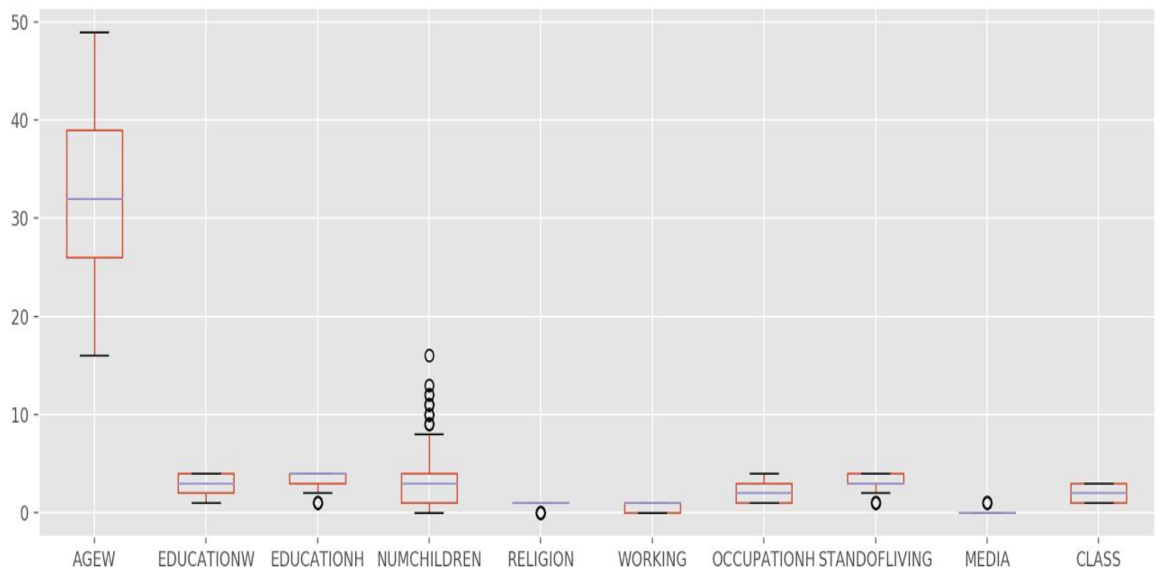
CS 4641 Project 1: Supervised Learning
Samuel Bretz (sbretz3, 903043279)

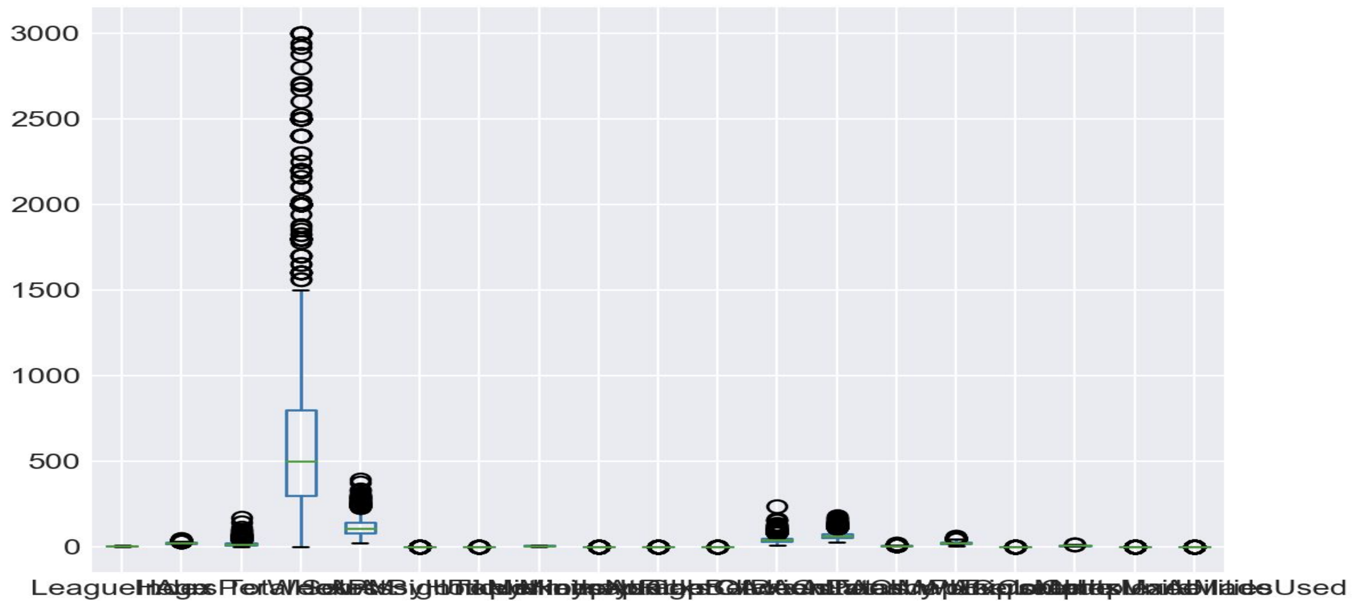
Classification Problems:

The two classification problems I chose are classifying contraceptive method choice for women, and classifying the league that players are in based on their game actions in the video game StarCraft. I think the CMC dataset will be an interesting problem because of the relatively small size of the dataset. I am curious to see if the algorithms are powerful/clever enough to be able to yield good performance with a smaller dataset size. The StarCraft dataset was compelling to use because I have been playing the video game for a long time, and noticed that some of the features that were chosen for the dataset seemed obvious to indicate performance but others did not seem like they would be useful whatsoever, so I want to see if those features end up being useful in the classification problem. Performance was rated using five different algorithms: Decision Tree, Neural Network, Support Vector Machine, Boosting, and K-Nearest Neighbors. I'll discuss each algorithm in several capacities related to their performance on the two datasets I have chosen and then compare and contrast their performance against each other. All models were implemented using the pandas and sklearn libraries in python, and plots generated using matplotlib. Common themes that I will address include but are not limited to: inductive bias, restrictive bias, overfitting, generalization, robustness to outliers, model complexity and computational intensity. Before continuing to the analysis of each algorithm's performance, I'll first discuss the datasets themselves.

