Supervised Learning

Luke Kehoe
lkehoe6

luke.kehoe@gatech.edu

1 INTRODUCTION

In this paper I will explore the performance of five machine learning algorithms on two problems each of a different nature and each using two different data sets. The performance of these will depend on the nature of the data and the problem being solved which will provide the contrast necessary to demonstrate the unique advantages and problems associated with each. Both objectively and comparatively.

The two data sets I have chosen for my analysis were the Cleveland Heart database (CHD) and the Auto MPG data set (AMD), both sourced via UCI. The respective goal of the machine learning techniques being implemented on these were predicting the presence of heart disease and the classification of a cars fuel efficiency.

The CHD is made of a 14 attribute subset of health bio-metrics from hospital patients taken from the original set of 76 along with a binary target attribute to indicate the confirmed presence of heart disease.

The AMD is made up of 8 automobile attributes including a continuous set of values for miles-per-gallon. The mpg is the attribute that set as the target attribute to predict. Fuel efficiency *in praxi* is considered poor, typical or very good. So the continuous nature of this attribute needed to be pre-processed in order to make the problem a classification problem.

2 IMPLEMENTATION AND PERFORMANCE

2.1 Pre-Processing

Due to the heightened focus on environmental concerns and fuel efficiency in recent decades current standards may be inappropriate as classification standards on AMD which is from the nineteen seventies. Therefore the upper and lower cutoffs for 'average' fuel efficiency (and therefore the definition of poor and good) a single standard deviation from the median was used.

CHD target variable was stratified numerically from 0 to 4 with 0 representing no heart disease while other values represented different types and degrees. Therefore non-zero values were transformed to 1 in order to create a binary classification for each representing simple presence.

Due to the relatively clean nature of both data sets dropping rows with invalid data created minimal loss. Therefore replacing or calculating placeholder values seemed unnecessary and very unlikely to effect performance of even a single algorithm.

2.2 Common Approaches

Across all algorithms and both data sets I implemented the use of a grid search with cross validation in order to optimize the hyper-parameters on each instance in order to to get the highest performance from each model.

To summarize, this searches through every combination of (provided) hyper-parameters against the metric of cross validation. Cross validation uses the validation set from the training data to prevent/judge over-fitting when selecting the optimal hyper-parameters for a final model.

Sci-Kit Learn provides a GridSearchCV for performing the grid search with cross validation which is what I used here also.

Splitting the data between training samples and testing samples was done with a 20% test ratio using Sci-Kit Learns train_test_split method.

Learning curves provide means of analyzing the effect of the number of training samples on the accuracy of the model both on the training data and the test data. It plots performance of the model on both the training set and the valida-

tion/test set. Below is the result of the decision tree with a shaded standard. deviation.

Validation curves are also a diagnostic tool for assessing the optimal choice of hyper-parameters. In the case of the decision tree I am showing the result of tree depth on performance below. Both measured on the training data and the validation (test) data.

A confusion matrix represents the grid of true positives, false positive, true negatives and false negatives. These are used to help assess a models performance via accuracy, precision, recall, and F1-score.

2.3 Decision Trees

I chose to implement my machine learning model for this algorithm using Sci-Kit Learns DecisionTreeClassfier.

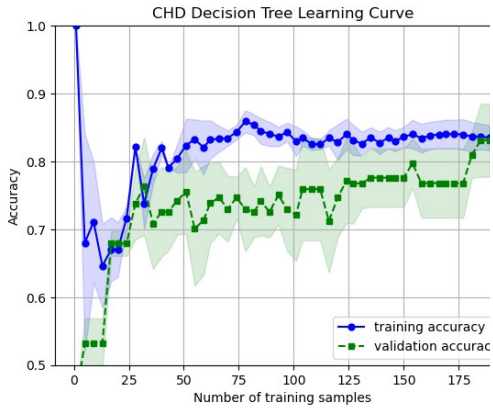
Decision trees can be optimized via pruning (setting maximum depth of tree in my case), limiting leaf size, splitting metric(criterion), minimal size of split and number of features used.

