Machine Learning - Assignment 1

Jonathan McEntee

9/21/2018

1 Blood Transfusion Dataset

The first data set I train the models on comes from the donor database of a blood transfusion center in Taiwan. The database includes features our models can train on, including the recency of the donor's last donation, and the number of total donations they've made. For a full description of features see the UCI website.

The objective here is to predict a binary variable indicating if the donor donated blood in March 2007. 10% of the data was withheld and put into a test set. The remaining 90% was placed in the training set to be fed into our models. The entire dataset contains 748 samples including 570 positive samples (people who donated blood in March 2007) and 178 negative samples. As the classes are somewhat unbalanced, I stratified the test train split.

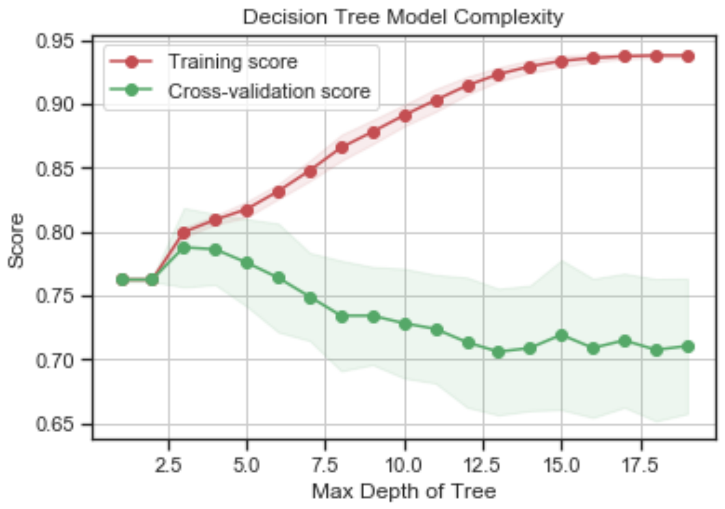
1.1 Why is this interesting?

Because through it we can attempt to predict the turnout of donors. Turnout prediction is a key problem in a number of fields, and one I'm eager to work on.

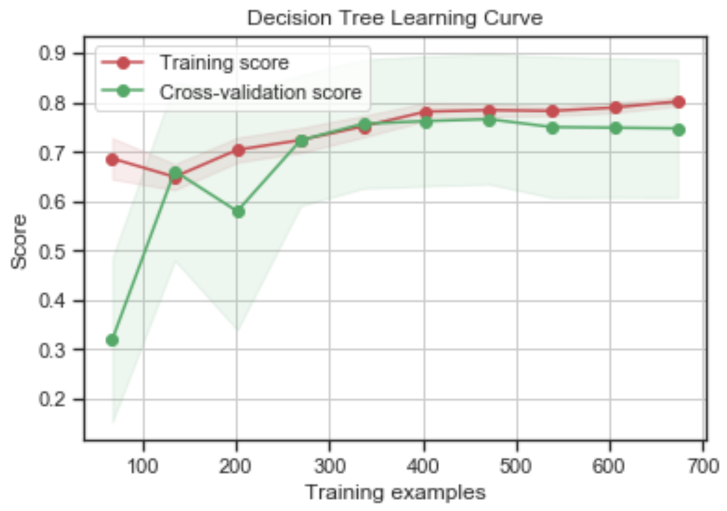
1.2 Decision Tree

The decision tree, like all the other models I trained for this paper, was taken from the scikit-learn package available for python. The decision tree here is based on the CART algorithm. Rather than entropy, a gini score is used to choose where to split the data. The decision tree can train on both categorical and numerical features.

I began my analysis by performing a grid search over the hyperparameters max_depth and min_samples_split to see which produced the highest cross-validation score (10 folds). The values max_depth = 3 and min_samples_split = 140 appear to produce the highest cross-validation score, so we'll use those as a starting point.



