# Supervised Learning

Wesley Tomjack

wtomjack3@gatech.edu

## Abstract:

In the upcoming pages, detailed descriptions and comparisons of differing supervised learning algorithms will be implemented to offer a glimpse into and hopefully expand the understanding of how algorithms interact with the provided data. The classification approach used with these learners will demonstrate how each models the provided data sets (when applicable) to generate a hypothesis, that can be queried later where the results will later be shown. Or in the case of our clustering approach (KNN), can be compared to it's k-neighbors to determine results and subsequently the accuracy of those results.

## Explanation of Data Sets:

- **Wine Quality Data Set:** This data set was provided by the UCI Machine Learning Repository, as is a collection of rated wines from the north of Portugal. It is a multivariate dataset with 12 attributes, 11 of which are used as the features for these supervised learning models. These range from fixed acidity, to density of the wine, all the way to the alcohol content, etc. The 12$^{th}$ attribute in the data set, is the classification label as it was the "quality rating (1-10)" of the wine.
- **Heart Disease Data Set:** Similar to the above data set, this one was also found on the UCI Machine Learning Repository. The data provided in this set had 14 attributes with in, 13 of which were the features, and the final being the classification label. Each of the attributes were measurements taken from patients out of the Cleveland Clinic Foundation. Features of the set included such measurements as Age, Sex, Cholesterol Levels, etc. The label is the severity of heart disease present within the patients that were measured. 0 being the absence of heart disease, and 1-4 being the level of severity if present within the patient. In this experiment, we removed the severity rating of the disease and converted those values into a 1, so that it became a binary classification problem. Where there is either the complete absence (0), or some presence (1) of the illness.
- **Testing vs Training Data:** A random sample of about 30 percent of both data sets was taken using the function train_test_split from sci-kit learn library to give us our test dataset, while the remaining 70 percent of the data was used in training.

The data sets, and classification problems to match were chosen simply because there were a myriad of differences between them, thus making them rather interesting to study. Both data sets were on the smaller size of samples simply because computing time necessary to classify extremely large sets. But the Wine data set had 5 fold the number of examples as the heart data set on which to train. This gave the opportunity to show how certain learning algorithms are more prone to overfitting, while heart data set was able to display how learning algorithms might underfit.

Another difference between the data sets is that the Wine data set was actually quite noisy in that the labels (ratings) had a possible 1-10. But most actually were between 4-7, while the feature values for each instance were extremely wide ranging, making accurate classifications difficult. The heart data set on the other hand was very evenly distributed data, and since it was turned into a binary classification problem (either having heart disease or not) the expectations were that it would be able to classify instances at a much higher rate.

# Analysis:

The algorithms below were used on both data sets previously mentioned, each of which were implemented by Sci-kit Learn, a python based library that features many machine learning algorithms and measurement functions to go along with them. Initially input parameters were self-tuned when generating the learners in order to have a better understanding of how they interacted within each data set. Later on hyper-parameterization through an exhaustive grid search (Appendix. i) was performed on each of the algorithms and training data used to help determine the optimal parameters to be used with them.

## Decision Trees:

Decision trees are learning algorithms that take in a set of data and try to model it in way to find a target concept iterating through the instances and splitting at certain features it deems most relevant in finding the concept. The way in which the algorithm chooses the particular feature to split on is based on a particular function, whether that be gini impurity, or entropy (information gain). In the case of the Sci-Kit Learn library used in this experiment, the default criterion used for the classifier was 'gini' (See Appendix. ii). Even if the gini function wasn't the default it would have been ideal for at least one of the data sets, "Heart Disease", because the data within was quite evenly spread (Raschka. 2015.).

Often times to prevent the overfitting of decision trees, there is a technique called pruning that takes place where tree nodes are removed based on the power of that particular node to classify instances. Pruning can take place pre or post model training. In the case of the library we chose there is no such explicit function to prune a tree learner, and in lieu of writing a stand alone pruning function it was possible to accomplish similar results by setting the minimum samples split parameter passed in to the learner as well as the maximum depth of the tree. The first parameter generated a requirement within the learner that ensured it would not split at an internal node until the number of samples within the node have been met. The latter parameter meant that the decision tree could not split past a certain depth, and that once hit those were the leaf. Both of these parameters in conjunction acted as pruning and allowed us to try and avoid overfitting of our learner.

**Self Tuning Using Validation Curve:**

For the self tuned parameters, the decision was made to alter both maximum tree depth, as well as minimum samples required for a split. A range for both parameters was set between 1 and 30, and cross validation on the training data was applied with 3 folds. Below in Figure 1.1 you can see that for our 1$^{st}$ data set optimal tree depth was at around 8 nodes, between our training score as well as the cross validation score of the training data. Minimum sample split actually did quite poorly when classifying the cross validation data, therefore that parameter was set at 3 on the first data set learner.

