1. Classification vs Regression

Your goal is to identify students who might need early intervention - which type of supervised machine learning problem is this, classification or regression? Why?

Answer: This is a classification problem because the result we are trying to predict is categorical, specifically binary in this case; will someone pass the final exam or not. Classification problems have outputs (y's) that are categorical as opposed to regression that has outputs that are continuous numerical values.

2. Exploring the Data

Can you find out the following facts about the dataset?

- Total number of students = 395
- Number of students who passed = 265
- Number of students who failed = 130
- Graduation rate of the class (%) = 67%
- Number of features (excluding the label/target column) = 30

Use the code block provided in the template to compute these values.

Answer: See iPython notebook code/output for answers to this question.

3. Preparing the Data

Execute the following steps to prepare the data for modeling, training and testing:

- Identify feature and target columns
- Preprocess feature columns
- Split data into training and test sets

Starter code snippets for these steps have been provided in the template.

Answer: See iPython notebook code/output for answers to this question.

4. Training and Evaluating Models

Choose 3 supervised learning models that are available in scikit-learn, and appropriate for this problem. For each model:

- What are the general applications of this model? What are its strengths and weaknesses?
- Given what you know about the data so far, why did you choose this model to apply?
- Fit this model to the training data, try to predict labels (for both training and test sets), and measure the F1 score. Repeat this process with different training set sizes (100, 200, 300), keeping test set constant.
- Produce a table showing training time, prediction time, F1 score on training set and F1 score on test set, for each training set size.

Answers below:

DecisionTreeClassifier:

This model can be used as a classification algorithm. It generates a decision tree that classifies examples based on Boolean nodes. The benefit of a model like this, if it fit the data well, would be that the fitted model would be able to be printed off and is human readable. This human readability would allow the school to fit the model once, then avoid using the costly computer resources in the future to determine if students might need additional help and guidance. There are a couple of weaknesses of decision trees though. They are extremely sensitive to overfitting the data. While that can be managed through parameter tuning, it is hard to have high confidence that the final model will generalize optimally. Also, since each leaf is dependent upon all of the decision nodes above it in the tree, it assumes variable interactions. Variables in a data set are not necessarily interdependent. Decision trees also have problems predicting out-of-sample instances. If a feature set/outcome pair is not in the training data set, it is very hard for a decision tree to generalize well when prediction that type of outcome.

I chose this model because it was simple, and potentially human readable. If it fit the data well, that would be a good model to use according to Occam's razor.

Decision Tree (default parameters)

	Training Set			Test Set (constant 95 points)		
Training Set Size	Training Time (sec)	Prediction Time (sec)	F1 Score	Prediction Time (s)	F1 Score	Combined Time
100	0.002	0.001	1	0.001	0.79433	0.004
200	0.005	0.001	1	0.001	0.704	0.007
300	0.009	0.001	1	0.001	0.74074	0.011

SVC:

SVC stands for support vector classifier and it is used to classify data use a support vector algorithm framework. Support vector machines are a good option to explore for a couple of reasons: (1) they can fit very complex data if the correct kernel is selected; (2) they are relatively easy to tune (using grid search to change values for C and gamma) to minimize overfitting; (3) SVM tends to have relatively good out-of-sample generalization[i]; (4) SVM will give a unique solution, because the However, SVM has some drawbacks. Especially compared to a decision tree, SVM is not human readable, or interpretable. The school would have to just "trust the model" which might seem like a black box to them. The other difficulty with SVM is selecting the kernel. It is difficult beforehand to know what kernel will create the best data fit. In this case, I'm starting with just the default 'rbf' kernel to see if the model will work fairly well.

After looking at the results from the decision tree model, it looked like it trained pretty quickly, but didn't seem very accurate. So, I thought that SVM might be able to fit the data better based on its advantages described above. This would be a sacrifice in model interpretation, but at the end of the day the school really wants to know which students to work with.