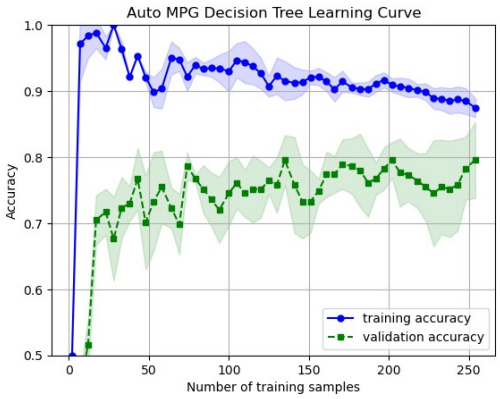
The search space I selected for these respectively were: 2 to 10, 1 – 10, Gini & Entropy, 2 – 10, and square root vs Log ^ 2.

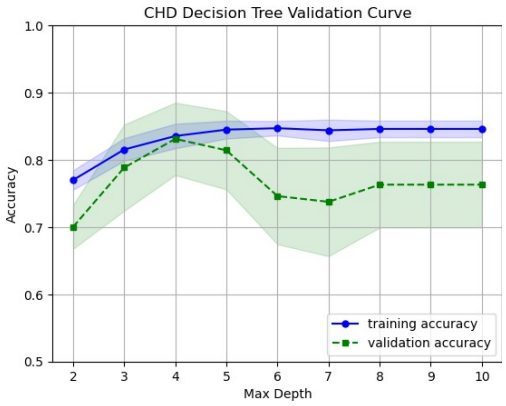
The results of the grid search found the optimal hyper-parameter values to be:

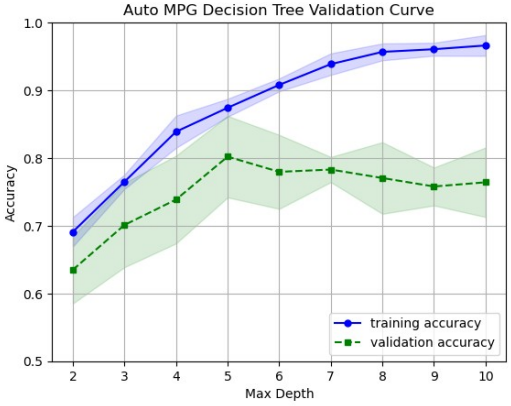
Hyper-Parameter	CHD	Auto MPG
Best Criterion	Gini	Gini
Best Maximum Depth	4	5
Best Minimum Samples for Split	2	3
Best Minimum Samples for Leaf	7	1
Best Maximum Features	Square Root	Square Root

2.3.1 *Decision Tree Learning and Validation Curves*









2.4 Boosting

For my boosted decision tree I used a popular and well known boosting technique called XGBoost from the like named library instantiated as an XGBClassifier.

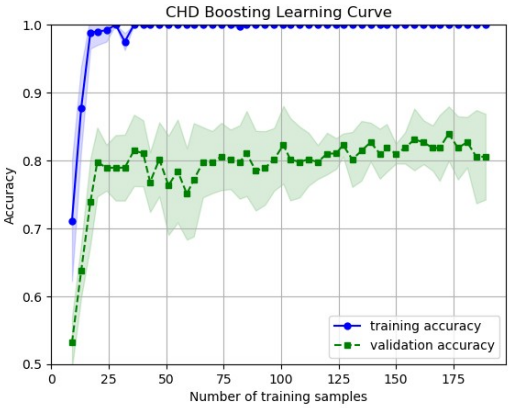
Boosting classifiers can be optimized with maximum depth, number of estimators and the learning rate hyper-parameters.

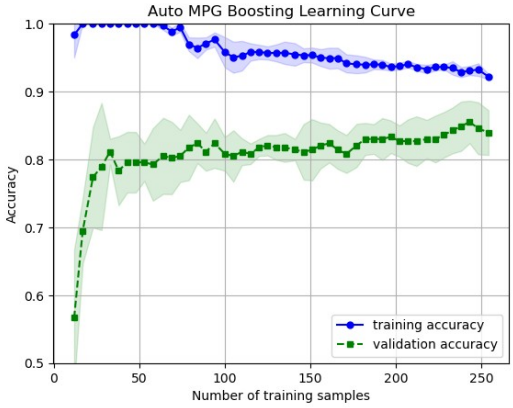
The search space I used for these were 2 to 10 for depth, 60 – 240 in 40 increments jumps and 0.1, 0.01 and 0.5 respectively.

