# Project 2

## Building an Intervention System

By Robert Lutz

January 26, 2016

Rev 2.0

# 1) Classification vs Regression

In this project we strive to build a machine learning system that identifies students in need of early intervention to address potential future academic difficulties. By feeding an algorithm with training data containing many former students' backgrounds and academic performances, we anticipate being able to predict which current students are at risk of failing their high school graduation exam due to similarities to former students who failed. The input data are mostly categorical, although it will turn out that this fact is of no consequence in determining whether the problem is one of classification or regression. The output we desire from the algorithm will be a prediction of the 'yes'/'no' type to the question 'do we expect this student to pass the final exam?' This output is clearly categorical.

Let's take a moment to discuss the distinction between categorical and continuous data. Although the boundary is fuzzy, one can think of the situation as follows. Categorical data is data on which it is meaningless to do basic arithmetic, for example eye color, parents' occupations and social security number. Continuous data is data on which arithmetic could be meaningful, such as distance from home to school and parents' income. But again the boundary is fuzzy. Is age categorical or continuous? If we are dealing with students who are all close to the same age, so that maybe there are three categories like 16, 17 and 18, then we might be better served by considering age categorical data. Ultimately we get to choose. (In theory at least; scikit-learn requires continuous feature data). Usually the choice to treat a category as categorical or continuous will be apparent.

There are two types of supervised learning problems: classification and regression. Classification problems are ones whose output consists of categorical data, whereas regression problems output continuous data. Because this output is the case of the early intervention system is categorical, our problem is one of **classification** rather than regression. Note that this has nothing to do with the fact that the inputs are mostly categorical, and everything to do with the fact that it's the output that's categorical. One could certainly have regression problems that contained entirely categorical inputs, as well as classification problems that contained entirely continuous inputs.

# 2) Exploring the Data

We are provided with data on **395** students, **265** of whom went on to pass while **130** ended up failing, yielding a graduation rate of **67.09%**. Because there are about twice as many passing students as failing, the accuracy metric might not be ideal for scoring our model; we choose to use $F_1$ score instead, which works better when the data is skewed. In addition to whether they passed or failed, we are provided data about the students' schools, genders, ages, addresses and 26 other features for a total of **30** features. Table 1 summarizes the relevant data.

Table 1: Relevant Data from the Student Dataset

| Parameter | Value |
|---|---|
| Number of Students | 395 |
| Number of Passing Students | 265 |
| Number of Failing Students | 130 |
| Graduation Rate (%) | 67.09 |
| Number of Features | 30 |

## 3) Preparing the Data

We find that the features data are not necessarily in the form we would like them to be. Many of the categorical features have character or string values, a situation which apparently does not lend itself well to interacting with the scikit-learn API. So it needs to be changed. We have chosen to use a function in the pandas API called `get_dummies` to split such char/string columns into columns that are effectively T/F (actually 1/0) for a particular attribute of a feature.

For instance, the 'sex' feature has been split into 'sex_F' and 'sex_M' features. The value of the 'sex_F' feature would be 1 if the value of the original 'sex' feature had been 'F' and so on. The exception to this rule is when the alphanumeric values of a feature are 'yes' and 'no', in which case the feature is maintained and 'yes'/'no' is directly converted to 1/0. Although the original code provided by Udacity left the label column unchanged with string values 'yes'/'no', we have decided to convert it to the integer 1/0 format in order to allow subsequent use of $F_1$ scoring rather than the default accuracy when we run grid search. Since we will eventually compare models based upon their $F_1$ scores, it makes sense to choose the best hyperparameters based on the $F_1$ score. The $F_1$ scoring algorithm in scikit-learn will not accept string values as its positive. Other changes to the code have been made to ensure compatibility with this modification.

Now that the data has been properly prepared, we are ready to split them into training and test sets. We choose a training set size of 300, which leaves 95 samples for the test set. We perform the split using the `train_test_split` function provided in the scikit-learn API.