SVM (default parameters)

	Training Set			Test Set (constant 95 points)		
Training Set Size	Training Time (sec)	Prediction Time (sec)	F1 Score	Prediction Time (s)	F1 Score	Combined Time
100	0.002	0	0.873418	0	0.82051	0.002
200	0.003	0.002	0.867314	0.001	0.82051	0.006
300	0.008	0.005	0.859649	0.001	0.81818	0.014

AdaBoostClassifier:

Boosting is an algorithm that uses the weighted average of many weak classifiers (in the case of AdaBoost, its linear hyperplanes) to predict the output class. One advantage of this algorithm is that its weighting is updated during each iteration to incentivize the correctly classify points in the next iteration that it got wrong in the previous iteration. In theory this helps the algorithm find a good fit to the data. Also, since it is an ensemble method (averaging outputs of other models) it has the property that is should generalize well. The real drawback for this approach is the time it takes to train. Essentially, a linear hyperplane decision boundary gets fit to the data during each iteration,

and then those hyperplanes are weighted averaged together. That's a lot of computation that the school might not want due to their concerns over cost and processing time.

While watching the Udacity training, Boosting was an algorithm I had heard of before but didn't understand yet. I was intrigued by the claim that it is resistant to overfitting, and thought that I would try it to see if it would have a better generalization error on the test set. I guessed that this model would have the best F1 score, even if it took a long time to train. That didn't end up being the case, at least not with the default parameters being used.

Boosting "Adaboost" (default parameters)

	Training Set			Test Set (constant 95 points)		
Training Set Size	Training Time (sec)	Prediction Time (sec)	F1 Score	Prediction Time (s)	F1 Score	Combined Time
100	0.061	0.006	1	0.007	0.8169	0.074
200	0.079	0.01	0.879377	0.008	0.69697	0.097
300	0.117	0.011	0.861244	0.007	0.77778	0.135

5. Choosing the Best Model

Based on the experiments you performed earlier, in 2-3 paragraphs explain to the board of supervisors what single model you choose as the best model. Which model has the best test F1 score and time efficiency? Which model is generally the most appropriate based on the available data, limited resources, cost, and performance? Please directly compare and contrast the numerical values recorded to make your case.

Answer:

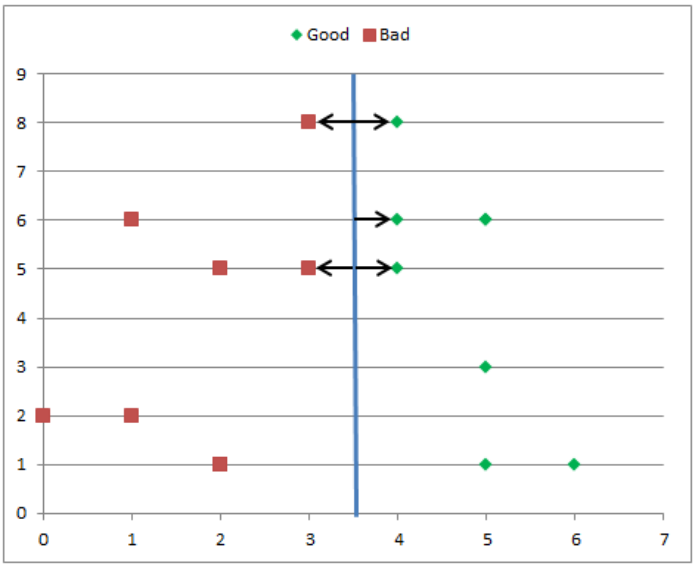
I tried decision trees, support vector machines (SVM) and the adaboost algorithm. Out of the 3 I chose the SVM model as the best model to refine and optimize. The biggest reason for this choice was the F1 scores obtained from the model compared to the other 2. The F1 score on the test set was consistently higher than 0.81 regardless of training set size. Decision trees gave fairly consistent F1 scores in the 0.70's (0.794, 0.704, 0.741) which seems to suggest that either the model parameters weren't tuned well enough or the model itself didn't fit the data as well. The AdaBoost model seems to have mixed F1 test results (0.817, 0.697, 0.778). This suggested to me that perhaps the model was getting lucky (or unlucky) based on the subset data points it was selecting from the 300 training points. So, I didn't expect this model to generalize well in the future with new data points.