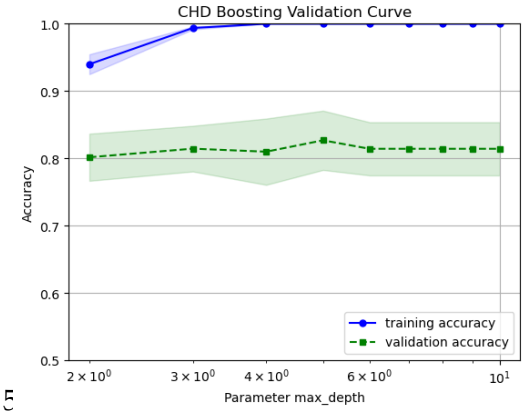
The results of the grid search found the optimal hyper-parameter values to be:

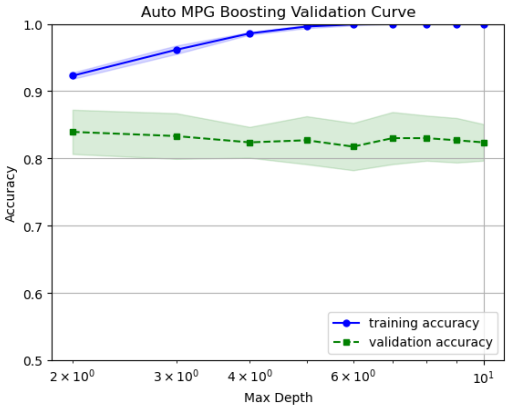
Hyper-Parameter	CHD	Auto MPG
Best Maximum Depth	5	2
Best Number of Estimators	180	60
Best Learning Rate	0.5	0.1

2.4.1 *Boosting Learning and Validation Curves*









2.5 K Nearest Neighbor

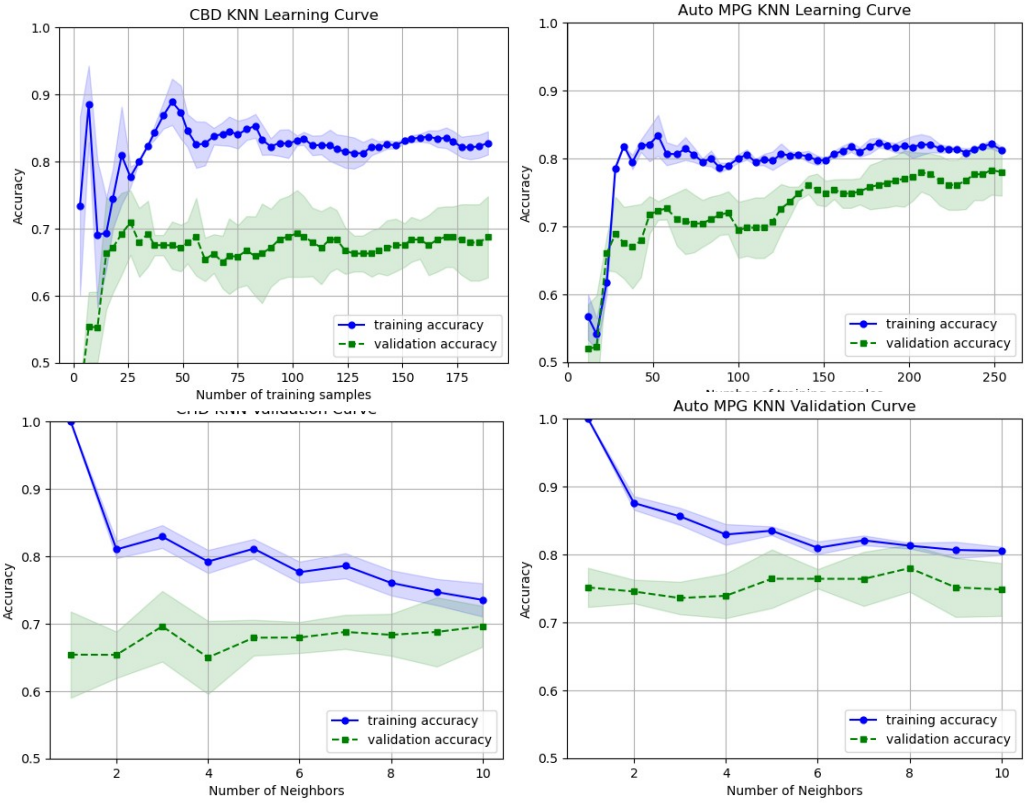
I chose to implement my machine learning model for this algorithm using Sci-Kit Learns KNeighborsClassifier.