## 4) Training and Evaluating Models

We require a thorough understanding of our data set in order to make a good decision about which models to investigate. What can we say about it? Well, first of all, we know that this is a classification problem. So we can throw out linear/multivariate regression immediately. More

interestingly, however, is that fact that there are very many features given the data set size: 30 features for only 395 total data points. That we are considering so many dimensions indicates we haven't utilized or simply may not have a great deal of domain knowledge, i.e., the knowledge of experts in secondary education that tells us what factors likely do or do not contribute to a student's likelihood of failing the graduation exam. Because we haven't excluded much *a priori*, we are forced to investigate a rather large number of features given a comparatively small number of data points.

The consequence of this is that we will need to work hard to avoid variance error do to overfitting. 395 data points would probably be sufficient to give us a good fit to to a feature space of perhaps five or six meaningful dimensions, but in 30 dimensions the data points would be so spread out that it may prove difficult to identify the true pattern in the data. This is the curse of dimensionality. It affects some models worse than others. The *k*-nearest neighbors is particularly hard-hit by the curse of dimensionality: with all of those dimensions, most of them irrelevant, how does one ensure a good metric of distance that reveals the true relationship between label and features? After all, the algorithm doesn't know which features are important and which aren't, it just finds the *k* nearest neighbors to the data point in question and predicts that point's label based on the neighbors. For that reason, we exclude *k*-nearest neighbors from further consideration.

There's one potential bias we need to disabuse ourselves of: despite the fact that we have 30 unique features to work with, there may very well be a good deal of residual noise remaining in the data. Whether or not this is true remains to be seen, but we would be wise not assume we will be able to explain away all variation with the space of 30 features available to us. Models that are tolerant of noise may well be worth investigating.

We are ready now to decide upon which three models to investigate. However much we would like, we can't investigate neural networks, since scikit-learn has not yet released a supervised neural network learner. So our list of potential models has been thinned down to three. We choose **decision trees** due to their speed and explanatory power, **support vector machines** because they offer the 'kernel trick' which may prove handy and **naive Bayes** as it works well on small, noisy data sets. Let's take a closer look at each in turn.

## 4.1) Decision Trees

Decision trees are learning models that predict target values based on simple decisions involving questions about feature data [scikit-learn]. They have wide applicability, being capable of both classification and regression. They train fairly quickly ($O(v^2n)$, where $v$ is number of features and $n$ is number of samples) [Su] and are capable of dealing with large data sets (prediction time is $O(\lg n)$) [scikit-learn]. Another benefit of decision trees is that they are 'white boxes', i.e., it is possible to understand precisely how they arrived at their predictions. In fact, one can produce graphical output from decision trees in scikit-learn in conjunction with Graphviz. This can be very helpful in understanding better the underlying data set. Having said that, decision trees, like most other algorithms, struggle when faced with lots of features. While that is the situation at hand, we can deal with it through the application of grid search and cross validation. While it's difficult to say that our data set is obviously well suited to learning by

Table 2: Relevant Data from the Decision Tree Classifier

| | Training Set Size | | |
|---|---|---|---|
| Parameter | 100 | 200 | 300 |
| Training Time (ms) | 1.38 | 1.69 | 2.02 |
| Prediction Time on Training Data (ms) | 0.56 | 0.60 | 0.66 |
| Prediction Time on Test Data (ms) | 0.56 | 0.56 | 0.57 |
| $F_1$ Score on Training Data | 0.820 | 0.816 | 0.813 |
| $F_1$ Score on Test Data | 0.792 | 0.807 | 0.808 |

decision tree, trees are so quick, flexible, and enlightening that it is worth investigating them anyway.

Key parameters for the decision tree model are summarized in Table 2. We see several interesting things. The training time and prediction time on the training set increase very nearly linearly with training set size. Prediction time on the test set is constant. The $F_1$ score on the training data decreases slightly with increasing training size. This is anticipated, as it is easier to find a great fit to a smaller training set than a larger one. The $F_1$ score for the test set is depressed at training size 100, then basically plateaus at training size 200. Thus we could make a good argument that a training size of at least 200 is necessary and sufficient to obtain optimal predictive capability from a decision tree.