Bellow is a Box and Whiskers plot of the CMC (contraceptive method choice) data set which has 10 features and 1474 instances. There are a couple things to note here: first off, the AGEW feature (woman's age) appears to be have a normal distribution, and second that the NUMCHILDREN feature (number of children that the woman has) has several outliers.

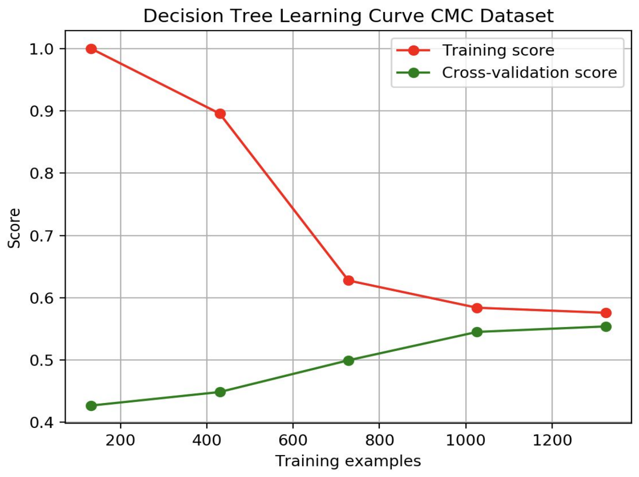


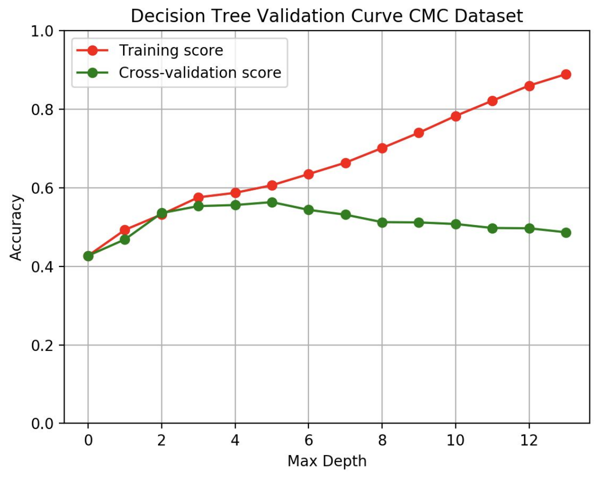
Below is a Box and Whiskers plot for the StarCraft data set, which has 20 features and 3377 instances. Since there are so many features and the text is indiscernible the features are in order from left to right: 'League Index', 'Age', 'HoursPerWeek', 'Total Hours', 'APM', 'Select By Hotkeys', 'Assign To Hotkeys', 'Unique Hotkeys', 'Minimap Attacks', 'MinimapRightClicks', 'NumberOfPACs', 'GapBetweenPACs', 'ActionLatency', 'ActionsInPAC', 'TotalMapExplored', 'WorkersMade', 'UniqueUnitsMade','ComplexUnitsMade', 'ComplexAbilitiesUsed'. It can be seen in the plot that 'Total Hours' has many outliers and otherwise seems to be normally distributed.

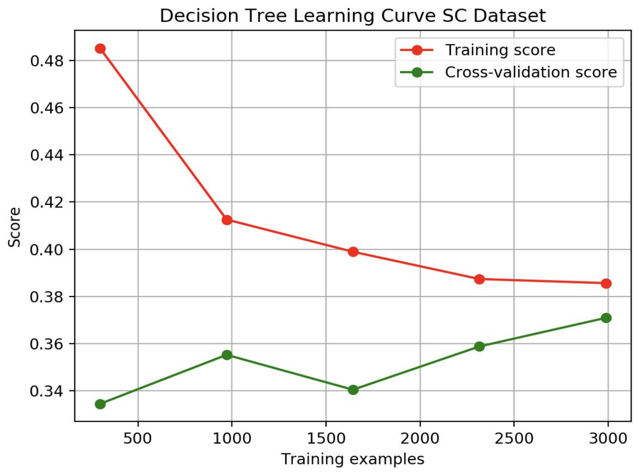


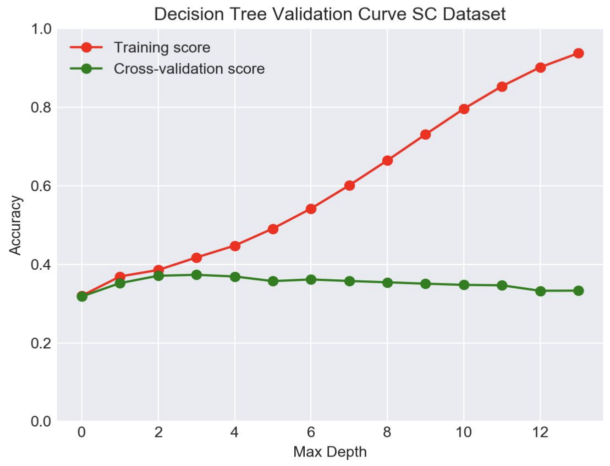
Decision Tree

The Decision Tree algorithm attempts to classify data instances by selecting criterion which split the data "best" (criterion which provide the best information gain). The decision tree itself is made up of nodes connected such that given an instance of the dataset the tree will provide a decision path which ultimately classifies it as belonging to one of the classes in the label set. In terms of bias, the algorithms used to create decision trees (ID3, CART 4.5) prefer smaller trees over larger trees and trees that put higher information gain nodes closer to the root. This is usually due to overfitting; trees that have larger depths have greater expressive power and therefore have a better ability to fit more closely to the training data and or noise within the training data. This can cause the tree to fail to generalize well when classifying test instances, resulting in lower testing scores but higher training scores. Below are learning curve and validation curve plots for the Decision Tree model on the Starcraft and CMC datasets.