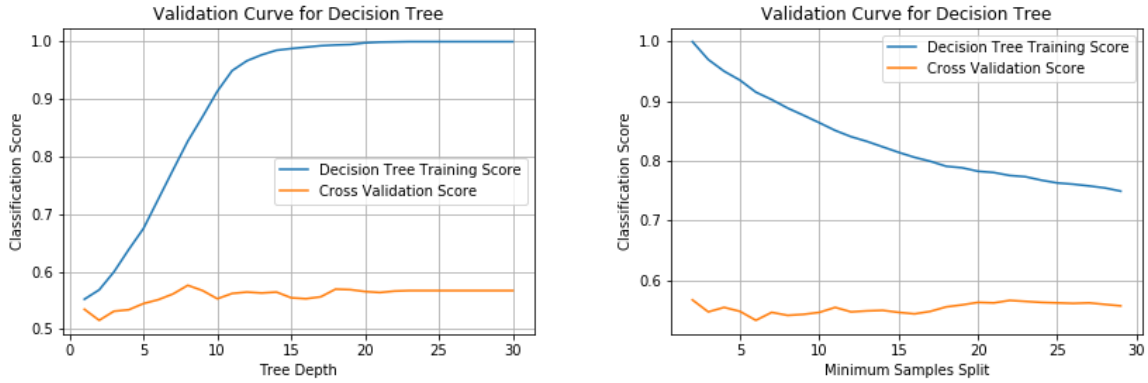



***Figure 1.1: The wine data set shows the validation curve for both maximum tree depth and min sample split.***

As can be seen in Figure 1.2 for the second data set, the training data classified much better when depth was around 10, but that when cross validation was performed the more appropriate tree depth was around 3. This could lead us to believe that adding any more layers of depth to the tree after that produced overfitting. As for the minimum samples split the ideal parameter value was at 16.

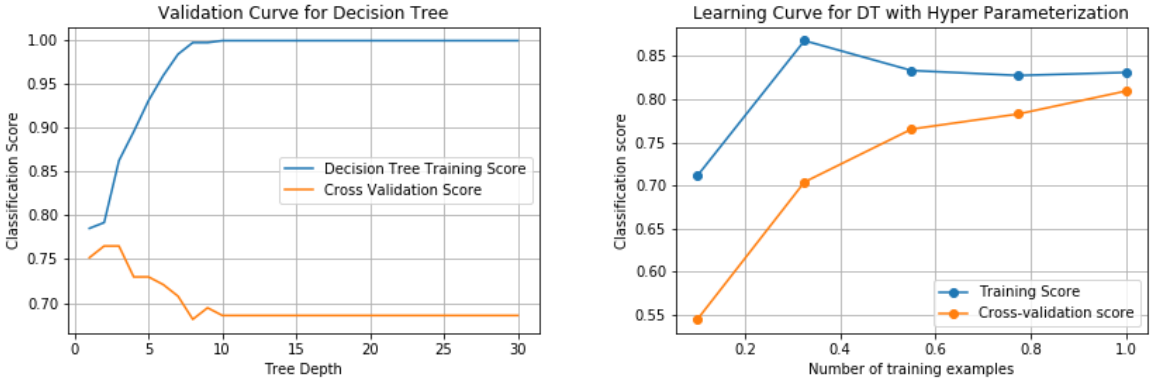



***Figure 1.2: The heart data set shows the validation curve for both maximum tree depth and min sample split.***

**Hyper-parameterization:** When computing an exhaustive grid search of decision tree parameters we provided search values for following parameters, max_depth ([1,…,50]), min_samples_split ([2,…,50]), criterion (gini, entropy) and splitter (best, random). The most optimal model for the supplied parameters for our data sets are shown below respectively:

- **Wine Rating:** {'min_samples_split': 2, 'criterion': 'entropy', 'max_depth': 18}
- **Heart Disease:** {'min_samples_split': 18, 'criterion': 'gini', 'max_depth': 3}