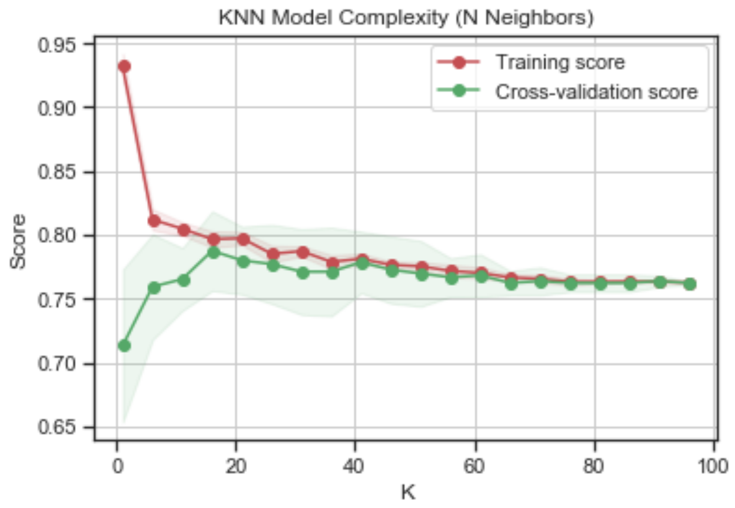
As the max_depth of the decision tree increases, the training score moves asymptotically towards about 93% accuracy (Why not 100%? See section 1.4). Meanwhile the cross-validation score peaks at 3 or 4 depth and then plummets from there. From this graph we see that the algorithm is able to construct trees that better fit the training data as max_depth grows past 4, but those increasingly elaborate trees do not generalize well. Which is exactly what we expect to see.



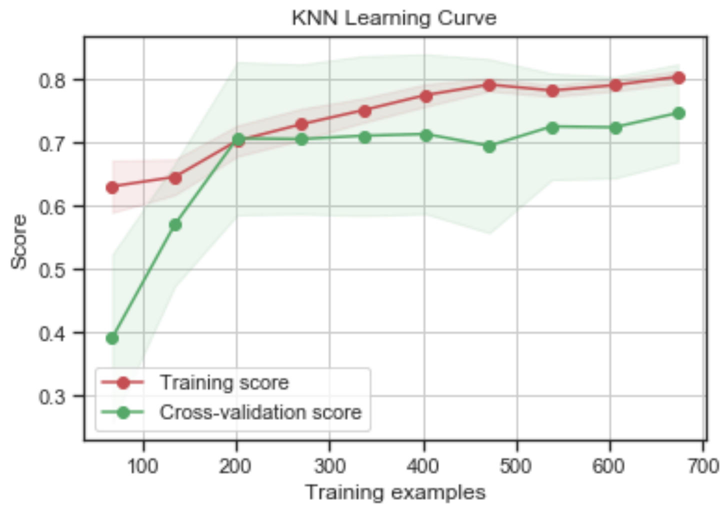
The learning curve for the decision tree shows both the training and cross-validation scores steadily increasing as more samples are made available. The training curve grows at roughly the same rate, showing that the gains made by having more training data are generalizing reasonably well.

1.3 K Nearest Neighbors

Our grid search suggests that the best parameters are setting n_neighbors = 16, p = 2 (euclidean distance), and weights = 'uniform' (no distance weighting). We'll use these parameters as a starting point



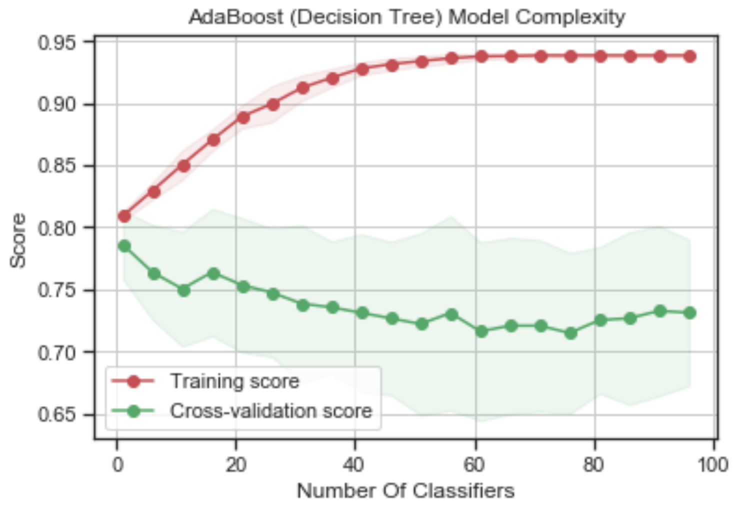
As we increase K the training score falls and the cross-validation score rises, until K reaches about 40, at which point cross-validation begins to fall as well. As K increases so does the bias of the model, and ultimately the scores converge. This plot suggests our grid search chose an appropriate value for k_neighbors.



The learning curve is similar to the decision tree. The cross-validation score rises with the training score, which shows that the model is not overfitting. The distance between the training and cross-validation curves suggests high variance in the model. This makes sense given we chose a low value for K.

