Chess Endgames

& Heart Failure Prediction

Steven Haertling

UTD 05/11/2021

Table of Contents

Introduction

I started dreaming of an algorithm to tell me if I had lost on the chess board after taking my Machine Learning basics class. I went to UCI data sets repositories for some chess endgame related data sets. I started with King Rook Vs. King Pawn(KRKP) and King Rook Vs. King, but the 2nd was a inductive logic programming problem. It was too complicated, so I focused on KRKP to prep the data for various algorithms from sklearn. The sklearn library classifiers require numerical features and treats them as continuous. I used a decision tree, bagging, boosting, k nearest neighbors, logistic regression, and multi-layer perceptron to classify the data sets. I have my program plotting confusion matrices, receiver operating characteristic (ROC), and precision vs recall.

When I went to repick a second data set from UCI's repository I found it down, so I picked a different one from kaggle. That one was a heart failure prediction data set that was 299 elements. I added the precision vs recall for this data set, since it could be important to catch people who are at risk for heart issues. I will break down the data sets, how they were processed, and the classifiers performances.

Data Sets

King Rook Vs. King Pawn A7

The KRKP data set comes in a string format, containing 36 features and 1 class. Most of the features were true false statements, so they could be mapped to 0,1 easily. The "katri" feature has 3 values that were not continuous. At first I tried mapping it to 0, 1, 2, but that makes it a continuous feature, when it was really 3 options. A way to fix that is with one-hot encoding, basically expanding the feature into 3 separate features. If it was b, the new map would be 1,0,0 instead of 0, continuing on through for n and w. Every classifier had a huge improvement when used on the one-hot.

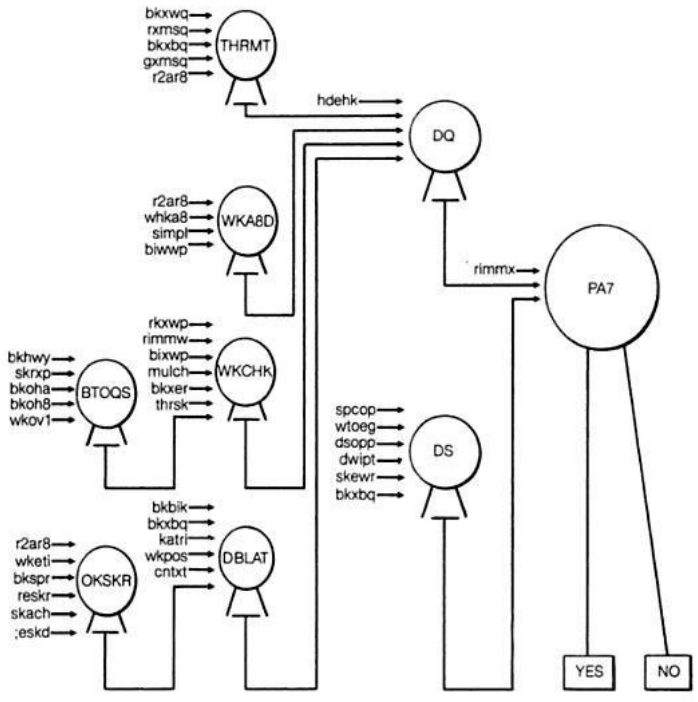
The data comes from Alen D. Shapiro from his book, Structured Induction in Expert Systems. The data set has 3196 elements with no missing values, and a breakdown of 1669 (52%) as white can win vs 1527 (48%) as white cannot win. Its a big data set with a balanced class (nowin/win). The typical board position is f, f, f, f, f, f, f, f, f, f, f, f, l, f, n, f, f, t, f, f, f, f, f, f, f, t, f, f, f, f, f, f, f, t, t, n, win. This makes it hard to decipher what is on the board, since the details of those feature names are in the book. The board position are abstracted away making this a problem that can be classified more easily than the KRK one which required domain knowledge included into the program.

