Classification vs Regression¶

In this project, as we want to predict whether a given student will pass or fail, given information about his life and habits, we treat it as a classification problem with two classes, pass and fail.

Exploring the Data

Total number of students: 395

Number of students who passed: 265

Number of students who failed: 130

Number of features: 30

Graduation rate of the class: 67.09%

Training and Evaluating Models

Model Consideration:

Before we manipulated it, the data was strongly categorical. Even age, though ordered, basically could be thought of as buckets between 15 and 22. So initially I thought that maybe the DecisionTree would be the most apt to deal with this problem. Even though we saw in the lectures that decision trees handle continuous data as well, the motivation for the algorithm presented in class, made me think that it would handle this "categorical" data well (even though we made it numeric). In class we had a boolean function with several weather related categories as input, and an output of play/no play tenis. Here we have data that is closely categorical, with a pass/no pass output. To get my bearings and not to impose any pre-judgement, I wanted to try most of the classifiers seen in class out of the box. I excluded neural net, since its recommended use requires the features to be scaled (one of the disadvantages of that algorithm). The result of this can be seen in the table below.

	DecisionTree	SVC	GaussianNB	AdaBoost	Kneighbors
Training time	0.0035 s	0.0069 s	0.0009 s	0.1022 s	0.0007 s
F1 score training set	1	0.874614	0.799737	0.870058	0.889301
Prediction time	0.0002 s	0.0007 s	0.0003 s	0.0053 s	0.0011 s
F1 score test set	0.702098	0.807571	0.767603	0.76675	0.780717

Kfold cross validation can be used to get the maximum use of a small data set like the one here. After separating the data into a training set (size 300) and test set (size 95), I used Kfold cross-validation on the training set with 10 folds to have a basic handle on how the models were performing on average. It is clearer now how they differ in performance. The decision tree trails all others in its f1 test score, while scoring perfectly on the training set (which no other does) so it seems to be an over-fitter. This is a known possible problem with trees. The SVC tops all others with an average test f1 of 0.808. KNN doesn't trail far with 0.781. Naive Bayes and AdaBoost are pretty much tied with test f1's of 0.768 and 0.767 respectively. As to narrowing the field to 3 models, I will discount AdaBoostClassifier and GaussianNB. Though I did not find this explicitly stated in the documentation, from the above table, AdaBoost is clearly expensive to train as its average training time of 0.1022s is approximately 15 times slower than the next slowest time of 0.069s by SVC. Since computational cost is a constraint, other algorithms may better serve this problem. GaussianNB was one of the fastest to be trained and had a decent average f1 test score. Two of Naive Bayes' advantages that are relevant to this problem are, its capacity to train on small amounts of data and its training speed. In this sense it is perfectly suited for the situation at hand. The main reason I passe on it, is the lack of tune-able parameters in sklearn's GaussianNB implementation. So I don't have a real chance of improving its performance from here, what I see now is what I get (I maybe wrong on this). As touched on in my previous comments decision tree is easy to conceptualize. It is

also fast to use, as predicting data has logarithmic cost on number of features (it had the fastest prediction time above too). A possible problem that I might encounter is its instability in generating a tree consistently upon data variation. Upon closer reading of section 1.6.4 of Sklearn's documentation, "Nearest Neighbor Algorithms", KNN as, I used it here, makes a choice of algorithm based on the data passed to fit(). The choice being between brute force, K-D tree, and Ball tree. This data set is too large for brute force (n = 300 >> 30) so we can consider KNN here as either K-D or Ball tree. Now, since the size of the feature space, is for computational purposes, D = 48 > 20 (after adding all the dummy classes) it is likely it is choosing Ball Tree. In any case its time complexity at prediction is O[Dlog(N)] which explains why its slower than the decision tree. Since the amount of student features is not likely to change much if KNN is put to practice in our scenario (suppose we choose it in the end for use of the school board), this can be consider a pro for this algorithm, since it is essentially logarithmic, just as the decision tree. Also from sklearn;

"*Ball tree and KD tree query times can be greatly influenced by data structure. In general, sparser data with a smaller intrinsic dimensionality leads to faster query times.*"

