Hu, Roy

Project 2: Student Intervention System

1.

This is a classification problem because the target variable that we're trying to predict takes on a discrete rather than a continuous value. Specifically, what we are interested in is a binary classification of the "passed" variable--whether a student will pass his or her high school exit exam or not, based on a vector of relevant features.

2.

Some basic summary statistics are as follows: our data set consists of 395 observations (students), of whom 265 passed the exam and 130 failed--a pass rate of 67%. There are 30 features including the target variable, with 1 target variable column.

3.

After extracting feature and target variables, and encoding factor variables, I split the data into a train and a test set for analysis.

4.

I start by using a straight-forward Support Vector Machine, which tries to maximize the margin between labeled data. The simplest case is when the data are linearly separable via a hyperplane, but the kernel can be customized to create a nonlinear divider in the vector space. The theoretical O(n) for time and space are between O(n^2) and O(n^3). Accordingly, SVMs work best when there is clear separation in the data, and less well when the data is noisy. Moreover, because training could potentially be in O(n^3) time, it might not be a good idea to use SVMs for large datasets; however, I choose to apply this model because for this smaller dataset using it should be fine.

Below is the the table which summarizes the SVM analysis:

	Training set size		
	100	200	300
Training time (secs)	0.002	0.013	0.008
Prediction time (secs)	0.002	0.003	0.006
F1 score for training set	0.859	0.869	0.865
F1 score for test set	0.783	0.776	0.775

Next, I try to do something a little more advanced in applying the AdaBoost algorithm. It was my favorite algorithm to learn about in the lessons. The idea behind AdaBoosting is that it uses a set of "weak learners" to build a strong one: during each iteration, examples that are hard to predict are given more weight on the subsequent iteration. In larger data sets, AdaBoost can be helpful because it’s not too slow, since it sequentially trains classifiers giving it a run time of something like O(N*T). Another advantage is that if you have a good weak learner, AdaBoost is not particularly prone to overfitting (though as always, it’s definitely possible). On the other hand, AdaBoost can be sensitive to noisy data. Thus, boosting might be better when prediction is hard, and I think for this smaller dataset it might not be the ideal algorithm, especially if there’s some noise or outliers. However, even if it doesn’t work here, I still want to keep it in the back of my mind, since for more complicated problems it does seem like ensemble/boosting methods work better by taking an average of predictions.

Below is the summary of the AdaBoost analysis:

	Training set size		
	100	200	300
Training time (secs)	0.148	0.119	0.134
Prediction time (secs)	0.007	0.005	0.005
F1 score for training set	0.953	0.883	0.865
F1 score for test set	0.720	0.805	0.779

Finally, I try the Bernoulli Naive Bayes. Despite their simplicity, Naive Bayes algorithms have show to work pretty well for things like email spam classification. One good reason to use Naive Bayes is that it works fast in O(N) linear time, unlike some of the other more expensive iterative approaches (although again, for this dataset I doubt it matters), especially saving time if the data set contains many features. It is easy to implement and generally works pretty well. Certainly, in one major downside is that Bernoulli Naive Bayes, much like any NB algorithm, depends on an assumption of independence, which means that features that are highly correlated with each other can cause the algorithm to fail. I think a lot of settings--potentially even this one--a lot of dependencies can exist among variables. For instance, in this data set, features like ‘studytime’ and ‘activities’ could be related.

Below is the summary of the Bernoulli Naive Bayes analysis:

	Training set size		
	100	200	300
Training time (secs)	0.003	0.002	0.002
Prediction time (secs)	0.000	0.001	0.005
F1 score for training set	0.838	0.817	0.865
F1 score for test set	0.761	0.759	0.771

5.

Dear Board of Supervisors,

The model that I am choosing to use in order to predict which students are at risk to fail their exit exam or not is the support vector machine (SVM). In simple terms, suppose we want to separate red dots from blue dots. There are of course a million different ways to do it, some ways better than others, but what we want to do is find the best way to separate the two types of dots.

Similarly, to build a student intervention system, one can picture lots of data points, each corresponding to a student. The data points are located in a spot based on the student’s characteristics: in this case, the two types we we want to know whether this student will pass the exam or fail. What the support vector machine does is find a way--the simplest case is a straight line, but it can certainly also be a squiggly line--to separate the data points for the students who pass from the data points for the students who don’t pass. As I mentioned earlier, there’s an infinite ways to draw a line to separate the two. However, intuitively, the best kind of separating line is that which creates the most space between those who pass and those who don’t pass. What my SVM will do is find that line.

Originally I was tempted to use a more complicated algorithm like AdaBoost, but it did quickly become clear that this dataset is small enough where there just won't be significant differences in runtime, even if some algorithms have much faster asymptotic O(n) time. I also am starting to see that the SVM is not as restrictive as it initially seems. As I wrote earlier, the goal of the SVM is to try to separate the data, with a linear hyperplane being the most simple case; however, the SVM can also maximize separation of a “soft” margin with other non-linear kernels as well as various tolerance levels for misclassification.

Based on the three models above, purely in terms of F1 score the model to use is AdaBoost, which has a score of 0.779 for the 300-sample. The big downside is that it also has the longest training time of 0.134 seconds. I think that for this small dataset, the difference is largely trivial. However, in a hypothetical case where there are many observations, using a Bernoulli Naive Bayes also definitely seems to make sense. Although the F1 score for the 300-sample is 0.771, which is the lowest among the three, because the algorithm runs blazing fast in linear time, the training time was only 0.002. In the end it seems like I am finding an accuracy vs. time tradeoff, though, again, given the smallness of the dataset, all three models are pretty comparable.

Finally, in order to truly find the best model, I tuned the parameters using grid search on each algorithm. The best model that I found was the SVM with rbf kernel, a C value of 1, and a gamma value of 0.1. The C parameter tells me how much I am willing to tolerate misclassification. A larger C penalizes misclassification more, leading to low bias but higher variance. Gamma can be thought of as “pushing” up points in higher dimensions to make it easier to find a separating hyperplane.

For my optimized SVM, I was able to achieve an F1 score of 0.827 on the test set, which I think is pretty good. The biggest potential downside that I can see is that the SVM is not the most efficient algorithm: under large data sets, it’s possible that using an SVM would be prohibitively expensive in terms of computational time cost. For a small dataset like this one, it’s hard to tell. But if I were analyzing a large data set, I would have to consider using a Bernoulli Naive Bayes, which had a decent F1 score of 0.814.