**Learning Curve:**

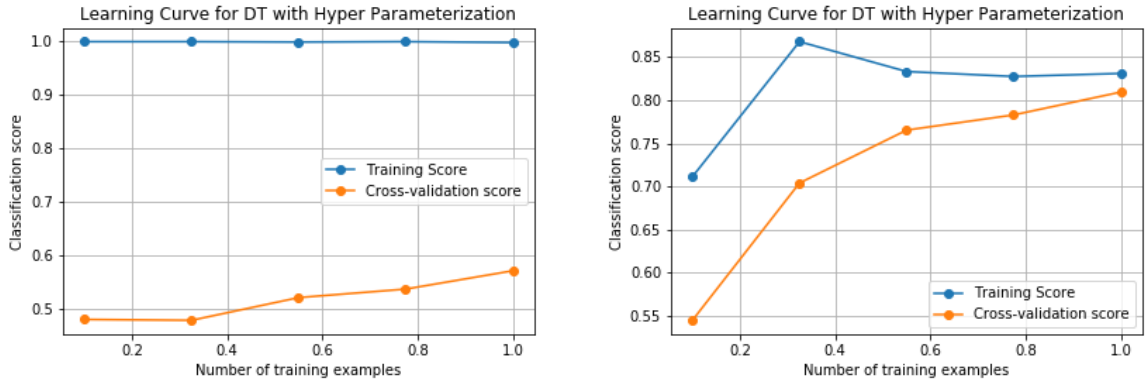



***Figure 1.3: Learning curve for both Wine (Left) and Heart (Right) data sets based on fraction of training examples used***

For Decision Trees we can see that as the amount of training examples used when training the model increases that it tends to classify the training data more accurately during cross validation. That is especially true of the heart data set, the accuracy percentage grows quickly until about half the data is used for training. This is expected since it being such a small data set, training less than half of the samples makes for an extremely underfitted model.

## K-Nearest Neighbor

K-Nearest neighbor models is an instance based learning approach in which a data set is mapped, and based on particular parameters queried against, in an effort to classify the point using a "majority vote" of it's neighbors. Parameters that can be manipulated against the algorithm, include the weight of each neighbors votes, the number of points in the neighborhood, and even how the distance is calculated between neighbors.

**Self Tuning Using Validation Curve:**

As shown below for both data sets when computing neighbor distance, each learner performed slightly better using Euclidean rather than Manhattan. This is believed to be the case due to the fact that both data sets had a relatively low number of features, where low dimensionality has been shown to work best with Euclidean.

In regards to the number of neighbors to be used when generating a neighborhood for a particular query, it can be seen that on both data sets the classification score peaked at around k= 10-13 range. There was less deviation within the classification score during cross validation of different k neighbor values for the wine data set, in comparison to the heart data. And this can be attributed to the aforementioned fact that the data within the heart data set was spread very evenly throughout the feature space, so the opportunity of having an instance or couple of instances that would ultimately misclassify the answer was higher.

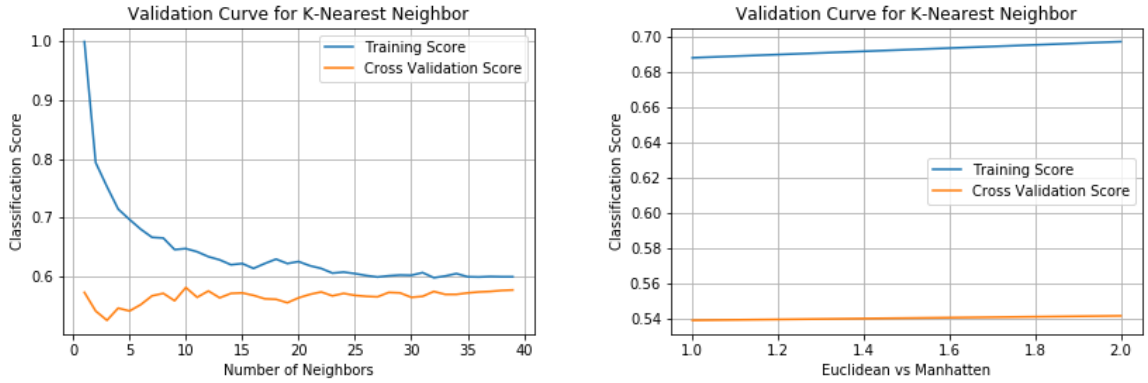



***Figure 2.1: The wine data set shows the classification performance against training data as well as the cross validation sets generated, for both number of neighbors as well as Euclidean vs Manhattan distance.***