If I'm understanding correctly (may very well not be) since sparcity of the data set *"refers to the degree to which the data fills the parameter space"* , then it seems to me that the data set here is somewhat sparse. The reason I say this is because when we added the dummies, essentially we added a lot of zero components to each student "vector". Since the student's mother, say, cannot be partially between being a Teacher and Healthcare worker, in regard to those categorical variables, there are regions in the feature space that are empty. Because no vectors will ever have components that are non-zero there. If this is indeed the case, then it would also be a pro for KNN. The advantages of SVC are that it is memory efficient because it only uses a subset of the training points in the decision function. This fits our computational cost constraint. It is also very customizable at the tuning phase (4 kernel options plus parameters). Since we are not concerned with probability estimates here, which it SVC does expensively, it has little downsides. Next I try each of these models with varying training set sizes, from 50 students to 300 in increments of 50, for a total of 6 training set sizes.

DecisionTreeClassifier

	Training time	F1 score training set	Prediction time	F1 score test set
Training samples: 50	0.00134015	1	0.00022912	0.655738
Training samples: 100	0.00187397	1	0.000470161	0.738462
Training samples: 150	0.00248003	1	0.000349045	0.7
Training samples: 200	0.00266218	1	0.000222921	0.692913
Training samples: 250	0.00324202	1	0.000225067	0.694215
Training samples: 300	0.00353885	1	0.000210047	0.744186

SVC

	Training time	F1 score training set	Prediction time	F1 score test set
Training samples: 50	0.00126004	0.868421	0.00067687	0.746667
Training samples: 100	0.00251102	0.853801	0.00161004	0.766234
Training samples: 150	0.0035069	0.867769	0.00148201	0.791946
Training samples: 200	0.00478506	0.870662	0.00180602	0.773333
Training samples: 250	0.00570989	0.874036	0.001755	0.772414
Training samples: 300	0.00794792	0.869198	0.00203085	0.758621

KNeighborsClassifier

	Training time	F1 score training set	Prediction time	F1 score test set
Training samples: 50	0.000905991	0.84507	0.000964165	0.706767
Training samples: 100	0.000568867	0.860759	0.00172305	0.723077
Training samples: 150	0.000732183	0.908257	0.00194407	0.68254
Training samples: 200	0.000803947	0.865772	0.00178885	0.729927
Training samples: 250	0.000707865	0.878453	0.00239015	0.677165
Training samples: 300	0.000725985	0.880361	0.00236201	0.731343

SVC showed consistently higher f1 test scores. The DecisionTree was very fast at predicting. KNN was very fast at training.

Conclusions:

Algorithm Selection:

The decision to use KNeighborsClassifier was mostly due to its low cost computationally. Given a battery of possible tuning parameters it was the only one of the algorithms that was feasably tunable given low resources. It performed better than other cheap alternatives like Naive Bayes, so it was reasonably well performing for its cost. As seen in the benchmark table in cell [82] out of the box it was outperformed only by the SVC, which is more costly to train. Still its improvement after tuning was only marginal.

Layman Explanation: KneighborsClassifier

The mechanism with which our model decides whether a current student will pass or fail is very intuitive. Each student has 30 attributes associated with him/her. These range from basic descriptors like their age, sex, and health, to behavioral descriptors like whether they are in a romantic relationship, how much time they devote to study, if they have any extra curricular activities. Some of the attributes are of things out of there control like whether they have internet access, the size of their family, and what neighborhood they live in. What are model process does, is that it assigns a relevant number to every one of these attributes. Just like for example you can take a house and assign it a longitude, a latitude, and maybe if it sits on a hill an altitude. In the same way that information like longitude, latitude and altitude, alows us to decide how far away two houses are from each other, we can decide how "far away" two particular students are from each other. Based on all those attributes like free time and age and so forth. Since we have information about which students have failed and which have passed, our model basically answers the question; how "close" is this student to other students who have passed. Maybe he is "closer" to students who fail. We can choose to compare him/her to the closest *single* student to determine how likely he is to pass or fail. Usually, however we tune the model to find a *group* of students closest to him/her, maybe the closest 4 students, or maybe 10 closest students. The exact number is determined while tuning. Since we use students that we know either passed or failed in this group, we can determine if our student in question is closer to others who pass or those who fail.

Final F1 score:

0.787