## 4.2) Support Vector Machines

Support vector machines (SVMs) are a relatively recent invention based on a rather ingenious and involved mathematical framework. They work best as binary classifiers, which is just what we are looking for. They work well even when the feature dimensionality is high with respect to the number of data points, again much like our current situation. Because the SVM algorithm whittles down a large data set to just a few important ones (the support vectors), they are quite memory efficient. And perhaps their greatest advantage is the so called 'kernel trick', which allows for non-linear relationships between features and labels to be linearized. They do have some downsides, of course. They work best when there is a clear distinction between classes, i.e., little noise. Despite the fact that we have 30 individual features, it's not at all axiomatic that our situation is low-noise. SVMs do not scale well to large datasets; training time is $O(n^3)$.

The relevant data for the SVM classifier are summarized in Table 3. We see that the training time and prediction time on the training data seem to scale superlinearly with training set size. It is possible that the run time increases as $O(n^3)$ as we would expect from theoretical considerations, but we have insufficient training set sizes to make a judgement. In any case, training or predicting with SVMs on larger datasets can be a slow process. The prediction time on the test data increases nearly linearly. The $F_1$ scores on the training data are nearly constant. The $F_1$ scores on the test data are slightly better than the decision tree (see Table 2), and seem

Table 3: Relevant Data from the Support Vector Machine Classifier

| | Training Set Size | | |
|---|---|---|---|
| Parameter | 100 | 200 | 300 |
| Training Time (ms) | 3.87 | 10.82 | 21.95 |
| Prediction Time on Training Data (ms) | 3.01 | 9.59 | 20.14 |
| Prediction Time on Test Data (ms) | 2.87 | 4.86 | 6.78 |
| $F_1$ Score on Training Data | 0.833 | 0.833 | 0.831 |
| $F_1$ Score on Test Data | 0.804 | 0.809 | 0.811 |

to plateau at a training size of 200. In this case one could argue that a training set size of 200 is necessary and sufficient for optimal predictive capability.

## 4.3) Naive Bayes

The Naive Bayes classifier utilizes Bayesian reasoning to identify the most likely of a set of hypotheses. It is 'naive' because it implicitly assumes there is no dependence of one feature on another. It works well when there is a lot of noise in that data, which may be the case with our dataset. It also works well on smaller training data sets [scikit-learn], much like ours. Naive Bayes may not perform optimally if the assumption of feature independence is blatantly violated.

Relevant data for the naive Bayes classifier is provided in Table 4. The training time and prediction time on the training data scale very nearly linearly with training set size. The prediction time on the test data is for all intents and purposes constant. Training and prediction are quite a bit quicker than with SVMs, but not as good as with decision trees. The $F_1$ scores on the training data are substantially lower than those observed for either the decision tree or the SVM. Despite what we said earlier about naive Bayes being a good choice for a small training set size, we see that the $F_1$ score for the 100 sample training data is really quite poor. Perhaps the naive assumption of feature independence has been violated. This is just speculation; a detailed analysis of feature correlation would be required to prove this conjecture. The $F_1$ scores for the test data are also quite low--very poor with a training set size of 100, and, while showing considerable improvement, still not competitive with decision trees or SVMs at larger training set sizes. In order for the naive Bayes model to be at all competitive with the others investigated, we must utilize a training set size of 300 data points.

Table 4: Relevant Data from the Naive Bayes Classifier

| | Training Set Size | | |
|---|---|---|---|
| Parameter | 100 | 200 | 300 |
| Training Time (ms) | 2.29 | 2.61 | 2.96 |
| Prediction Time on Training Data (ms) | 1.11 | 1.44 | 1.77 |
| Prediction Time on Test Data (ms) | 1.09 | 1.10 | 1.10 |
| $F_1$ Score on Training Data | 0.677 | 0.792 | 0.797 |
| $F_1$ Score on Test Data | 0.579 | 0.732 | 0.754 |