1.4 AdaBoost

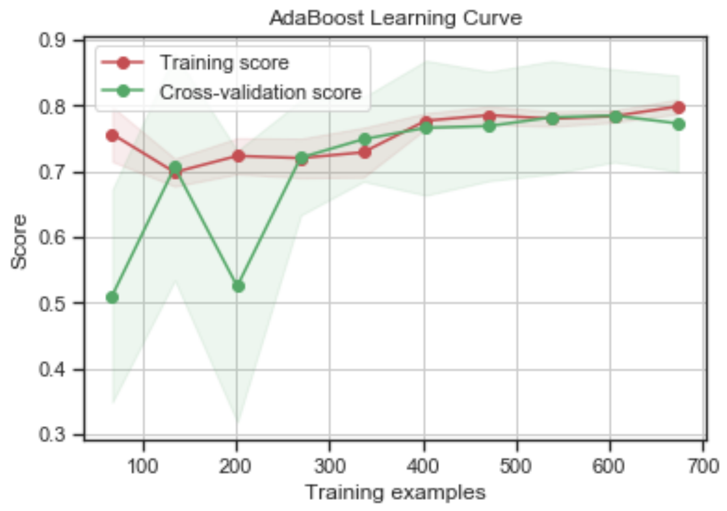
Running a gridsearch for the AdaBoost model gives us learning_rate = 0.1 and n_estimators = 29 as optimal parameters. We set our decision tree parameters at max_depth = 4, min_samples_split = 40, the same parameters we chose for our final model before.



Surprisingly a single decision tree appears to perform better than multiple boosted trees. As the number of estimators increases, the training score rises, but the cross-validation score falls. This signals that the model overfits with more estimators.

A brief aside: the AdaBoost model should be able to perfectly fit the training data with enough estimators. So why does it asymptotically approach 93%? It turns out that about 30% of the dataset has features that are exactly the same, but some of these samples with identical features will have a different classification. As such, there are “two versions of the truth” on a large percentage of the data.

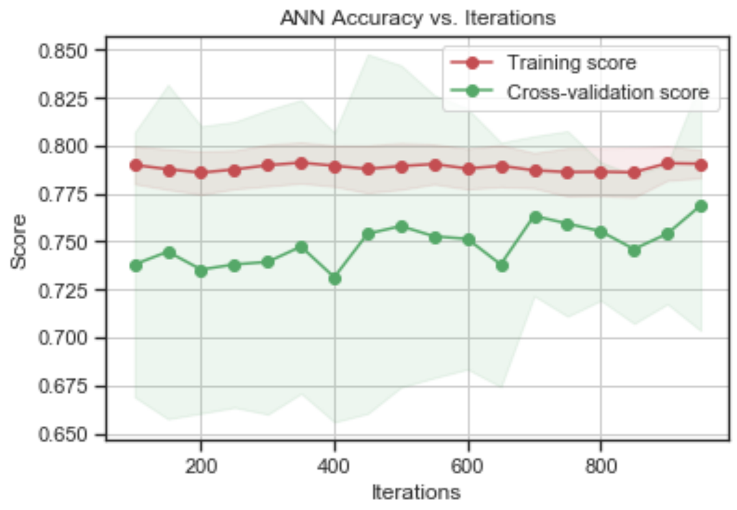
I thought this problematic data might be behind the AdaBoost model performing more poorly than the decision tree, but removing the data and retraining the model produces a similar result. I set the number of estimators at 3 to avoid overfitting the data.



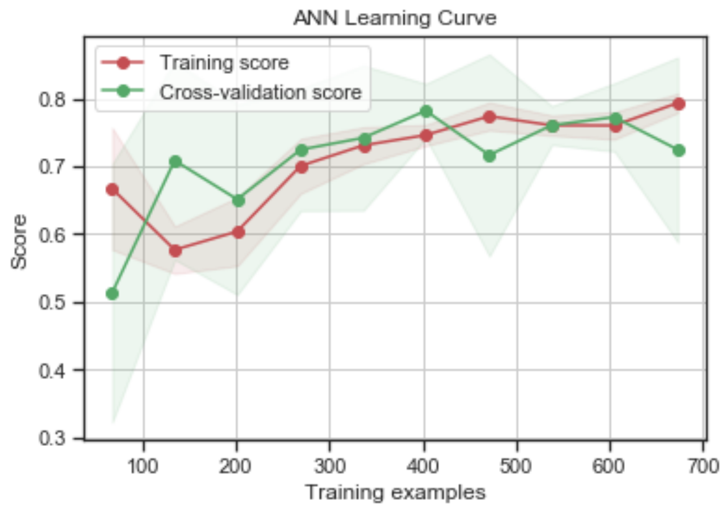
The learning curve shows the training score and cross-validation score converge within 300 samples. By 500 samples the scores seem to have stopped rising.