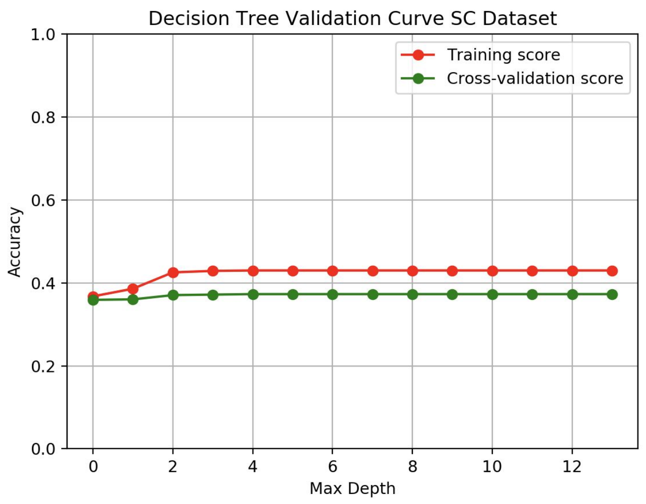
StarCraft Dataset:

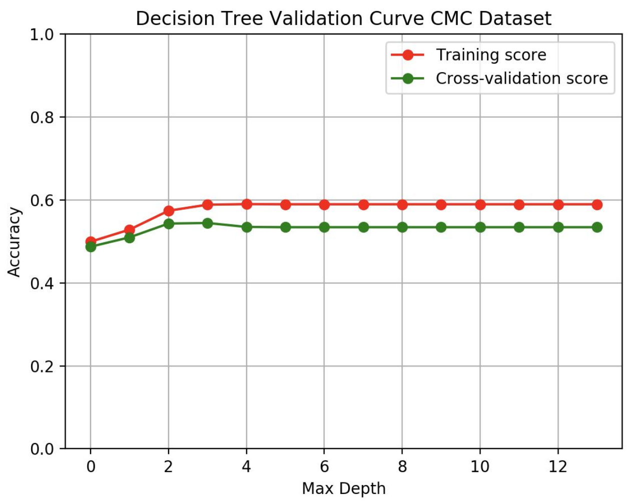
Max Accuracy: 36.72% (2.57x better than random chance), Optimal Tree Depth: 3, Wall Clock Time: 1.976s

CMC Dataset:

Max Accuracy: 54.97% (1.65x better than random chance), Optimal Tree Depth: 5, Wall Clock Time: 1.973s

The plots were generated using scikit learn DecisionTreeClassifier with no pruning and no tuning of hyperparameters (a vanilla decision tree) and 10-fold cross validation. The validation curves were plotted first, then the optimal depth discovered was used to plot the learning curves. In the learning curves, it can be seen that there is initially very high training accuracy relative to testing accuracy. This can be explained by the model fitting very closely to the first few data points. Since there is a small amount of data at first, the tree can essentially memorize the training data. It can also be deduced that the StarCraft dataset is much noisier than the CMC dataset from the fact that the initial training score on the CMC learning curve plot is 1.0, but the same metric on the StarCraft dataset is ~0.49. Hence, even with very few training examples, the decision tree still has a hard time fitting to the training data in the StarCraft dataset and therefore yields a lower initial training accuracy while via the same logic in reverse the CMC dataset should have very low noise. In the validation curves, it can be seen that as the max depth of the decision tree grows larger past a certain point, the testing accuracy begins to fall. This is a classic overfitting scenario; the tree becomes expressive enough to fit too closely to the training data and loses its ability to generalize well to the testing set. The wall clock time is relatively fast for decision trees. Since they classify as eager learners decision trees construct their model first tree via the information gain calculation per node in the tree (which is computationally cheap), then they just have to follow the decision path for new predictions. Since the information gain calculation is the only intensive calculation and it is done quickly, the runtime can be expected to be relatively low compared to other algorithms. In an attempt to combat overfitting, there are pruning techniques that can be applied to decision trees to prevent loss of generalization; they include post-error pruning, weight-based pre-pruning and minimum-sample requirements in leaf nodes. Each of these aims in general to keep the tree from fitting too closely to the training data and forcing the model to remain general, so it can generalize well to the testing set and achieve better predictive accuracy. Below are the same plots as above, except that they use pruning (generated using scikit learn GridCV search to tune hyperparameters). The models are much more robust to overfitting and the curves are much smoother, suggesting that the models are also better at dealing with noise in the data.