## 5) Choosing the Best Model

We have come to the point when we must decide upon the model to recommend to the school board of directors. A summary of the data relevant to this decision is shown in Table 5. There are a couple main takeaways from this table. First, the Naive Bayes classifier does not do nearly as good a job at classifying passing students as do the other two models, a fact apparent from a cursory glance at the $F_1$ scores. The decision tree classifier and support vector machine classifier are comparable in their classification performance as measured by $F_1$ score. Second, the decision tree and naive Bayes classifiers are substantially faster than the SVM. So taking classification ability and train/prediction speed into account, we must conclude that the **decision tree** classifier is the right one for the job. Note that we do not anticipate a systematic application of grid search with cross validation to change our recommendation. The `GaussianNB` learner in scikit.learn takes no hyperparameters, and grid search will do nothing to identify a faster variant of support vector machine.

Let's go ahead and optimize the decision tree classifier using grid search with cross validation. In order to do a proper grid search, we need to change the default scorer from accuracy to $F_1$ score. Since we are using $F_1$ score as the final arbiter of the best model, it makes a lot of sense to use that selfsame $F_1$ score to extract the 'best' model from grid search. Recall that we earlier mentioned that several changes were made to the data label values and subsequent

Table 5: Comparison Table for the Three Models

| Model | Training Set Size | Training Time (ms) | Prediction Time Test (ms) | $F_1$ Test Score |
|---|---|---|---|---|
| Decision Tree | 200 | 1.69 | 0.56 | 0.807 |
| SVM | 200 | 10.82 | 4.86 | 0.809 |
| Naive Bayes | 300 | 2.96 | 1.10 | 0.794 |

code in order to enable $F_1$ scoring in grid search. We will investigate `max_depth` from 1 to 10 and `min_samples_leaf` between 2 and $2^6$. We find something very interesting: while the `min_samples_leaf` is an unsurprising 32, the ideal `max_depth` is 1! An optimal decision tree trained on a full 300 sample training set shows an $F_1$ score of **0.808**.

Let's try to understand why the ideal tree is so stumpy. An example depth 1 tree is shown in Figure 1. We see that the singular indicator of passing the final exam is no prior failures. Beyond that, grid search is unable to reliably and consistently identify other factors that contribute to passing the final exam. While we had a prior suspicion that most of the features must turn out to be irrelevant as predictors of passing, it is surprising to see that only one feature can be said to truly matter. It may very well be that with additional student data additional features would be revealed as important. Alternatively, it's possible that our domain knowledge has missed the mark and the best predictive indicators have been left off the feature list. So it is worth noting that the choice of `max_depth` 1 is not to be thought of as carved in stone. In the future, as more data become available or additional features are identified, we should revisit grid search with a view toward increasing tree depth by identifying additional predictors of passing.

Let's take a moment for a brief overview of the process by which a decision tree classifies data. A decision tree classifier is an algorithm that seeks to split the classified training data into subgroups such that the subgroups are on whole 'cleaner' than the original group. By cleaner we mean that, in our situation, one subgroup contains a higher proportion of 'yes' values than the original group, while the other contains a higher proportion of 'no' values. We see from Figure 1 that the principal determinant of passing or failing the final exam is a history of prior failures, even just one. The decision tree chose to split on this features based on a quantity known as information gain, which we will not discuss in more detail here.