K Nearest Neighbor (KNN) can be optimized primarily through it's namesake; the number of nearest neighbors used. However, other hyper-parameters to consider are weighting of neighbors, metric used to measure distance between neighbors, and which version of the KNN algorithm.

The search space I used for these hyper-parameters were:

Hyper-Parameter	Individual Search Space
Number of Neighbors	1 to 10
Weights	Uniform, Distance
Metrics	Euclidean, Manhattan, Chebyshev, Minkowski
Algorithm	Auto, Ball Tree, KD Tree, Brute Force

2.5.1 *KNN Learning and Validation Curves*



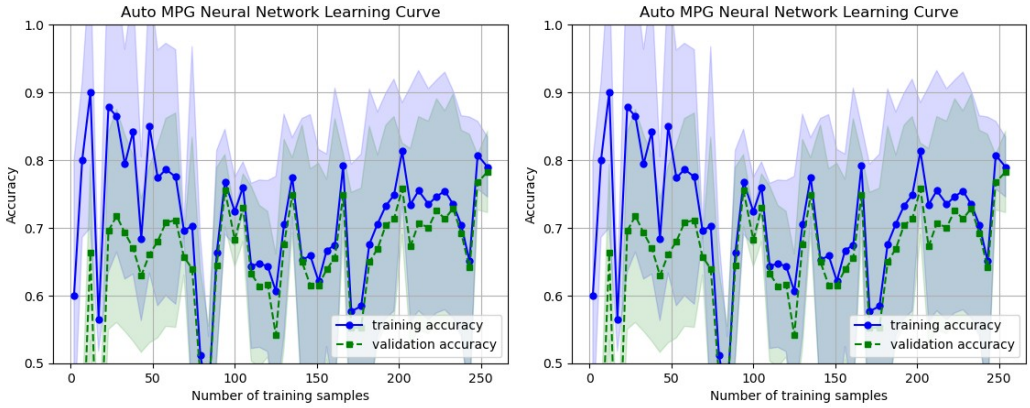
2.6 Neural Network

I chose to implement my machine learning model of this algorithm using Sci-Kit Learns neural network MLPClassifier.

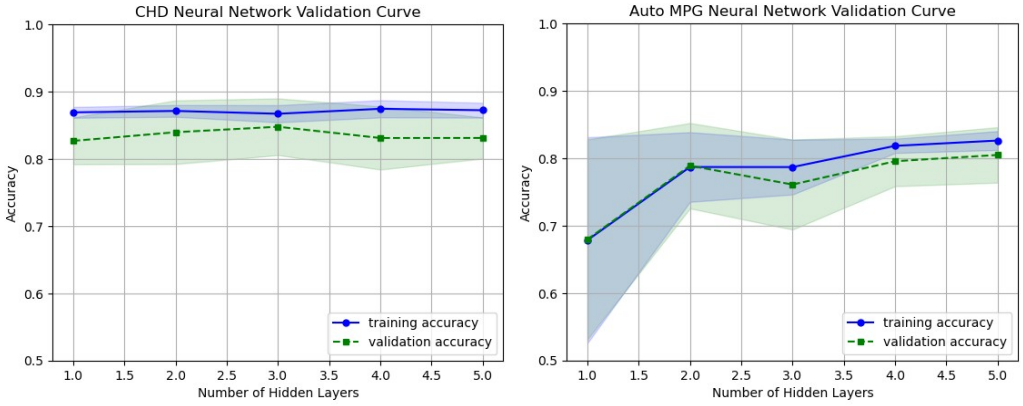
Neural networks have more than a few hyper-parameters but I will focus on optimizing mine via number of hidden layers, activation function, solver/optimization algorithm, learning rate and alpha/regularization.