1.5 Artificial Neural Network

Our grid search over the neural net algorithm suggests three hidden layers with 80 nodes each is the optimal model for this problem.



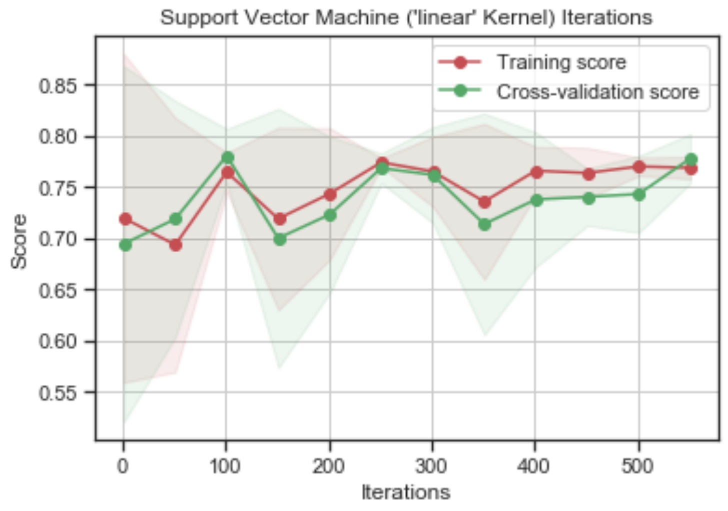
Looking at how the model trains over iterations, the ANN has many peaks and values in cross-validation accuracy. The training accuracy stays roughly the same.



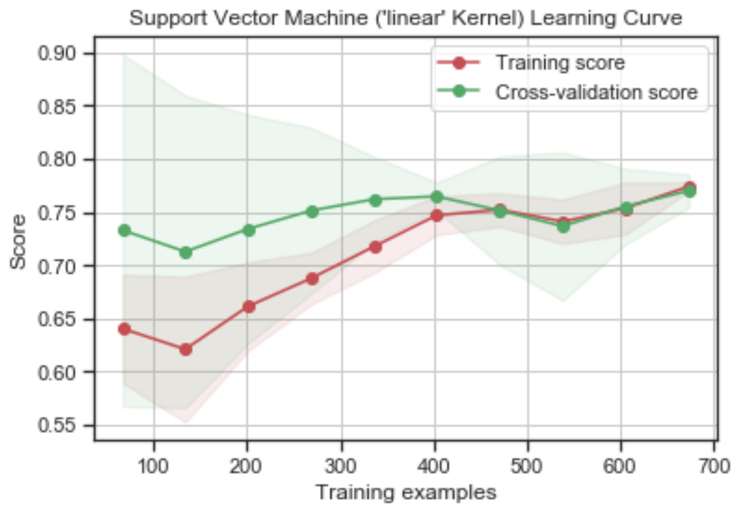
The learning curve is similar to other models we've trained so far. The training and cross-validation scores rise together. At first rapidly and then leveling off.

1.6 Support Vector Machine

A grid search suggests using a radial basis function kernel, with C = 100 and gamma = 0.001. But further experimentation shows a linear kernel with C = 0.1 to be superior (see 1.7 below).



The SVM rises and falls in accuracy as it tries to fit the hyperplane. This is true for both the training and cross-validation scores. This is different from the ANN iterations graph which saw training error stay relatively flat while cross-validation error rose and fell.



Strangely the cross-validation score starts out above the training score. Though as more samples are added the two scores converge. This suggests the SVM ultimately has a high bias. That makes sense as this kernel is simply trying to linearly separate the two classes with a hyperplane.

1.7 Model Scores

Model	Accuracy	Scoring Time	Training Time
K Nearest Neighbors	78.7%	3.6 ms	5.5 ms
Support Vector Machine (Linear)	78.7%	1.6 ms	35.2 ms
AdaBoost	77.3%	2.3 ms	14.2 ms
Decision Tree	77.3%	1.8 ms	4.4 ms
Neural Net	77.3%	3.1 ms	247.5 ms
Support Vector Machine (RBF)	65.3%	2.8 ms	30.5 ms

The models are very close to each other when scored on the test set. In fact, some models score exactly the same. KNN and the linear SVM have 78.7% accuracy. AdaBoost, the decision tree, and the neural net all have 77.3% accuracy. A bug in the code is possible, although scikit-learn is thoroughly tested. Maybe the models all selected hypothesis functions with identical output.