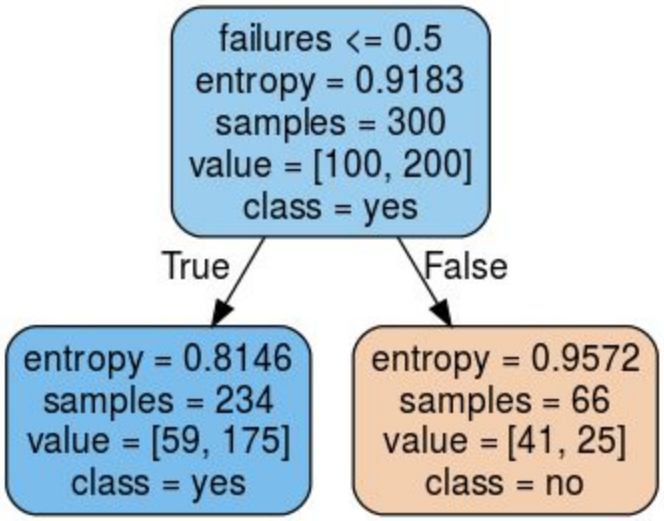


Figure 1: Example Optimal Decision Tree with `max_depth`=1

Now, despite the tree shown in Figure 1, it's not necessarily the case that the tree be branched only once. The branching process described above can be repeated on each new node created by the previous split. For example, the node on the lower left in Figure 1 containing 59 'no's and 175 'yes's could conceivably be branched again, perhaps by a feature relating to the gender of the student, to produce two new nodes. At the same time, the node on the lower right containing 41 'no's and 25 'yes's could hypothetically be branched again, perhaps by a different feature relating to the student's alcohol consumption. Or it could be branched by the same feature as it's sibling node. Or maybe the algorithm would decide not to branch it further. In this way the tree can keep branching and branching--as long as there are more features to consider and more data to be split--in the most optimal way as determined by the decision tree algorithm, utilizing the concept of information gain. Once the tree is fully branched, the terminal unbranched nodes, called leaves, are assigned a value to output. In our case, a leaf with more 'yes's than 'no's would output 'yes' when queried, and vice versa. Training of the decision tree classifier is thereby complete.

Once training is complete, the decision tree classifier is ready to make predictions. We can feed it data pertaining to a particular student and the classifier will then run through the decision tree to make a prediction. For instance, suppose we fed our depth 1 decision tree from Figure 1 data pertaining to a female student who has three alcoholic drinks a week, no prior failures, etc. The decision tree first asks 'Does this student have any prior failures?' Since she does not, the decision tree follows the left branch and reaches a leaf--an unbranched decision point--that contains a decision output 'yes'. The tree predicts that she will pass. In this manner the decision tree can make predictions on any given student for which we have data.

Before we close, it might be worthwhile to compare the $F_1$ score of the optimal decision tree--0.808--to that of a 'dumb' classifier that looks at all the training data and simply guesses 'yes' to each student data vector it is given. In our original data set we have 265 passes and 160 fails (see Table 1). The 'guess-yes' classifier would consider the 265 passes to be true positives ($p_t$) and the 160 fails to be false positives ($p_f$). There are no false negatives ($n_f$) according to the 'guess-yes' classifier. So we can calculate the precision as

$$P = \frac{p_t}{p_t + p_f} = \frac{265}{265 + 160} = 0.6709$$

The recall can be calculated as

$$R = \frac{p_t}{p_t + n_f} = \frac{265}{265 + 0} = 1$$

Thus we calculate the $F_1$ score for the 'guess-yes' classifier as

$$F_1 = \frac{2PR}{P + R} = \frac{2(0.6709)(1)}{0.6709 + 1} = 0.803$$

That an $F_1$ score of 0.803 from a dumb classifier is nearly as good as the best $F_1$ score achieved by the optimal decision tree is somewhat disappointing. In fact, in looking at Table 5, the 'guess-yes' classifier gets a higher $F_1$ score than does the best Naive Bayes classifier! What can we conclude from this? A couple things perhaps. First we probably have too little student data. Substantially more--on the order of 10x or 100x--would allow us to develop better models. Secondly, we may very well have failed to include some of the most important features in our

feature space. We may wish to go back, consult with experts in secondary education, and attempt to identify a new space of features that would be worth pursuing further.

## Bibliography


Alpaydin, *Introduction to Machine Learning, 2nd Edition*, 2010

Mitchell, *Machine Learning*, 1997

Scikit-Learn, http://scikit-learn.org (see sections 1.4, 1.9, 1.10 of the User Guide)

Su & Zhang, 'A Fast Decision Tree Learning Algorithm', 2006

Udacity, www.udacity.com (see videos related to Machine Learning Nanodegree)