Hyper-Parameter	Individual Search Space
Hidden Layers	1, 2
Activation	Identity, Logistic, TanH, ReLU
Solver	Euclidean, Manhattan, Chebyshev, Minkowski'
Alpha	LBFGS, SGD, Adam
Learning Rate	Constant, Adaptive, Inverse Scaling

2.6.1 *Neural Network Learning and Validation Curves*



/



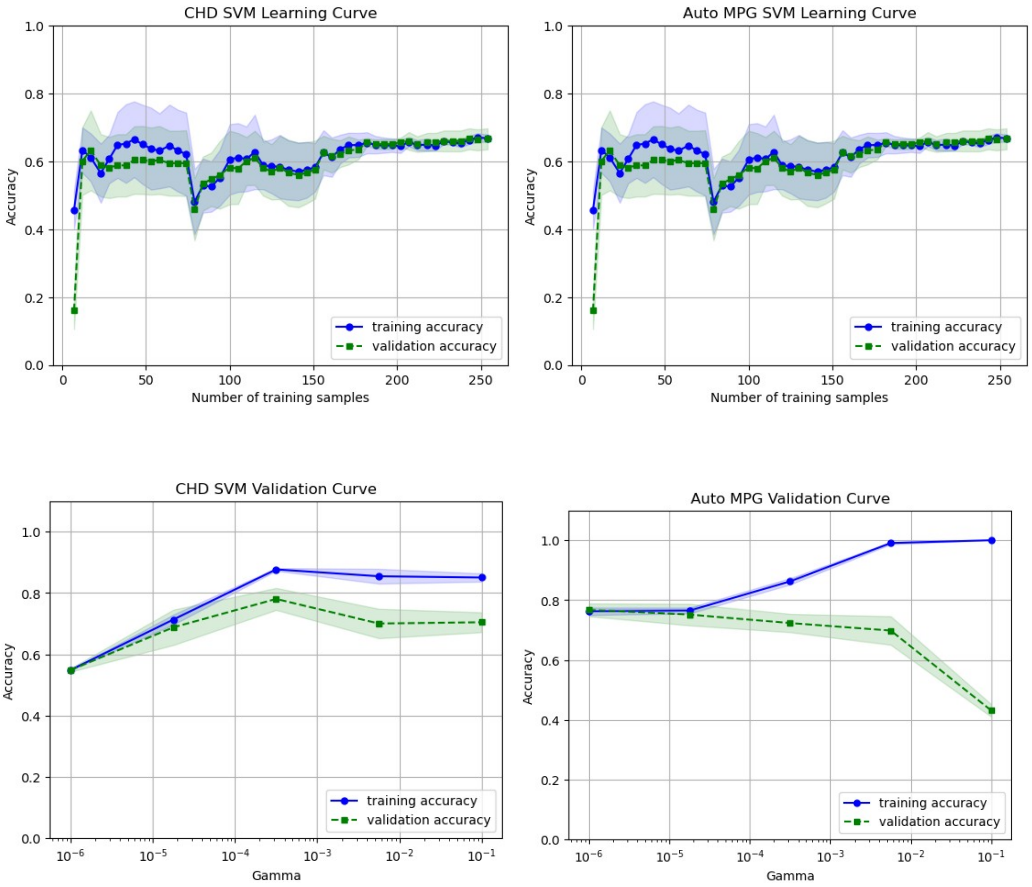
2.7 Support Vector Machine

I chose to implement my machine learning model for this algorithm using Sci-Kit Learns SVC which is their SVM classifier.

There are several tune-able hyper-parameters for SVM but due to the high computing cost and exponential increase in computing time a few values of just a couple were chosen.