Neural Network

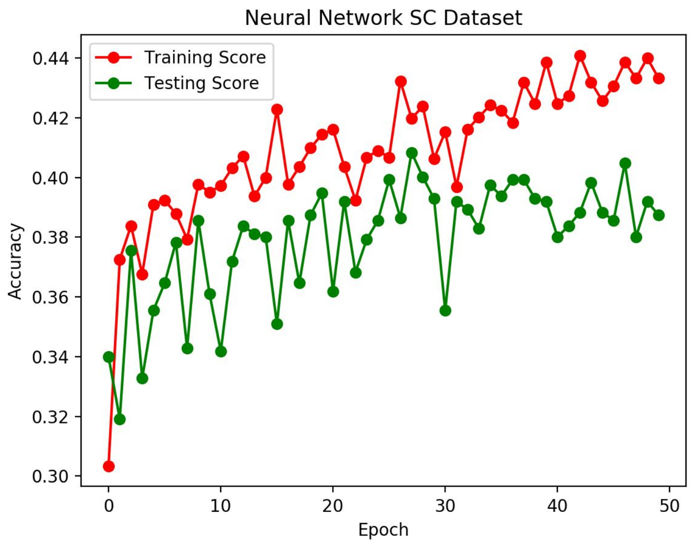
Neural Networks are a network of ‘neurons’ (perceptrons) that are connected to each other and are multiplied by a set of weights that causes them to activate or not activate based on a predetermined threshold. Neural networks are classified as “eager” learners, meaning that the network will attempt to fit a function to the training data that will best classify the testing data. A neural network can be seen as a function approximator since it is essentially a giant function which attempts to optimize a set of coefficients (weights) until the function is optimal and Cross-Entropy loss is minimized. The expressivity of neural networks is extremely high for this reason. Given enough layers a neural network can approximate any function; however therein lies the achilles heel of neural networks. Since training makes use of the iterative backpropagation algorithm to update the weights in each layer it can become extremely computationally intensive and training times are high. The tradeoff is typically high performance if the network is given enough data to train on. Below are plots of the number of epochs used in training against the accuracy of the model. The accuracy with the neural net is higher than other algorithms which is most likely due to the expressivity and complexity of the model. The plots once again confirm the idea that the SC dataset is noisier than the CMC dataset since the CMC curves are much smoother in the plot. The general trend is that as the number of epochs used increases the model accuracy increases; intuitively this makes sense since the backpropagation algorithm is given more attempts to optimize the weight set for predictions. The wall clock time is slower than most of the other algorithms but not the slowest; I expected training to take much longer but since only 50 epochs were needed to reach optimal performance the model trained quickly. The plots were generated each with 7 hidden layers and a softmax activation function as the final layer.

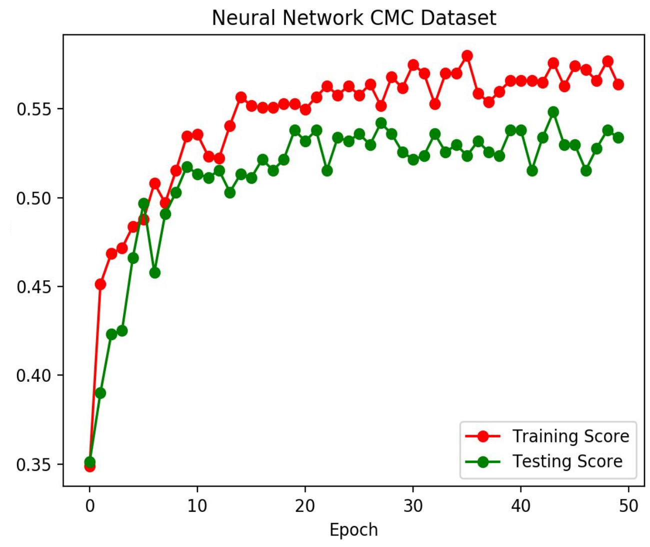
StarCraft Dataset:

Max Accuracy: 40.46 % (2.822x better than random chance), Optimal # Epochs: 40, Wall Clock Time: 12.689s

CMC Dataset:

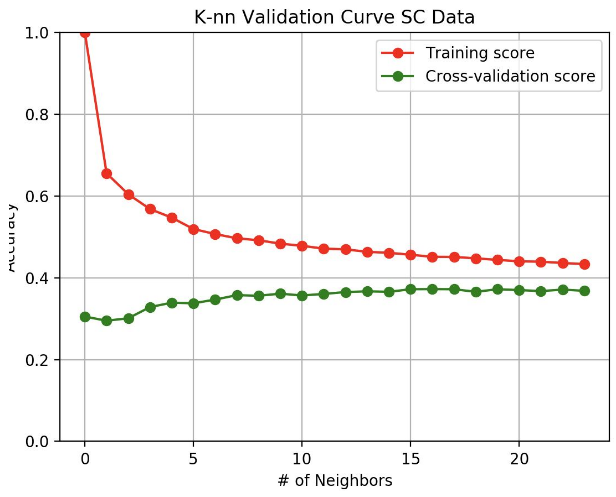
Max Accuracy: 55.59% (1.667x better than random chance), Optimal # Epochs: 32, Wall Clock Time: 22.987s