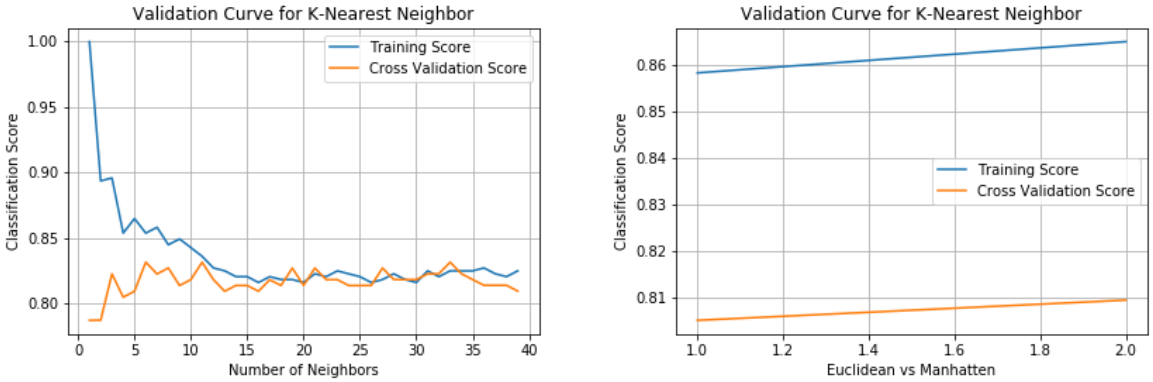



***Figure 2.2: The heart data set shows the classification performance against training data as well as the cross validation sets generated, for both number of neighbors as well as Euclidean vs Manhattan distance.***

**Hyper-parameterization:** The grid search performed on the K-Nearest Neighbor learners, interpreted the best values of the following parameters and returned these in the best estimator. For number of neighbors in the neighborhood (n_neighbors), it was given a range between 1 and 40. The power parameter (p) is given two values and is the distance function applied to neighbors, p = 1 which was interpreted to be Manhattan Distance, and p =2 being Euclidean Distance. Weights parameter (weights) was passed in as the options ‘uniform’, and ‘distance’, where uniform weighted each point in the neighborhood equally and distance meant that closer points in the neighborhood influenced the queried response more heavily.

- **Wine Rating**: {'n_neighbors': 33, 'weights': 'distance', 'p': 1}
- **Heart Disease:** {'n_neighbors': 14, 'weights': 'distance', 'p': 1}

**Learning Curve:**

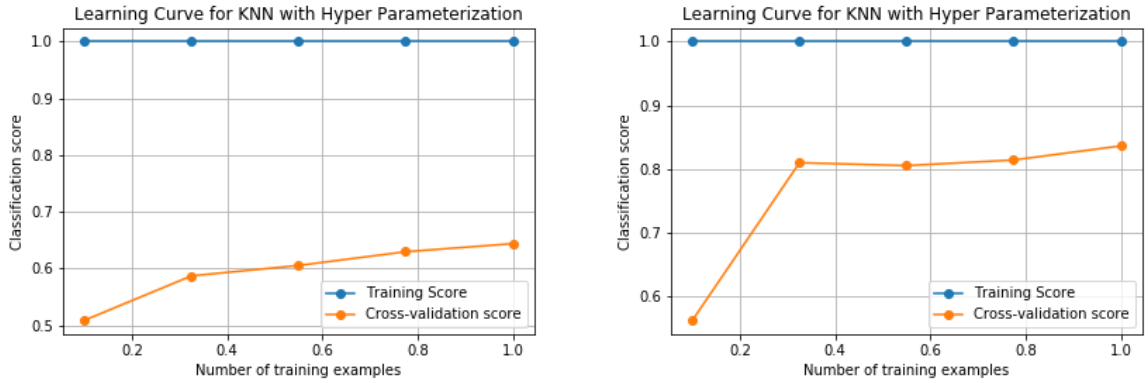



***Figure 2.3: Learning Curve for Wine data set (Left), and Heart Data Set (Right)***

## ADA Boost (with Decision Stumps)

Decision trees are often able to more accurately classify when given multiple many are used together and an aggregate of their training is used as the concept. That is what AdaBoost does, taking in Decision Trees that are only split allows it to have many hundreds of weak learners each growing the weights of incorrectly/difficult to classify instances.

**Self Tuning Using Validation Curve:**

Seeing as learning rate is directly competing with the number of estimators it is hard to truly tell using validation curves the appropriate rate without tuning against multiple different estimator numbers. As learning rate increases the contribution of each estimator decreases, so for the default value where n_estimators is 200. Below in Figure 3.1 the learning rate for Wine Data set was optimal at .001, and .1 for the heart data suggesting that for 200 estimators the need for the contribution from each was larger. Especially in the wine classification.


***Figure 3.1: Learning Rate Validation Curves for both Wine Data (Left) and Heart Data (Right)***

Increasing the estimators worked in improving classification scores up to a point (200 for Wine Rating, 300 for Heart Disease). The reasons that more estimators didn't improve learners trained on the wine

data, is because of the limited features and the noise of the data. While having more features and evenly distributed data proved beneficial to adding more estimators in the case of heart data.

**Hyper-parameterization:**

The number of estimators during AdaBoost was used during hyper-paramterization search as well as the learning rate in order to establish an optimal model.