Hyper-Parameter	Individual Search Space
Gamma	Linear, Scale, Auto
Kernel	RBF, Polynomial, Sigmoid

2.7.1 *SVM Learning and Validation Curves*



3 ANALYSIS

CHD

Algorithm	Accuracy	Precision	Recall	F1 Score
Decision Tree	0.91	0.83	0.67	0.74
Boost	0.92	0.73	0.80	0.76
KNN	0.81	0.67	0.53	0.59
Neural Network	0.92	0.85	0.83	0.83
SVM	0.79	0.76	0.43	0.55

Auto MPG

Algorithm	Accuracy	Precision	Recall	F1 Score
Decision Tree	0.86	0.83	0.87	0.84
Boost	0.92	0.73	0.80	0.76
KNN	0.83	0.84	0.77	0.78
Neural Network	0.87	0.46	0.60	0.52
SVM	0.73	0.47	0.58	0.50

Below I will compare algorithms to themselves between data sets and then compare algorithms against each other within a data set in order to analyze the relationships between data, problems and algorithms as comprehensively as possible within confines of this paper.

3.1 Comparison between Data Sets

3.1.1 *Decision Tree*

For both data sets the decision trees performed similarly with regards learning rates (see 2.6.1 charts). They approached near their maximum accuracy very quickly, doing so around only 50 samples. Overall across all metrics decision trees were comparable.

The reason for this somewhat uniform and acceptable performance in both cases comes down to the nature of decision trees themselves. They are versatile and practical for data sets of small to medium and sometimes large sizes while be able to capture non-linear relationships. Or inversely one could say this prevents them from being excelling particularly well or badly on any reasonable problem or data set.

As can be seen from the validation curves decision trees can easily be allowed to over-fit the data. This was particularly true for AMD. The reason for this may be an issue with weighting of certain attributes over the mpg of a car that require a deeper tree and more importantly that AMD has less features to work

with. The deeper the tree the greater the possibility of over fitting. With a more evenly distributed weight relationship between features and that there were more features to consider, the possibility of a greater delta between training and testing when using sub-optimal hyper-parameters becomes more likely.

3.1.2 *Boosting*

Boosting performed almost identically between data sets and it did so with very different optimal hyper parameters. However, as can be seen with the validation curves for depth there was little difference between the range of tree depths and was within a standard deviation which likely explains the difference. An analysis of estimator count is likely to find the same. Boosting by relying on the dynamic nature of randomness by using many weak models to create a strong one is probably causative as to it's consistent performance between data sets. This is more likely the case since my choice of XBG has proven itself and is known for being such.

3.1.3 *KNN*

KNN proved to be superior on the AMD overall and had a smaller delta between testing and training over sample size and number of neighbors. KNN performs better with features more directly related to the target which is the case with fuel efficiency and weight compared to heart disease heart rate. It also does better on more balanced data which due to my classification definitions was directly made so while the heart data is far more imbalanced where most do not have heart disease. This suggest a performance gain could be had via pre-processing such as over/under sampling in CHD.

3.1.4 *Neural Networks*

The learning curves for NN were erratic with large standard deviations. This can be explained by NN being overly complex for smaller data sets of which CHD and AMD may be for it. They are also more sensitive to initial settings. During the training the models were failing to converge. Generally, NN need careful architecture design to get good performance.

3.1.5 *Support Vector Machines*

For both sets the SVM models approached their best accuracy very rapidly. In this case it can be explained by the nature of SVMs; they are designed to be ro-

bust to outliers. This allows a quick convergence to their potential. As can be seen with the validation curves with gamma, the AMD is able to over fit more easily. This is most likely due to the extra class/dimension of the AMD targets. The curse of dimensionality means that there is more margin between instances for AMD so a smaller margin (what an increasing gamma causes) is more likely to over fitting.

3.2 Comparison across Data Sets

At face value of an equal weighting between performance metrics Boosting/XGBoost ranks highest overall. It is known for it's robustness in performance so may not be surprising. Since it's designed to be so and has the ability for great feature selection along with levering convergence via the nature of randomness it is unlikely to fail dramatically on any particular type of data. Decision trees came in second by this standard which similar is due to the general suitability. Despite the limitations of the data for neural networks they did adequatly. With very large data sets, extreme numbers of features and non-linearity of features to targets then neural networks would very possibly supplant both and certainly decision trees.

4 REFERENCES

1. https://scikit-learn.org/stable/
2. https://xgboost.readthedocs.io/en/stable/