On top of F1 performance, the processing time and data storage requirements needed to be considered for this project. As mentioned in the discussion of the AdaBoost algorithm above, it takes a lot of time to train and predict so it was noticeably worse that the other 2 options regardless of training set size. The SVM model and the Decision Tree model had quick training times (<0.009 for all options tried), and had even faster prediction times (<0.001 seconds). The prediction time is more important over the long run because this is what the school will be using regularly to make predictions.

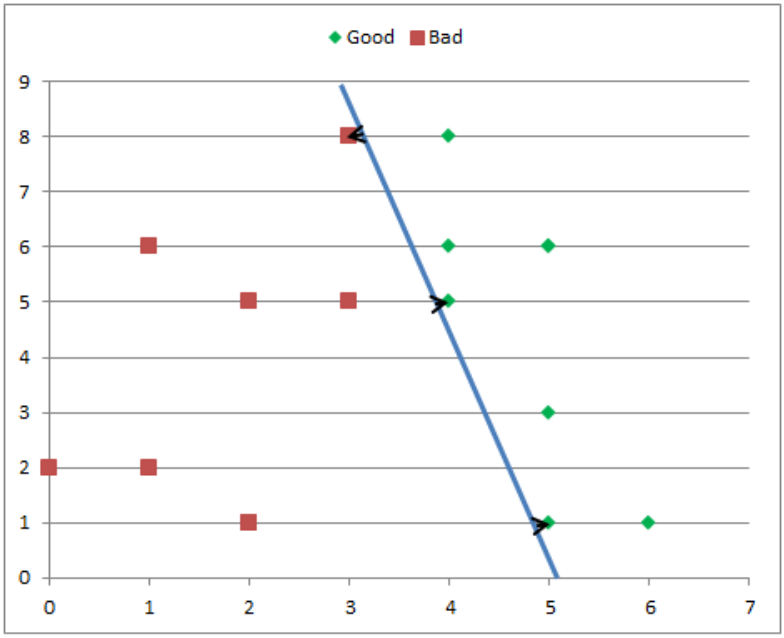
Since the end user wants to minimize the data storage and processing time while maximizing the F1 score, I thought the 100 or 200 point SVM model would likely work well for them.

In 1-3 paragraphs explain to the board of supervisors in layman's terms how the final model chosen is supposed to work (for example if you chose a decision tree or support vector machine, how does it learn to make a prediction).

Answer: For simple problems support vector machines look for a line that will best split the data into categories while keeping as much space a possible between data points in different categories. If you had a graph where all the points on the left were red and all the points on the right were green, it would create a (semi-)vertical line right in between the farthest right red point(s) and the farthest left green point(s). I say "right between" because SVM tries to maximize margin, which is another way of saying that it creates a dividing line with the most space between the line and the closest points of each category (see chart below for how this looks in a 2D graph).

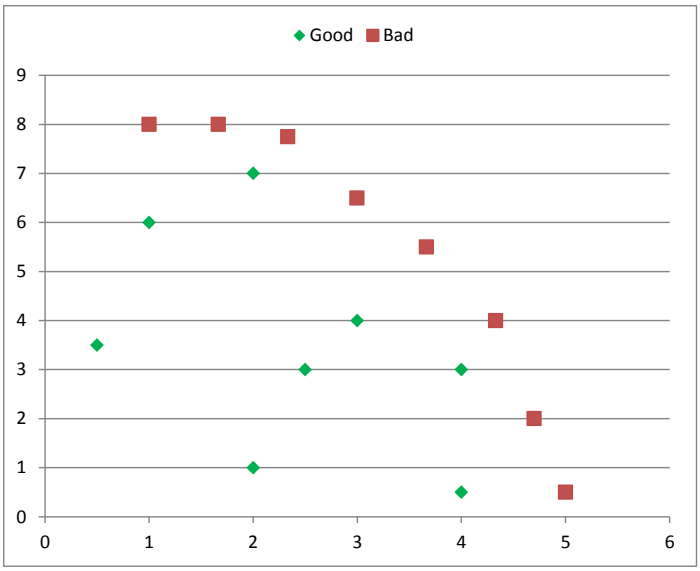


The way SVM finds this solution is by maximizing the distance between the blue decision boundary and the closest data points of each category. You can see below that if we had selected a different blue decision boundary, that the data would still be separated, but would NOT maximize this distance from the data points near the boundary.