**Wine Data:** {'n_estimators': 200, 'learning_rate': 1}

**Heart Data:** {'n_estimators': 300, 'learning_rate': 1}

**Learning Curve:**

Observed in the below learning curves, the Adaboost classifier performed marginally better as the size of data increased. Due to the smaller amounts of data it can be assumed that much more data would be needed in order to really see a noticeable difference among the fraction of data samples trained on.

As seen it performed poorly against cross validated data set 1, due to the amount of noise within, and ultimately classified almost 25% more accurately for the heart disease data set. This shows that noise was more detrimental to this particular learner, opposed to not having as much data to train on.

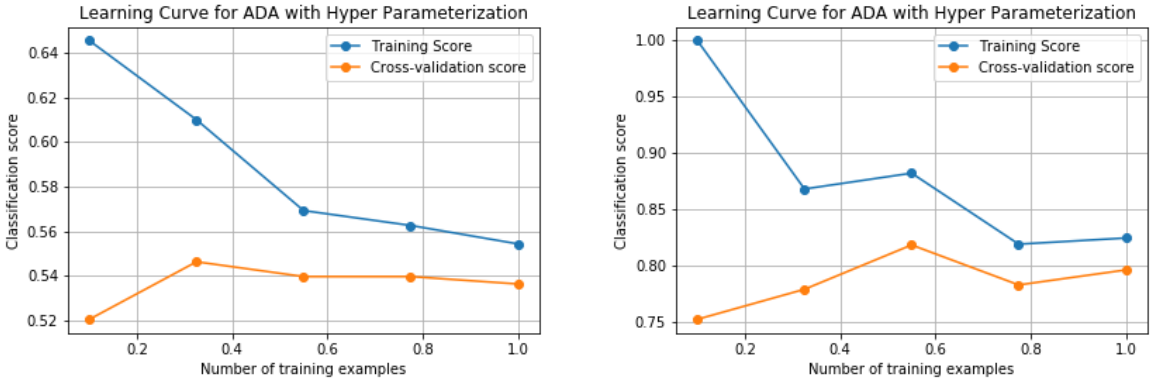


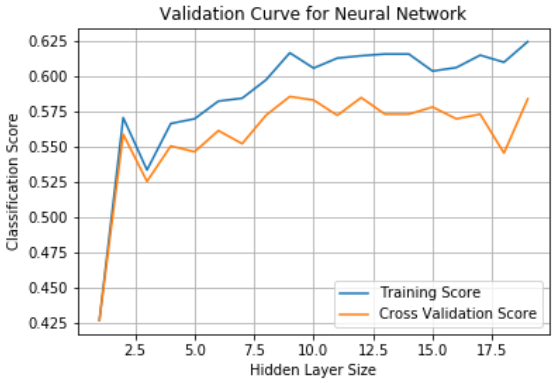

***Figure 3.3: Learning Curve for Wine data set (Left), and Heart Data Set (Right)***

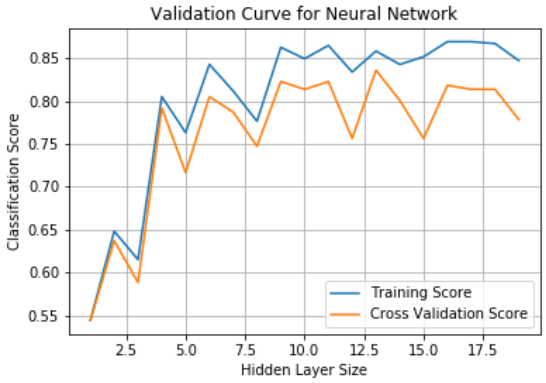
## Artificial Neural Networks

A multilayer perceptron classifier, is a neural network learning algorithm that attempts to classify new instances based on a target concept that was generated using a system of hidden network nodes that uses features of the provided training data.