The decision tree has a shorter training time than KNN, but since our max_depth was set to 4 that's not terribly surprising. Other than that, KNN outperforms the eager learners on training time, but is the slowest on scoring time. The neural net has by far the longest training time, the result of needing to train 240 nodes and their weights for its hidden layers.

2. The Adult Dataset

The adult dataset, retrieved from the UCI Machine Learning Repository, has over 48,000 instances each representing an adult with information on age, level of education, race, sex, and a number of other features to train on. A full listing of features can be found on the UCI website.

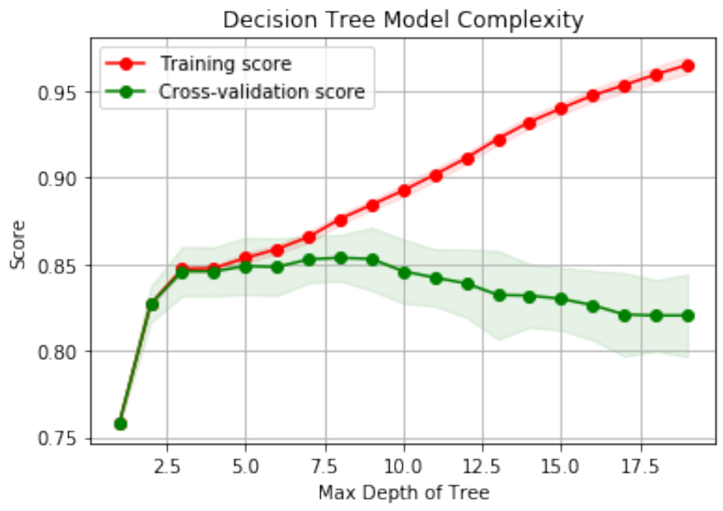
The objective is to use these features to predict whether the subject's income is greater than $50,000. Due to time constraints, I subsampled the 48,000 samples down to 6,000. I then split 10% of that dataset off to be a test set. The 6000 instance subsample has 4547 negative examples and 1453 positive examples. The test set was stratified.

2.1 Why is this interesting?

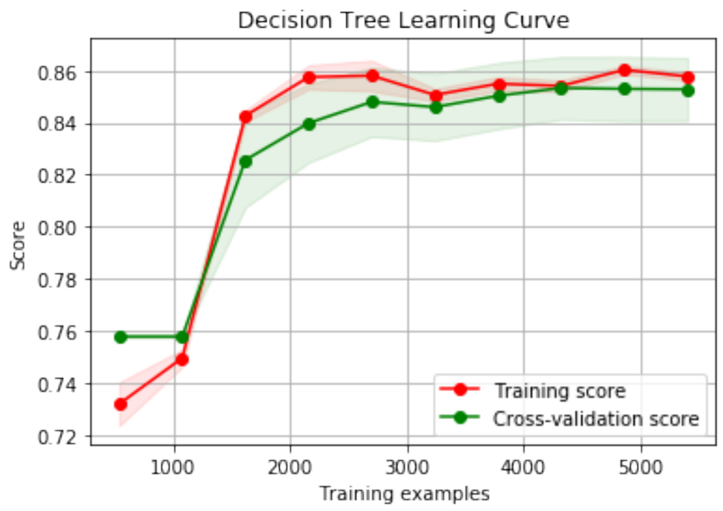
This dataset is interesting because it enters the realm of social sciences. Datasets like this can be used to analyze society-wide trends, and can help us learn what causes societal problems like inequality.

2.2 Decision Tree

I performed a grid search using 10-fold cross validation to determine the optimal hyper-parameters. The result suggested max_depth = 9 and min_sample_split = 640. To make sure this was the correct hyper-parameter setting, I scanned over max_depth and plotted the effects of different parameter values on the model's accuracy.



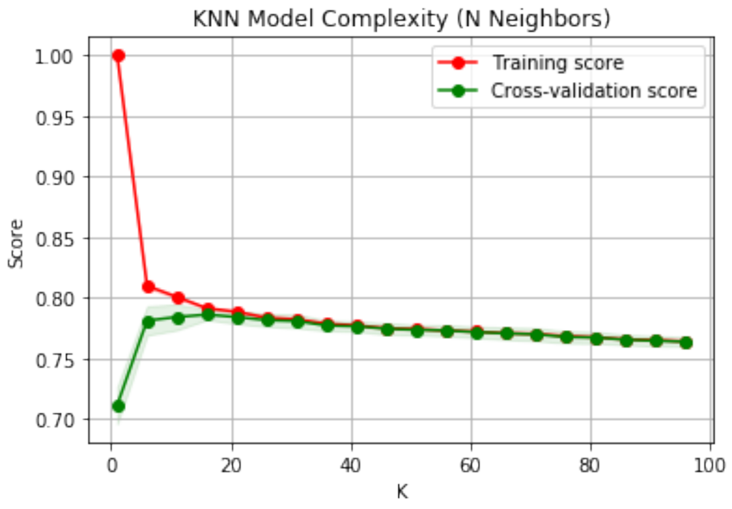
At first the decision tree rapidly gains accuracy as the maximum depth increases. This indicates that it's able to find a few powerful features to branch on that produce significant accuracy. The accuracy continues to climb slowly until a max depth of 9, what we expected. From there the training and cross-validation curves diverge, showing the model begins to overfit the data above a max depth of 9.