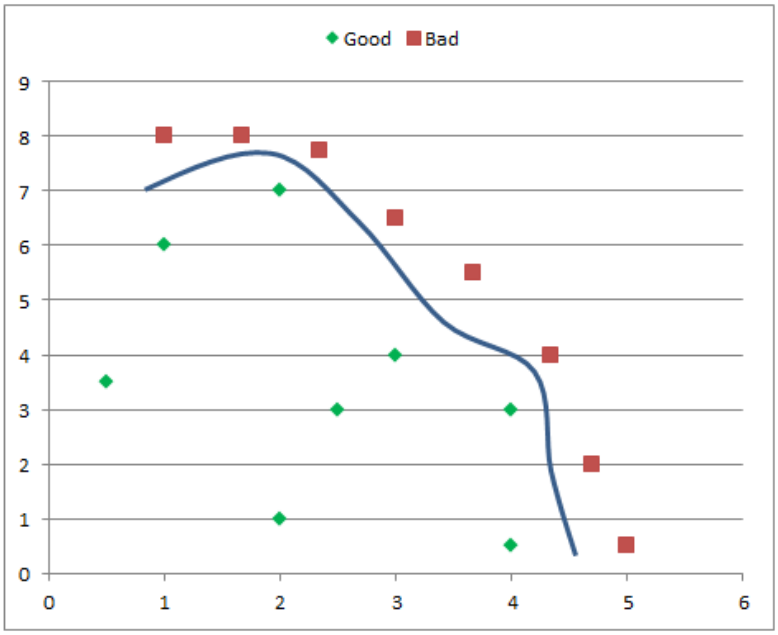


Once we have this blue decision boundary (essentially an equation for the line that splits the data) we can classify new data points by determining which side of the line that data point lies on. For example if we were give the data point (4.5, 3) we would know it was on the right side of the vertical line and classify it as green.

When data gets more complex solving this problem gets more difficult. Let's use the chart below as an example



It is impossible to draw a straight line that will separate the “good” and “bad” points in this chart. So support vector machines uses what is called the “kernel trick”. This kernel essentially adds another layer to the data. Think about this data being in 3 dimensions instead of 2 where the green points are closer to the reader (coming out of the page) and the red points are farther away (behind the page). If that’s true, then we can split the data using the plane of the page. It gets hard to visualize this if we have more than to variables we’re using to predict the output but the effect is still the same. If a final answer were put on our example chart, it would look similar to this hand-drawn curve below. We would predict new examples similar to the linear example. A point at (4,6) is above and to the right of the blue decision boundary so it would be classified as “Bad”.



Fine-tune the model. Use gridsearch with at least one important parameter tuned and with at least 3 settings. Use the entire training set for this.

What is the model's final F1 score?

Answer:

I fine-tuned the model using grid search for the kernel, C and gamma parameters. Here's the parameter space I explored to get my final model parameters and scores:

- Kernel = ('linear', 'rbf', 'sigmoid')
- C = (.001, .01, 0.1, 1.0, 10.0, 1000.0)
- Gamma = (.001, .01, 0.1, 1.0, 10.0, 1000.0)

When the grid search was complete, its cross-validated F1 score was 0.8013766... and the best fit parameters were kernel = 'rbf', C = 10.0, and gamma = 0.001.

SVM (kernel = 'rbf', C = 10.0, gamma = 0.001)

	Training Set			Test Set (constant 95 points)		
Training Set Size	Training Time (sec)	Prediction Time (sec)	F1 Score	Prediction Time (s)	F1 Score	Combined Time
100	0.001	0.001	0.853503	0.001	0.82051	0.003
200	**0.003**	**0.002**	**0.842444**	**0.001**	**0.8375**	**0.006**
300	0.02	0.009	0.822757	0.001	0.82051	0.03

After tuning the parameters, and looking at the performance metrics, it appears that the training set of 200 with the optimized parameters would best fit the school's needs. It doesn't take long (0.006 combined seconds), its F1 score is the highest on the test set, and it doesn't require the whole data set for information storage.

[i] http://www.diw-berlin.de/documents/publikationen/73/88369/dp811.pdf