

4 Training and Evaluating Models

Support Vector Machine (Classification)

- The general application is by finding well-fit line that divides the data with the largest margin. Its strength can rule out outliers. It's an effective classifier with projects with high dimensions, such as this one, with more than 30 dimensions. Its weakness is that it is sensitive to overfitting if the number of features in a dataset is high or if the dataset is very large. These can cause a lot of noise for the SVM.
- I chose this model to work with the high number of dimensions of the student intervention project. Moreover, preprocessing rows adds on to the dimensions.
- Training Sizes 100 200 300
- Training Time (secs) 0.001269 0.003408 0.006559
- Prediction Time (secs) 0.001530 0.003358 0.010491
- F1 score for training set 0.884058 0.883721 0.876106
- F1 score for test set 0.761905 0.800000 0.800000

Multinomial Naive Bayes

- The general application is finding the probability that the students will fail or not, assuming that all the features are independent of each other. MultinomialNB() is the most appropriate of the three Naive-Bayes classifiers. The feature columns are all discrete, and they are neither binary nor continuous. Its strengths are that it requires a small amount of training data and that it can be fast, and its weakness is that it provides as a bad estimator.
- I chose this because it fits well for preprocessed and discrete columns of this project, as opposed to the other Naive Bayes classifiers.
- Training Sizes 100 200 300
- Training Time (secs) 0.001110 0.001232 0.001464
- Prediction Time (secs) 0.000800 0.001092 0.001107
- F1 score for training set 0.828125 0.810997 0.793503
- F1 score for test set 0.666667 0.751880 0.794118

K Neighbors Classifier

- The general application of K Neighbors Classifier is to call the closest neighbors. Its strength is that it has no training time, and it becomes more effective with large training data. Its weakness is that increasing k makes the classification boundaries less distinct.

- I chose this model by hypothesizing that a point's closest neighbors would more accurately predict that point's outcome.
- Training Sizes 100 200 300
- Training Time (secs) 0.000615 0.000718 0.000889
- Prediction Time (secs) 0.002058 0.004329 0.007720
- F1 score for training set 0.820896 0.823529 0.844749
- F1 score for test set 0.753846 0.776978 0.808219

5 Choosing the Best Model

Of all the models run, the KNeighborsClassifier() classifier performed the best, based on the combined performance of its test F1 score, time efficiency, available data, limited resources, and cost.

Both its training and testing F1 scores improved as the data size increased. The trend indicates that the model would improve if given more data. Given all the data, its test F1 score performed the best of the three. Of the three classifiers, it had the quickest training time; it also had the slowest prediction time, exponentially increasing with the size of the training dataset.

The SVC() classifier shows a decrease in training F1 score performance as it was fed more data. There was no improvement of F1 score when the training dataset increased from 200 to 300. Compared with KNeighborsClassifier(), SVC() didn't yield a better F1 score when given the full amount of training data of 300. MultinomialNB() had the fastest combination of training and prediction time. It also had the biggest increase in test F1 score, when increasing the data size from 100 to 200 to 300. Though, its F1 score, with a training dataset of 300, didn't perform as well as the other two.

The final model, KNeighborsClassifier(), maps out the data in hyperspace. When it makes a prediction on a student, it takes weighs in all of his features, such as his parents' occupations, whether or not he wants to go to college, etc., and it looks at all the other students who are most similar to him, based on these features. This model will take the students most similar to him and determine if he will pass or fail by looking at the other students' pass/fail rate. We tune this model by playing around with some parameters, such as the number of students who have influence in deciding if this target student will pass or fail.

The final model's F1 score is 0.8027.