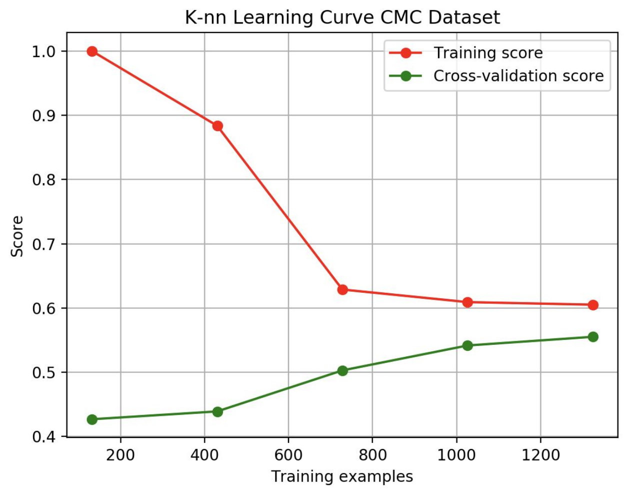


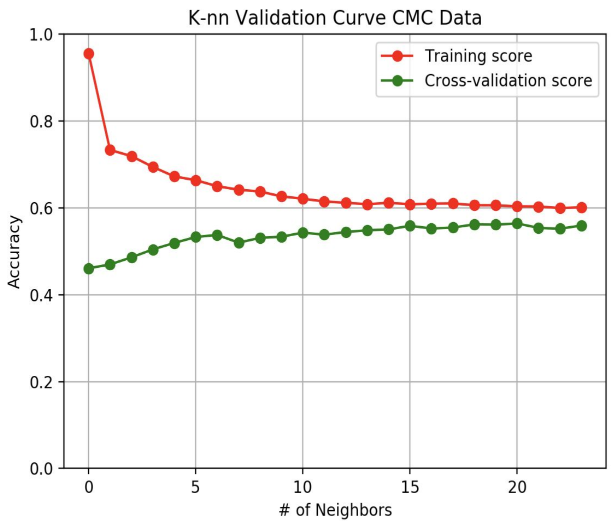


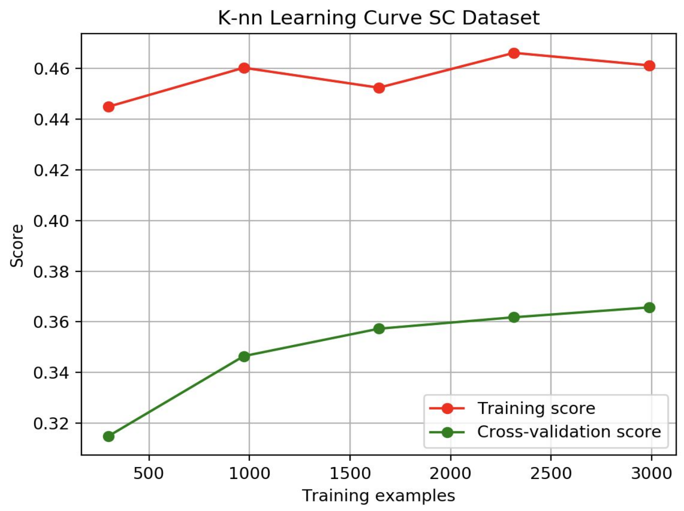
K-nearest Neighbors

The K-nearest Neighbors algorithm finds the k data instances that are closest to the given instance and uses these "neighbors" to predict the class of the given instance (they are often weighted based on how far away they are using varying distance metrics). K-nearest Neighbors falls into the category of instance based learners where instead of learning a model before making predictions like a decision tree or neural network all training instances are stored and then sampled when predictions are made. This feature dictates the behavior of the algorithm in several capacities. One upside is that training time is fast since there isn't a model that needs to be fitted to training data, however runtime can be very slow and computationally expensive since the data must be stored all at once and distance calculations must be made repeatedly (O(md) where m = #examples, d = #dimensions). Another upside is that the algorithm is very simple, there aren't any hyperparameters to tune except for the number of neighbors. The number of neighbors heavily influences variance and bias as well as performance; if the number of neighbors is low, prediction curves will be jagged and fit closely to noise in the data. However if the number of neighbors is too large, the prediction curve will become too vague and won't fit well enough to be accurate. There is a sweet spot in the middle of these two extremes where the prediction curve is not too specific and not too vague where performance excels. Below are validation curves and learning curves for the K-nearest Neighbors algorithm (generated using scikit learn KNN classifier) using the CMC Dataset and StarCraft dataset.









StarCraft Dataset:

Max Accuracy: 36.84% (2.58x better than random chance), Optimal # Neighbors: 15, Wall Clock Time: 6.937s

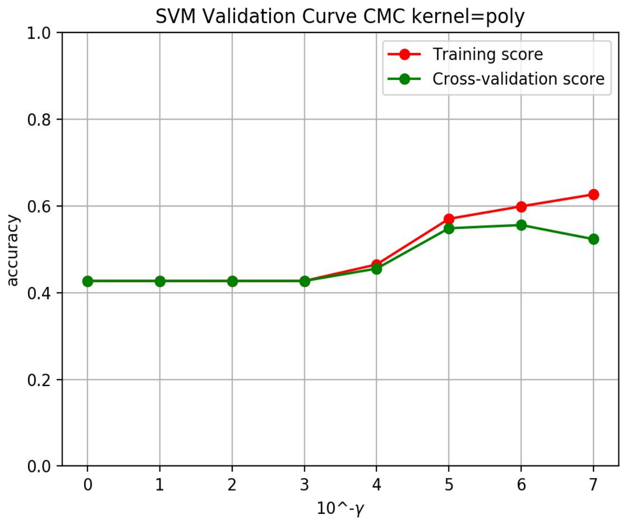
CMC Dataset:

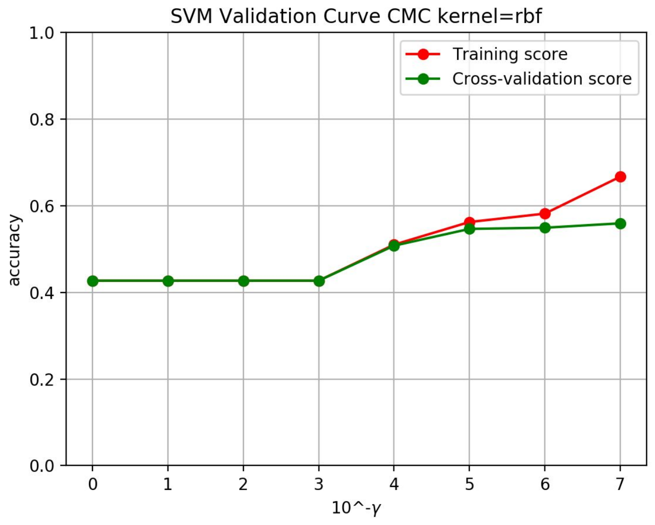
Max Accuracy: 55.79% (1.637x better than random chance), Optimal # Neighbors: 20, Wall Clock Time: 2.800s