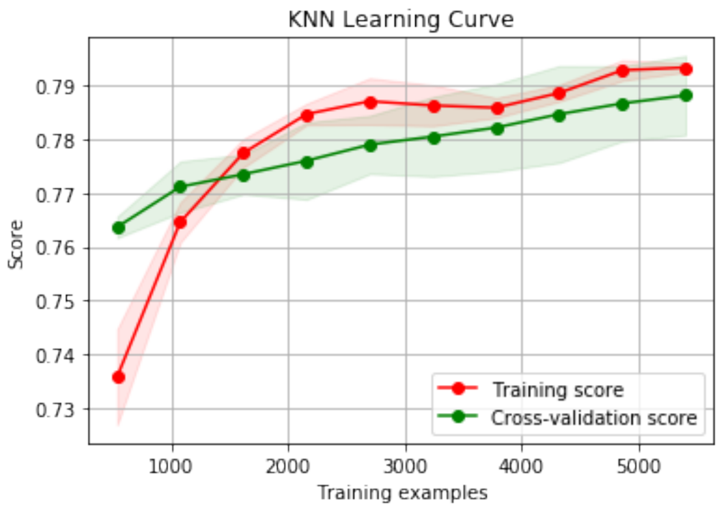
Learning occurs most rapidly between 1000 and 2000 samples, but continues to steadily increase up to 4000 samples and levels off from there. This suggests we have enough data to train the model.

2.3 K Nearest Neighbors

A grid search finds the optimum model for our dataset is unweighted, has K = 17, and p = 2 (euclidean distance).



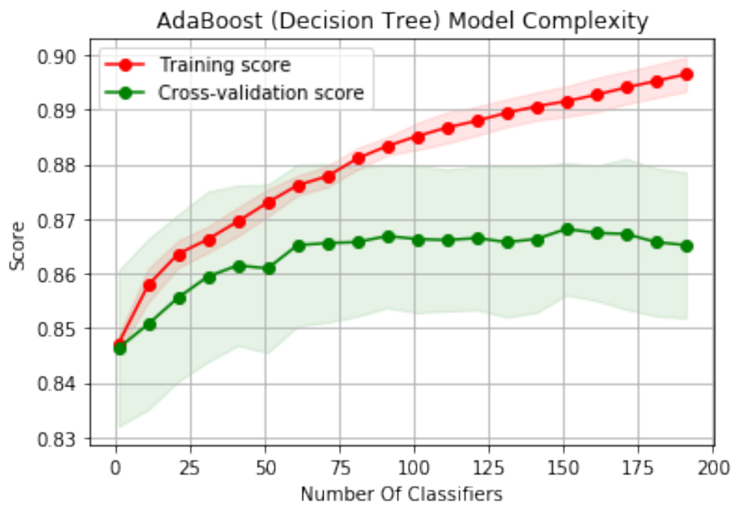
Plotting the training and cross-validation accuracies over different values of K confirms that 17 is an optimal value. We see that at K = 1 the training accuracy is 100%, as we expect. Moving from K = 1 towards K=17 the training accuracy decreases, but the cross-validation accuracy increases, showing the model is underfit before K = 17. Moving forward from K = 17 both training and cross-validation accuracy fall, showing the model is becoming overfit.