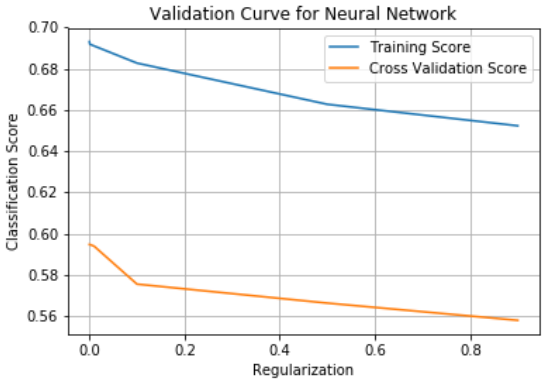
**Self Tuning Using Validation Curve:**

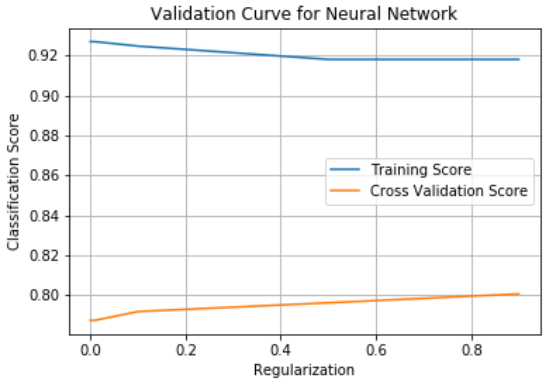
Self tuning the hidden layer size, allowed the determination that for the first data set 9 nodes within each hidden layer was optimal for classification and that for the second data set 13 nodes returned the highest classification score against cross validation.




Different regularization (penalty) parameter values were also tuned on both data sets and it was found that a very low rate of .01 worked best for the wine data set, because of the noisy data with in it. While a larger regularization parameter worked better for data set 2 (heart)




**Hyper-parameterization:** Grid search was performed on each of the following parameters for the multi-perceptron classifier, those being number of nodes within each hidden layer, the alpha or regularization value, and different types of activation functions (tanh, relu, or identity).

- **Wine Data Set:** {'alpha': 0.001, 'activation': 'tanh', 'hidden_layer_sizes': 19}
- **Heart Data Set:** {'alpha': 1, 'activation': 'tanh', 'hidden_layer_sizes': 6}

'tanh' is returned as the optimal activation function for both data sets because the correlation between certain features and the resulting label are prominent. The tanh function being sigmoidal between -1 and 1 allows for negative instances to be weighted much more strongly, thus improving the overall classification (Sharma. 2019).

**Learning Curve:**

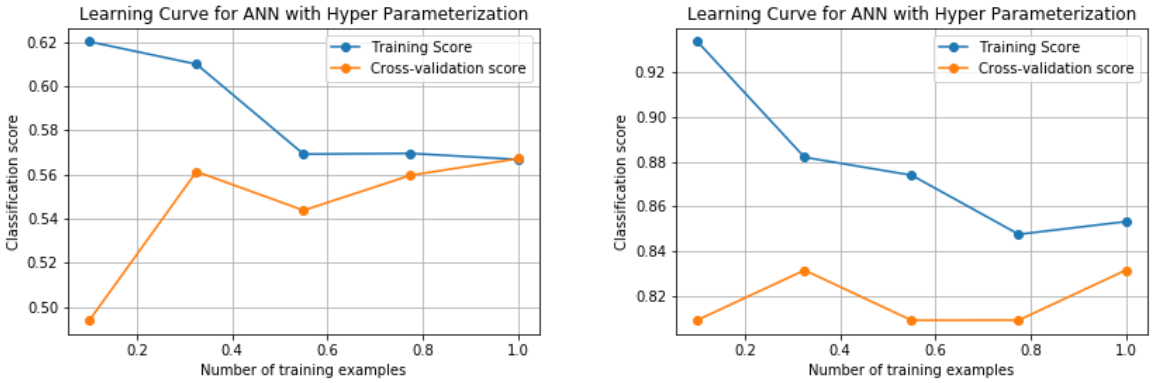



***Figure 4.3: Learning Curve for Wine data set (Left), and Heart Data Set (Right)***

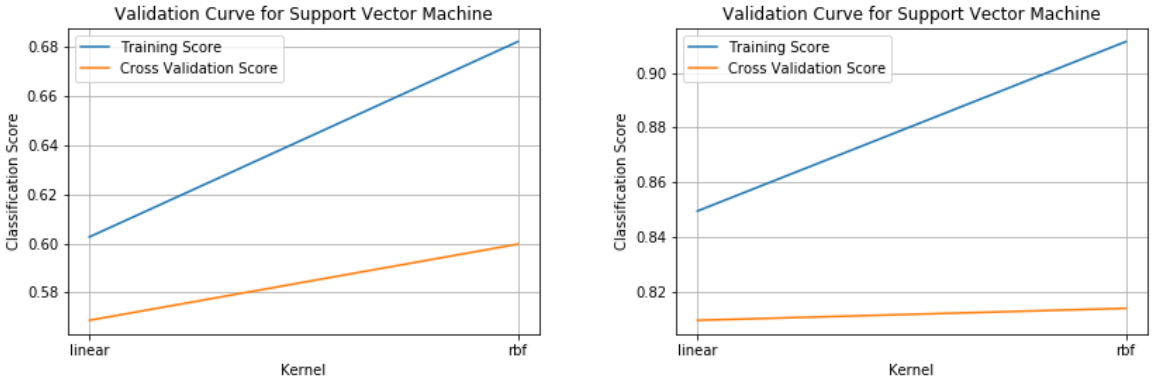
The multilayer perceptron learner, performed quite well on cross validated data when trained against the full training set for the heart data, meaning adding more layers may not help improve the score, rather having more data on which it could be trained would help.

