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 Training:

Kfold cross validation can be used to get the maximum use of a small data set like the one here. 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). 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. The result of this can be seen in the table below. 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 over-fitting. 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.

	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

Model Evaluation:

DecisionTree: Generates a tree representation of a decision function, where each node in the tree represents an if-then-else decision rule, and each leaf finally places the input in a given category (pass/fail in our case).

Pros:

- Simple to understand and interpret. The decision function can explicitly be drawn out as a tree structure.
- The cost of predicting is logarithmic on the size of the training data.
- It can handle categorical and numerical data equally well.

Cons:

- They are prone to overffiting the data (as can be seen in the table) unless pruned, and/or tuned through its various parameters (i.e. minimum samples per leaf, maximum depth). This can be costly.
- They can be unstable; small variations in the data can generate completely different trees.
- They can create bias trees if some classes dominate.

Chosen/rejected: Chosen

Reasoning: Before we manipulated it, the data was strongly categorical. Even age, though ordered, basically could be thought of as buckets between 15 and 22. Even though the decision tree can handle both categorical and numerical data equally well, overall the structure of this data seems like a good fit for the tree like decision function. We can easily see how if then rules can be deduced from the attributes, things like, *if (has internet connection) then … ,* or *if (father's job == Teacher) then … else if (father's job == Healthcare worker) then …* , so on and so forth. The data is also not too badly out of balance (approx 2:1 pass to no pass ratio) which mitigates one of the cons. Tunning the tree to improve its performance may prove costly and there is already some evidence of over-fitting above. Using the model for prediction is fast which would be advantageous in a situation of limited computational resources.

SVC: Seeks to maximize the decision boundary between classes by solving a quadratic programming problem.

Pros:

- Effective for data with a high number of attributes
- Still effective when there are more attributes than data points
- Its memory efficient because only a subset of the data is required (support vectors)
- Highly tunable at the tuning phase (also a con). In sklearn's implementation we have 4 kernel function (plus the ability to define a custom kernel) plus around a dozen parameters

Cons:

- Harder to conceptualize relative to other models. The entire mechanism of the algorithm is a rather abstract linear algebra/analysis problem. (Though this con only really applies when seeking to interpret results in an intuitive way)
- Though both its training time and prediction time were relatively fast out of the box, the model can be costly to tune do to its sheer number of configurations (can also be a pro).
- Can only expensively provide probability estimates. (Not so relevant here)

Chosen/Rejected: Chosen

Reasoning: For this particular problem since we are not required to estimate any probabilities there seems to be little downside to using SVC. On the other hand, some of its strengths are also not particularly relevant since, the number of attributes is pretty small compared to the size of our data set. Its memory efficiency definitely fits within a scenario were we have small resources. Its f1 score on the test set was also best out of all, out of the box.

GaussianNB: Applies Bayes theorem with the assumption of independence between every pair of attributes. From the training data it calculates particular P(**x**|y)'s (i.e. given a label what is the probability of the input point **x**) fits them to a Gaussian distribution. It then uses that model, applying Bayes, to approximate P(y|**x**) for new inputs **x.**

Pros:

- Requires only a small amount of training data to estimate the necessary parameters.
- It can be very fast compared to more sophisticated methods (evidence of this can be seen in the table)

Cons:

- Bad estimator of probabilities. (Not so relevant here)
- Sklearn's implementation has little in the way of tune-ability.

Chosen/Rejected: Rejected

Reasoning: 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 pass 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).

AdaBoostClassifier: It trains multiple weak classifiers (weak meaning they are better than guessing i.e. generalization error < 0.5), and then combines them into a single *boosted* classifier using a weighted voting scheme.

Pros:

- Computationally efficient (taken from Intro to boosting pdf)
- No difficult parameters to set
- Versatile – a wide range of weak learners can be used

Cons:

- Weak learner should not be too complex to avoid overffiting
- There needs to be enough data so that the weak learner requirement is satisfied.

Chosen/Rejected: Rejected

Reasoning: From the above table, AdaBoost seems 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. Also from the Intro to Boosting reading, it seems that choosing the weak learner properly is particularly important for this algorithm to perform well. Simply using the default DecisionTree weak learner from sklearn's implementation, resulted in the somewhat costly, and not particularly strong performance seen in the table above. I experimented briefly using grid search and a parameter set (the DecisionTree was left as the weak learner choice) but did not achieve significants gains in performance given the computational cost. This leads me to believe that experimenting with the choice of weak learner would be more fruitful, but I opted to postpone such a study for later.

KNeighborsClassifier: Given an input point, **x**, it finds the k closest points to **x** in the training set and then applies a majority voting scheme regarding their labels, to determine the label for **x**.

Pros:

- Relatively fast prediction time in general, O[Dlog(N)] where D is the number attributes and N the number of training examples (D is unlikely to change much in this scenario, so its really more like O[log(N)])
- Easier to conceptualize and reason about than other more sophisticated models.

Cons:

- Not memory efficient, since it makes predictions by directly using the data as a "model". Must therefore keep the data stored.

Chosen/Rejected: Chosen

Reasoning: 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 sparsity 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 Health care 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. However if our training data set were to grow very large, this will come at significant memory cost.

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 feasibly 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, 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:

0.787