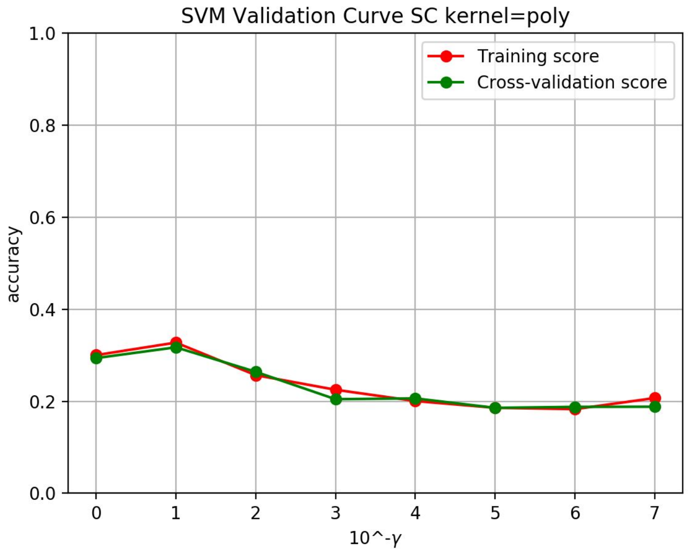
Both datasets' validation curve show 100% accuracy for 1 neighbor since with 1 neighbor, the algorithm will just be querying for a data point that is itself (and classify with 100% accuracy). As the number of neighbors begins to grow the training accuracy comes down and accuracy begins to come up. This is as expected since we will are getting close to a number of neighbors that provides a good middle ground between fitting to closely to noise and generalizing too much. The learning curves were generated using the optimal number of neighbors for each dataset, and confirm that the SC dataset is noisier than the CMC dataset, seeing that the CMC learning curve starts off at 1.0 and the SC learning curve starts at ~.445 and shows some variance throughout the rest of the curve. There is quite a difference in wall clock time between the two datasets with CMC being 2.47 times faster than StarCraft, which makes sense since the StarCraft dataset is roughly 2.5 times larger than the CMC dataset.

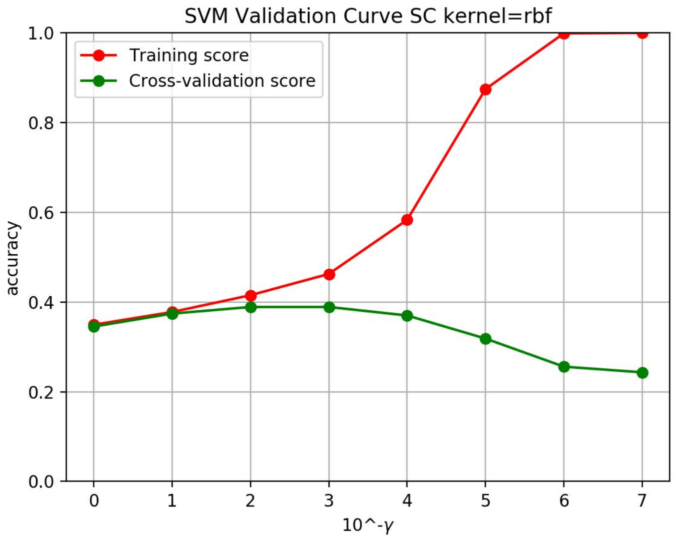
<u>Support Vector Machine</u>

The support vector machine attempts to find the optimal hyperplane for linearly separable patterns of data. The algorithm leverages "kernel functions" for the calculations of these hyperplanes; the most interesting part of SVMs are their ability to transform the input space to a dimension that offers the best margin (the maximum distance separation of points in different classes from a given hyperplane) for hyperplanes that separate the input data. SVMs are resistant to overfitting since they are regularized; there is a measure in place that penalizes "how misclassified" a given data point is. This measure is typically minimizing a loss function called Hinge Loss. By using regularization SVMs do a good job of keeping from fitting too closely to the data. While not impervious to noise and outliers, SVMs do a good job of dealing with them since the regularization takes care of not paying outliers too much mind and data points close to the hyperplane are given much more importance. Below are plots for the CMC and StarCraft datasets with the linear kernel, radial basis function kernel and polynomial kernel plotted with the regularization constant against the accuracy.









StarCraft Dataset:

Max Accuracy: 39.7% (2.78x better than random chance), Optimal Gamma: 10^-7, Wall Clock Time: 20.439s (rbf kernel) 22.138s (poly kernel)

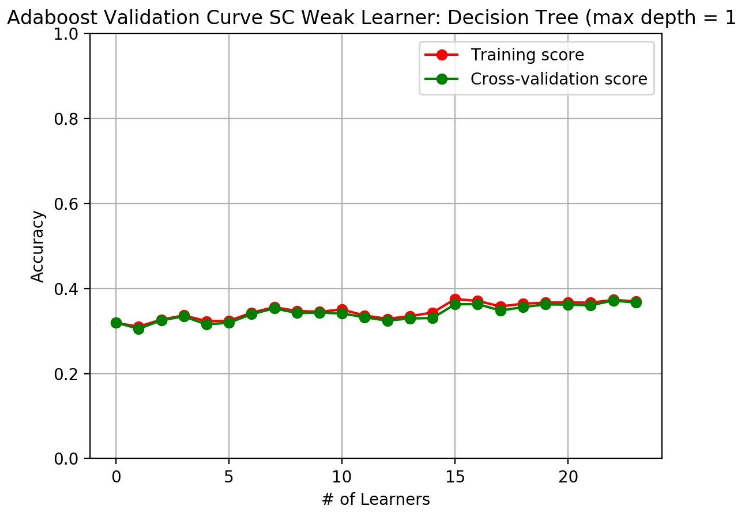
CMC Dataset:

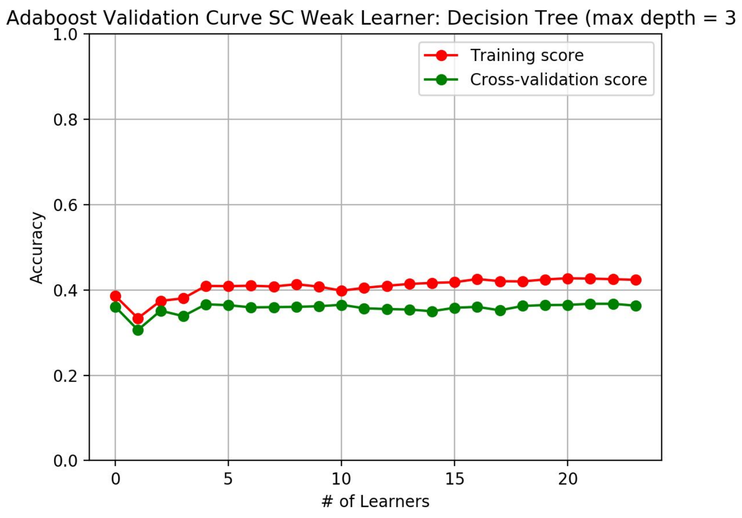
Max Accuracy: 55.83% (1.68x better than random chance), Optimal Gamma: 10^-1, Wall Clock Time: 16.331s

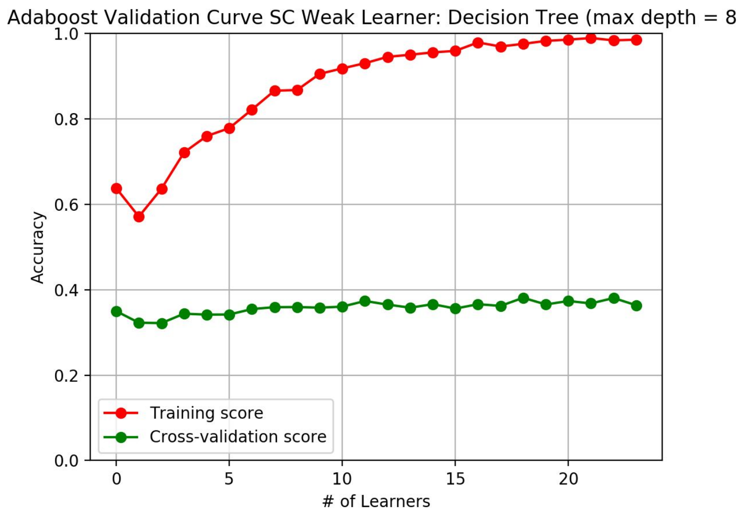
The focus of these plots is the regularization constant. A low regularization constant aims at keeping the decision hyperplane smooth, while a high regularization constant aims at classifying every example correct which creates a decision surface that is much more jagged. It can be seen in the plots that as the regularization constant increases, the model begins to overfit. This occurs because the model complexity increases greatly and the model is allowed to fit very closely to the training data, which hampers generalization greatly. It is noticeable however that the degree to which the models overfit is much less than for other algorithms; decision trees in particular.

Boosting

Boosting was explored using the AdaBoost algorithm, which uses "weak" learners to identify difficult points in the datasets and the combine the weak learners into a "strong" classifier. There are numerous advantages to this method including its speed, simple implementation, versatility since no information about the weak learner is required (it can be any algorithm or model, so long as it qualifies as weak, i.e. it classifies training data at more than 50% accuracy) and it is one of the few algorithms that is provably effective given that the classifiers it is combined from qualify as weak. The downside of AdaBoost and boosting in general that low accuracy and overfitting can occur if the weak classifiers are either too weak or too complex. If they are too complex, overfitting will occur since the overall model is made up of the weak classifiers and if the weak classifiers are complex enough to overfit so will the overall model. On the other hand, if the weak classifiers are too weak (low accuracy scores) the overall model will yield low accuracy scores via the same logic. This is well demonstrated in the plots below.