## Support Vector Machine

Essentially SVM's are algorithms that create a hyperplane decision boundary surrounding similar data instances, using a particular learning function called a kernel to deduce similarity between the data samples within the feature space.

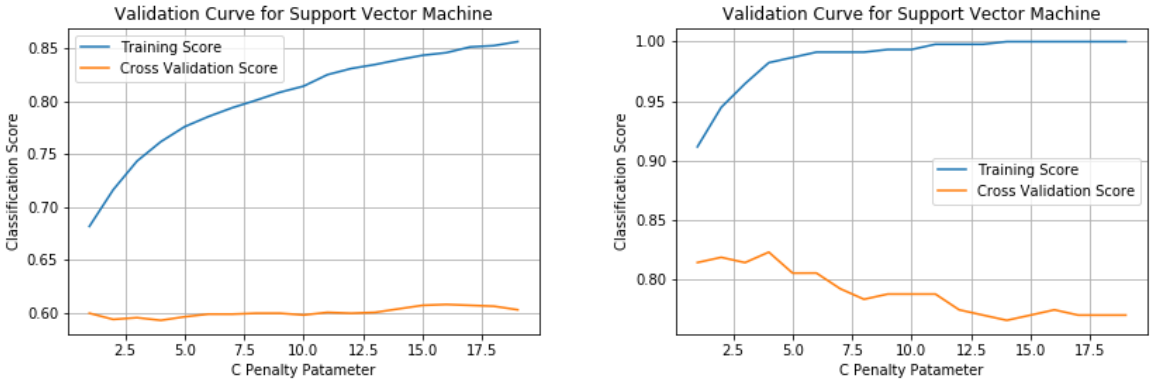
**Self Tuning Using Validation Curve:**

Within the SVM it accepted parameters to alter the kernel function that is used to determine similarities, between data. In the experiment the SVM was supplied the linear kernel as well as the radial basis function kernel ('rbf') to train on and compare it's classification score on cross validated data. As can be seen the rbf kernel performed better on both the wine data and the heart disease set. The data sets were not linearly separable, therefore it's expected that having a rbf kernel would outperform the linear. As for why the classification results don't show a larger difference when comparing both models is due to the fact that there wasn't high enough amounts of training data, causing the underfitting of both.




Increasing the penalty parameter 'C' of the support vector machine up to different allowed the learners to classify better. The larger the value of C the smaller the margins for the decision boundary, which is

why the optimal value for C in the first data set below was 17 because of the amount of noisy data with in the wine data set. It also explains why setting C lower on data set 2 was more successful at classifying, the wider decision boundary margin allowed for a more optimal function to be generated.




**Hyper-parameterization:** Performing grid search on the support vector classifier, produced similar results to the self-tuning.

- **Wine Data Set:** {'kernel': 'rbf', 'C': 16}
- **Heart Data Set:** {'kernel': 'rbf', 'C': 4}

**Learning Curve:**

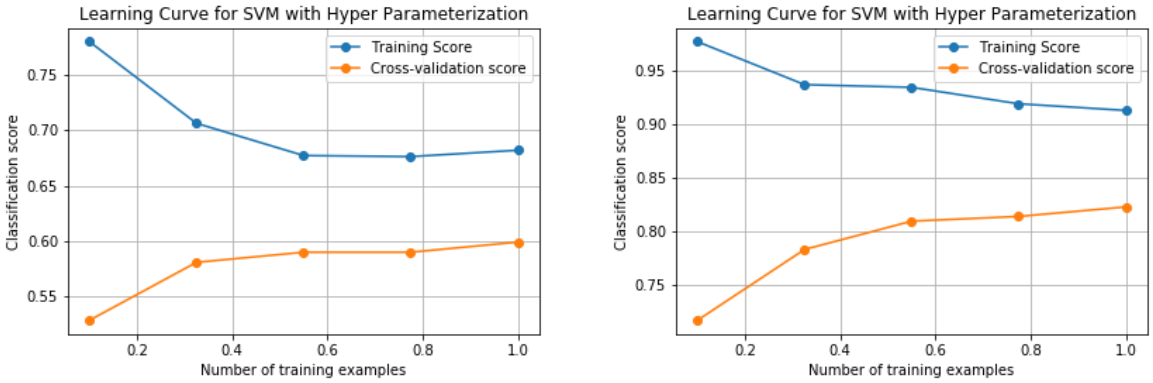



***Figure 5.3: Learning Curve for Wine data set (Left), and Heart Data Set (Right)***

As expected SVM's didn't need much data trained on it to accurately classify instances. After training on about 30 percent of the samples available was seemingly sufficient as the learning curve tapered off and didn't improve drastically with more data.