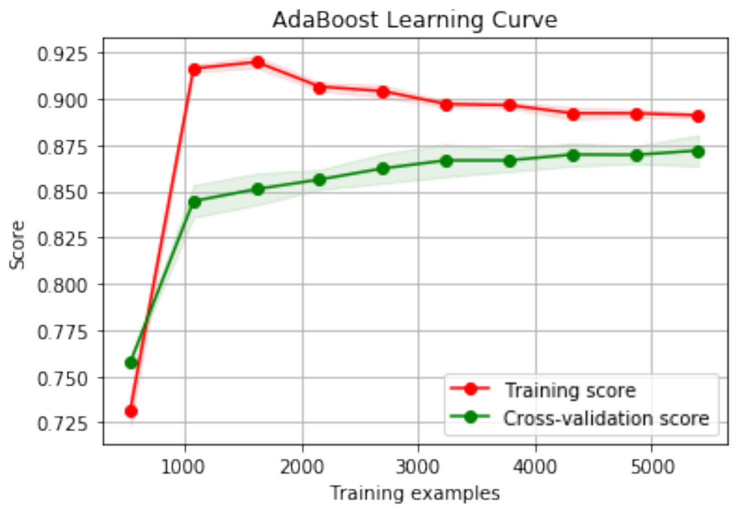
The cross-validation score continues to rise consistently as samples are added. This model could probably reach a higher accuracy with more samples. As is, it performs significantly worse than the decision tree. It's worth noting that this dataset has over 100 features when one-hot encoded. KNN might be struggling to deal with the high-dimensionality of the problem.

2.4 AdaBoost

The grid search suggests an AdaBoost classifier with 77 estimators and a learning rate of 0.1 is optimal. For our base estimator we use a decision tree with min_sample_split = 640 as before, but we set max_depth = 4 for a weaker classifier less prone to overfitting.



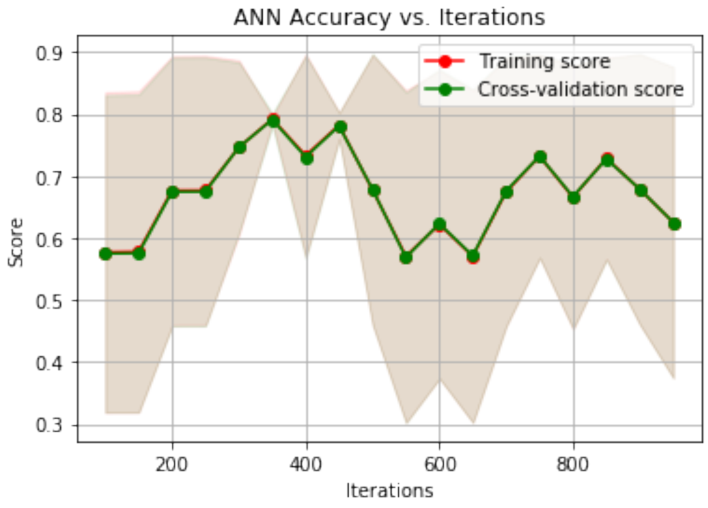
The AdaBoost algorithm seems to reach peak accuracy at 150 estimators. Although it's hard to tell, as the variance of the cross-validation score is high. Rather than stick with the grid search results, I decided to use 150 estimators.



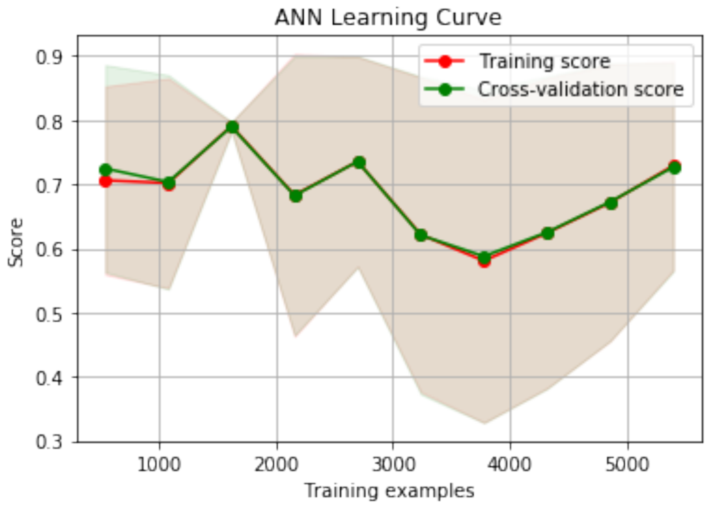
I find it interesting that the training accuracy immediately jumps up to 90% after 1000 or so samples, and then slowly decreases as more samples become available. I suspect this is because a small amount of data is easier to overfit. The cross-validation accuracy continues to grow even past 5000 samples. This is another model that might improve in performance if given the entire training set. In any event, it is slightly more accurate than the decision tree.

2.5 Artificial Neural Network

The grid search suggests a single hidden layer with 90 nodes. Let's see how it trains over iterations.