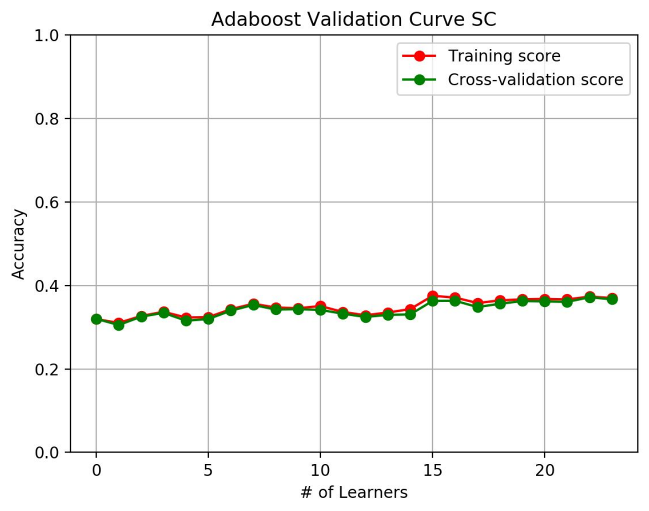


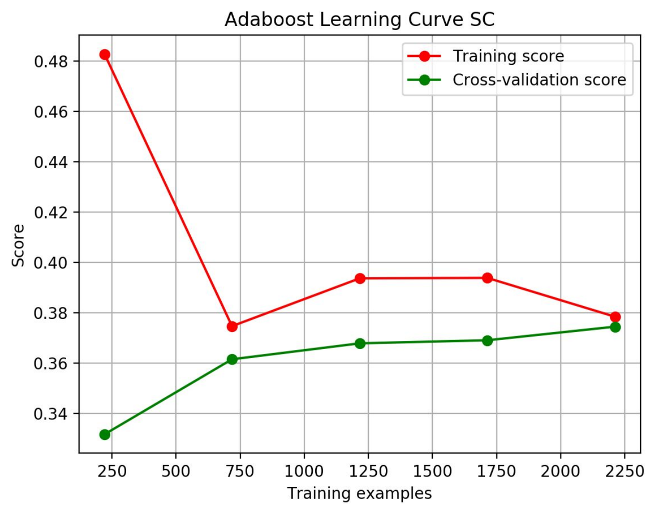


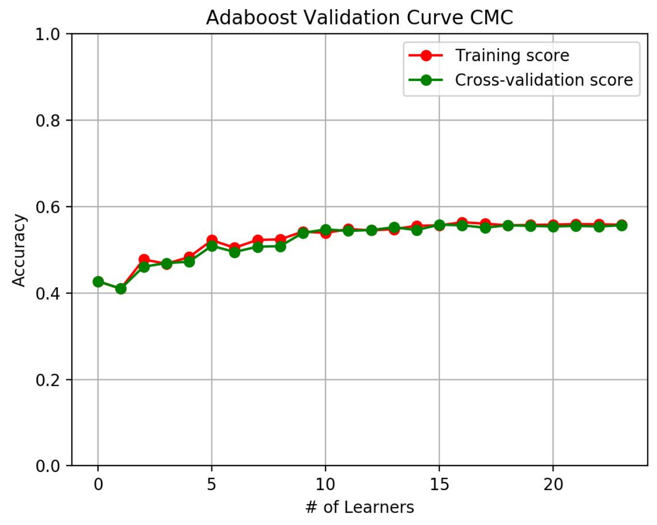


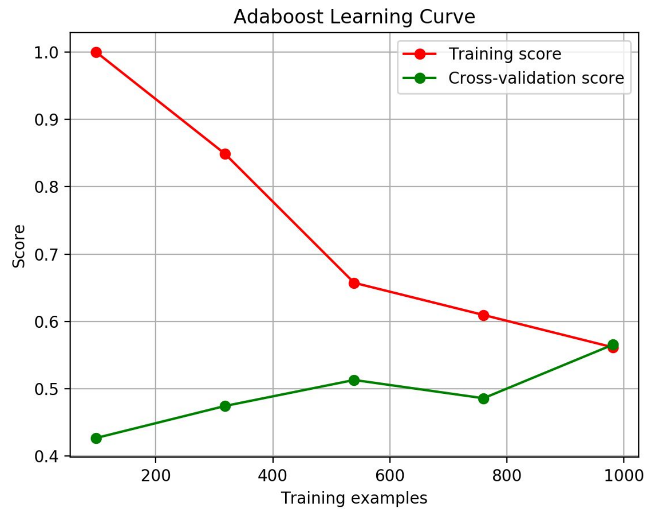
It can be seen in these plots the effect of increasing the complexity of the weak learner. When the max depth of the weak learners is 1 (a decision stump) there is no overfitting because the weak learner is the least complex it can be. When the max depth increases to 3, there is a small amount of overfitting, but when the max depth goes up to 8 the training score can be seen to shoot up drastically with no increase in testing score.

Below are the validation plots for the CMC Dataset and StarCraft dataset generated with the AdaBoost Classifier in scikit learn.









StarCraft Dataset:

Max Accuracy: 38.44% (2.69x better than random chance), # Learners: 1, Wall Clock Time: 6.937s

CMC Dataset:

Max Accuracy: 55.79% (1.68x better than random chance), # Learners: 1, Wall Clock Time: 16.331s

Conclusion

The choice of what algorithm is “best” is difficult. Each algorithm is useful in different aspects and has different focuses. Something that I have noticed is that the raw accuracy level across all algorithms for both datasets is relatively similar (+/- 4%) which honestly seems somewhat troubling even though this is part of why I thought the problems might be interesting. My first thought is that there just isn’t enough data in these datasets to achieve high accuracy levels; the CMC dataset has 1474 instances and the StarCraft dataset has 3377 instances, with 10 and 20 features respectively. Obviously more data is always helpful for anything in machine learning but I think that more data would boost the accuracy levels for all of the algorithms. Since there wasn’t a large difference in accuracy levels across algorithms the criterion for which algorithm is best changes. In my view, the decision tree algorithm is the best algorithm for several reasons. First off from the viewpoint of Occam’s Razor the decision tree algorithm is very simple to implement and works on a very simple idea. It is also immensely expressive. The only other algorithm with higher possible model complexity is most likely a neural network. The balance between simplicity and complexity in the decision tree model is incredibly useful because it allows for easy recognition of overfitting. The decision tree also has the best wall clock runtime (1.97s!) of any of the algorithms on these two datasets. This may be skewed somewhat since the datasets were somewhat small but by making a general assumption that runtimes for each algorithm would scale according to dataset size, decision tree still looks to be the fastest. Decision tree was also the easiest of these algorithms to implement and use. The parameters for hypertuning were very intuitive as to how they affected the model and the outcomes. For these reasons I’ll say that the decision tree was the best algorithm out of the group.