## Classification Comparative Analysis

Ultimately based on how noisy the data was on the Wine Rating data set, as well as the added complexity of it being a multi-class classification problem, it was no surprise that the learners for the heart disease (binary classification) all vastly outperformed in terms of accuracy.

K-Nearest Neighbor learner did better on the wine data set than the other algorithms specifically because it didn't have to generate a concept with the noisy data and could classify queried instances accurately, approximately 8 percent better than any other learner on that data set.

The artificial neural network learner support vector classifier, and boosted decision tree performed the best on the heart data set. With ADABoost algorithm hitting classification of about 91 percent, this is expected to be the case due to the evenly distributed data, and the ability of the weak learners within the classifier to find the correct weights of each feature as such.

Support vector classifier performed overall the best over both problems. It was in the top tier of classification accuracy percentage for both data sets compared to the others, and was able to perform with the reasonably small data set (Wine: 1600 Instances) and the very small data set (Heart : 300 Instances). It was also impressive in terms of training and query time to go along with its higher classification score. When scaled support vector machine would appear to be a learning algorithm that would be able to give results quickly and accurately.

As for time taken to train (Figure 6.2) as expected KNN was extremely fast as it doesn't try and generate any concept, all of it's work is in query time where it tries to classify each instance. Decision tree was also quick in training because of the small number of features used during splitting. The longest training time by far was that of Artificial Neural Networks, due to the complexity it takes and the amount of nodes in each hidden layer.

Query times (Figure 6.3) were all exceptionally low, even KNN which can be explained because of the size of the data sets. Calculating all the distances and neighbors was negligible time with such limited data.

## Results

### Accuracy Results

| | SVM | KNN | ANN | DT | ADA |
|---|---|---|---|---|---|
| Wine Rating Accuracy Percentage | 62.50% | 67.25% | 59.50% | 61.50% | 56.50% |
| Heart Disease Accuracy Percentage | 81.58% | 75.53% | 86.84% | 80.26% | 90.79% |

***Figure 6.1***

### Training Time

| | SVM | KNN | ANN | DT | ADA |
|---|---|---|---|---|---|
| Wine Training Time | 5.57 | 2.3 | 31.5 | 2.4 | 6.3 |
| Heart Training Time | 1.4 | 0.78 | 8.32 | 0.93 | 1.74 |

***Figure 6.2***

### Query Time

| | SVM | KNN | ANN | DT | ADA |
|---|---|---|---|---|---|
| Wine Query Time (Seconds) | 0.014 | 0.04 | 0.002 | 0 | 0 |
| Heart Query Time (Seconds) | 0.014 | 0.02 | 0.0009 | 0 | 0.001 |

***Figure 6.3***

## Appendix

i) **Grid Search**: Grid search is a technique used when tuning parameters, to return the optimal values for your particular learner. A grid is provided to the algorithm, which is comprised of different values for specific parameters. For example the number of neighbors parameter in a K Nearest Neighbor algorithm, could be provided to grid search as [0,1,2,3,4,5..,n]. Grid search will construct several versions of KNN with the provided parameters, and in the case of exhaustive grid search will create all combinations of parameters and where each model can be represented as a point on a grid. Each model is then tested against the training data provided using cross validations (kFolds the variable used in our experiment). The model that returns the lowest mean squared error is the one that is chosen as the "best estimator" and those corresponding parameters are the ones used when further evaluating data (Bergstra. 2012.). Seeing as it exhaust all parameter combinations this is very expensive computationally.

ii) **Gini Impurity**: Per the book Python Machine Learning written by Sebastian Raschka, the gini impurity function "can be understood as a criterion to minimize the probability of misclassification." Often times it yields results very closely related to that of information gain, but performs better if data sets are perfectly mixed (Raschka. 2015.).

## Bibliography


Bergstra, J., & Bengio, Y. (2012). *Random Search for Hyper-Parameter Optimization*. 25. http://jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Grid Search—Model Evaluation. (n.d.). Retrieved September 21, 2019, from Coursera website: https://www.coursera.org/lecture/data-analysis-with-python/grid-search-e4fyg

Raschka, S. (2015). *Rasbt/python-machine-learning-book* [Jupyter Notebook]. Retrieved from https://github.com/rasbt/python-machine-learning-book

Sharma, S. (2019, February 14). Activation Functions in Neural Networks. Retrieved September 22, 2019, from Medium website: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6