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	0.010 s	0.023 s	0.002 s	0.158 s	0.001 s
F1 score training set	1	0.869198	0.808824	0.868778	0.886878
Prediction time	0.001 s	0.005 s	0.001 s	0.007 s	0.003 s
F1 score test set	0.755906	0.758621	0.75	0.779412	0.721805

After this preliminary experiment, I don't feel I can say any algorithm is particularly suited or ill-suited here. All of them out of the box pretty much give very similar f1_scores on the test set after training. Each have a priori advantages/disadvantages. As touched on in my previous comments decision tree is easy to conceptualize. It is also relatively fast to use, as predicting data has logarithmic cost on number of features. A possible problem that I might encounter is its instability in generating a tree consistently upon data variation. Nearest neighbors also has that advantage of conceptual simplicity. Since the number of features is small relative to the data size in this problem SVC is also a viable option due to how customizable it is during tuning (4 kernel options plus parameters). Adaboost is exclusively a classification algorithm, so is appropriate for this project. Naive bayes is touted as possibly extremely fast. 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	0.002 s	1	0.001 s	0.80597
Training samples: 100	0.003 s	1	0.000 s	0.725806
Training samples: 150	0.004 s	1	0.000 s	0.697674
Training samples: 200	0.004 s	1	0.000 s	0.71875
Training samples: 250	0.005 s	1	0.000 s	0.710744
Training samples: 300	0.005 s	1	0.000 s	0.765625

SVC

	Training time	F1 score training set	Prediction time	F1 score test set
Training samples: 50	0.001 s	0.90625	0.001 s	0.738462
Training samples: 100	0.002 s	0.85906	0.001 s	0.783784
Training samples: 150	0.003 s	0.870813	0.002 s	0.771429
Training samples: 200	0.004 s	0.869281	0.002 s	0.77551
Training samples: 250	0.006 s	0.879177	0.002 s	0.758621
Training samples: 300	0.008 s	0.869198	0.002 s	0.758621

GaussianNB

	Training time	F1 score training set	Prediction time	F1 score test set
Training samples: 50	0.001 s	0.666667	0.000 s	0.468085
Training samples: 100	0.001 s	0.854962	0.000 s	0.748092
Training samples: 150	0.001 s	0.808743	0.000 s	0.736842
Training samples: 200	0.001 s	0.832061	0.000 s	0.713178
Training samples: 250	0.001 s	0.817647	0.000 s	0.746269
Training samples: 300	0.001 s	0.808824	0.000 s	0.75

AdaBoostClassifier

	Training time	F1 score training set	Prediction time	F1 score test set
Training samples: 50	0.104 s	1	0.006 s	0.645161
Training samples: 100	0.095 s	0.953846	0.006 s	0.72
Training samples: 150	0.098 s	0.912821	0.006 s	0.757576
Training samples: 200	0.100 s	0.882562	0.006 s	0.805755
Training samples: 250	0.102 s	0.886427	0.006 s	0.776978
Training samples: 300	0.105 s	0.868778	0.006 s	0.779412

KNeighborsClassifier

	Training time	F1 score training set	Prediction time	F1 score test set
Training samples: 50	0.001 s	0.8	0.002 s	0.761905
Training samples: 100	0.001 s	0.823529	0.002 s	0.666667
Training samples: 150	0.001 s	0.816327	0.003 s	0.677419
Training samples: 200	0.001 s	0.86121	0.002 s	0.666667
Training samples: 250	0.001 s	0.889503	0.002 s	0.711111
Training samples: 300	0.001 s	0.886878	0.002 s	0.721805

The models' f1_scores varied little as training size increased. The SVC seemed the most stable across different training data set sizes. KNN had a strange u-shaped behavior, starting with a highish score decreasing, and then rising again. I decided to further explore them with grid search.

Conclusions:

Algorithm Selection

My code reviewer pointed out that I had used the whole training set while I was still in model selection phase. After correcting this, the results from grid search in cell [100] became more comprehensible. Based on the computations performed there, although all algorithms (even if only marginally) benefited from the tuning process, two of them in my opinion are good candidates for consideration for final selection. These are DecisionTreeClassifier and KNeighborsClassifier. The SVM was very costly to tune with grid search, taking 45 seconds, with the next longest tuning being DecisionTree's at 15.5 seconds. Adaboost had a relatively fast training time of 10 s, but grid search failed to find a combination of parameters that significantly improved its final f1 score. For the the full training set size of 300, the DecisionTree had a marked f1 score improvement from 0.766 (cell [97]'s output) to 0.803. KNeighborsClassifier similarly benefited from the grid search to improve its f1 score from 0.722 to 0.770. In previous attempts, when I found a good combination of parameters, performance seemed dependent on the particular run I had just observed. As hinted to me after the code review, the train/test spliting process can have marked effects on the algorithm's final performance, especially on small data sets like this one. To check that I had found consistently good parameters with grid search, I did a final test by averaging 100 f1_scores for each of 6 random_state seed values in train_test_split. As can be seen from the output of cell [115], both KNeighbors and DecisionTree sustained the good performance exhibited after grid search in cell [100], and also differed little from one another in f1_score. However KNeighbors grid_search training time was a mere 2.9 seconds. Therefore under the rubric of: available data, limited resources, cost, and performance, KNeighborsClassifier is the best model for use in this student intervention system.

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, allows 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

Given the way I selected the KNeighborsClassifier, I present the average and standard deviation of f1 scores across the different train/test splits of data.

Scores:

```
0.787096774194, 0.786666666667, 0.810126582278, 0.802547770701, 0.782051282051, 0.815789473684
```

```
Average f1 score: 0.797379758263
Std for the f1 scores 0.0128035136636
```