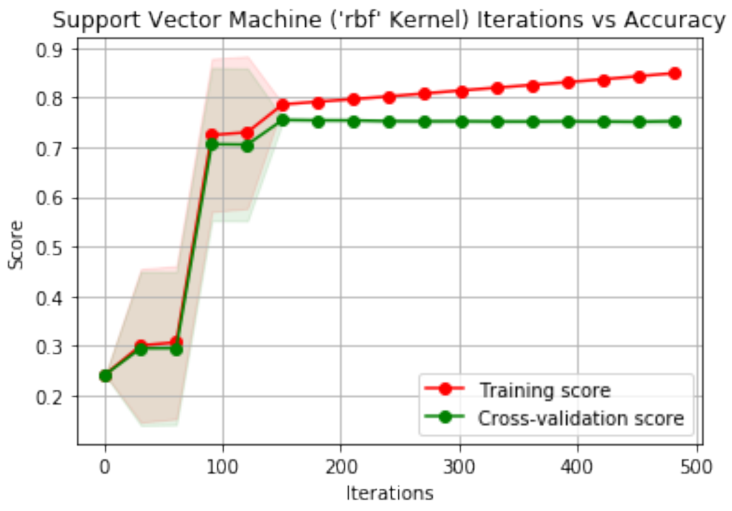
It's puzzling that the training and cross-validation errors perfectly overlap one another. ANN should be able to consistently increase its training score through gradient descent, but that doesn't seem to be the case here.



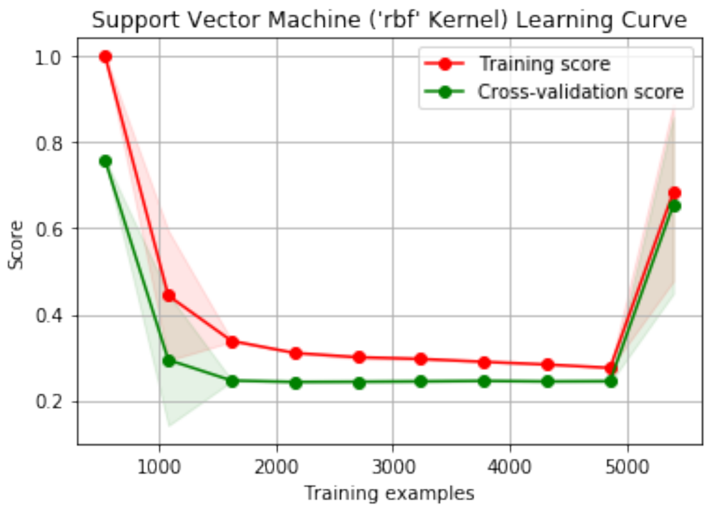
Another puzzling graph with training and cross-validation scores perfectly overlapping. The model achieves about 80% accuracy (with zero variation) after 1500 or so samples, then quickly drops, and slowly rises again.

2.6 Support Vector Machine

A grid search suggests we run our SVM classifier with parameters C = 10, gamma = 0.1, and a radial basis function kernel. Since we were asked to test two kernels, so I also investigated a linear kernel.



At some point this SVM decides to start outputting 0's (income lower than $50,000) 97% of the time. Which explains the huge drop in variation after 150 or so iterations. It is improving its performance on the training set, but those gains don't hold for data it can't see.



The SVM is handling the problem very poorly. The only explanation I can think of for its poor performance is that it is labeling almost everything as a 1 (income greater than $50,000). That would give you about 25% accuracy on the dataset, and no variation (the cross-validation folds are stratified). It eventually pulls out of this rut at 5000 samples. The linear SVM has similar behavior, performing at about 25% accuracy, but never pulling out of the rut.

2.7 Model Scores

Model	Accuracy	Scoring Time	Training Time
AdaBoost	87%	43.9 ms	4745.3 ms
Decision Tree	85.2%	3.5 ms	30.5 ms
Neural Net	79.3%	6.9 ms	392 ms
K Nearest Neighbors	78.8%	47.5 ms	34.5 ms
Support Vector Machine (RBF)	75.3%	405.5 ms	5820.1 ms
Support Vector Machine (Linear)	25.2%	3.5 ms	517.7 ms

On the test set the AdaBoosted decision trees perform best, followed by a single decision tree. The fact that the data is mostly categorical may make it easier for the decision tree to find meaningful features. The model that trains the longest is the RBF kernel SVM which may be represent the difficulty in finding a hyperplane through the data (despite the high dimensionality of the problem). The linear kernel SVM is apparently broken when applied to this data. I suspect that whatever caused the huge decrease in accuracy for the majority of the RBF SVM's learning curve is what's causing this effect.