I needed to translate the csv string values to integers for sklearn classifiers to be able to use the data. I used numpy generate from text to read in the csv to a numpy matrix. I encoded the features one using labelencoder. This made everything [0,1] instead of ['f','t'], but katri was [0,1,2]. I eventually split katri into 3 new features using LabelBinarizer. The last bit of preprocessing was to use train test split from sklearn. I did 10-cross validation on every classifier and was about to get tight standard deviations on any training size. (1%, or 30% ~.15 std dev)

bkblk	['f' 't']	reskd	['f' 't']
bknwy	['f' 't']	reskr	['f' 't']
bkon8	['f' 't']	rimmx	['f' 't']
bkona	['f' 't']	rkxwp	['f' 't']
bkspr	['f' 't']	rxmsq	['f' 't']
bkxbq	['f' 't']	simpl	['f' 't']
bkxcr	['f' 't']	skach	['f' 't']
bkxwp	['f' 't']	skewr	['f' 't']
blxwp	['f' 't']	skrxp	['f' 't']
bxqsq	['f' 't']	spcop	['f' 't']
cntxt	['f' 't']	stlmt	['f' 't']
dsopp	['f' 't']	thrsk	['f' 't']
dwipd	['g' 'l']	wkcti	['f' 't']
hdchk	['f' 't']	wkna8	['f' 't']
katri	['b' 'n' 'w']	wknck	['f' 't']
mulch	['f' 't']	wkovl	['f' 't']
qxmsq	['f' 't']	wkpos	['f' 't']
r2ar8	['f' 't']	wtoeg	['n' 't']
		y	['nowin' 'won']

Figure 1: KRKP Data set

"Structured Induction in Expert Systems" Alen Shapiro (1987) Turing Insitute Press in association with Addition-Wesley



Procedural hierarchy evolved in the course of computer induction from expert-supplied examples of a decision rule for ending King and Pawn (on a7) versus King and Rook (White to move). Boxes denote primitive attributes whose bodies were instantiated in each case by induction of a linear (non-nested) if-then-else expression.

Figure 2: The layout of features from Shapiro's book. Makes it look like a boolean solver.

Heart Failure Prediction

This data set has fewer features and a much smaller sample size than the UCI data set, it is also already in integer form. I picked this one for ease of running the same sorts of algorithms on it as the KRKP problem. It didn't have an string and non continuous data, but it did have outliers and continuous features. I used zscores to remove outliers and a min max scalar to set a boundary for the continuous features. The features in the data set can be seen in figure 3. It contains 12 features and 1 class.

The data comes from a paper in the BMC journal written by Chicco and Jurman. They conclude that using just serum_creatinine and ejection_fraction as features lead to more accurate predictions than using the whole dataset. I will try this out to see if I get similar results.

	count	mean	std
age	299.0	60.833893	11.894809
anaemia	299.0	0.431438	0.496107
creatinine_phosphokinase	299.0	581.839465	970.287881
diabetes	299.0	0.418060	0.494067
ejection_fraction	299.0	38.083612	11.834841
high_blood_pressure	299.0	0.351171	0.478136
platelets	299.0	263358.029264	97804.236869
serum_creatinine	299.0	1.393880	1.034510
serum_sodium	299.0	136.625418	4.412477
sex	299.0	0.648829	0.478136
smoking	299.0	0.321070	0.467670
time	299.0	130.260870	77.614208
DEATH_EVENT	299.0	0.321070	0.467670

Figure 3: Heart Failure Data set

Test Results

King Rook Vs. King Pawn A7

Decision Tree

I thought the decision tree would be a good classifier with all the binary features. I show in figure 4 and 5 the confusion matrix and ROC curves from before using hotshot. We can see at too low a depth, the classifier is terrible. The decision tree ranged from terrible at 56% accuracy to 96% 10-Cross Validation average with different depth and feature values. I am going to use the hot-shot method to look at the results from the classifier to find average confusion matrices, accuracy, true positive rate, false positive rate, and the area under the ROC curve.

The test split on this hot-shot version of KRKP is 70% test data and 30% training data. I kept the trees above 90% to average the, since those are the only one worth using. I tossed 56 to 76 % models, these come from being hamstrung with a low depth and low feature split count. The average confusion matrix is 1036 true negative, 20 false positive, 33 false negative, and 1149 true positive. The average true positive rate was 97%, false positive rate of 2%, an average ROC score of .98, and a average accuracy of 97.6%. Decision tree is a excellent option for this data. The theme for the best preforming trees looks to be the ones with max features after depth 3 do better. The ROC curve plots are generated by the python scripts and put in folder /HotShot/dtreePlots/ with the confusion matrices.

The smallest tree with a good 10 cross validation score is a tree of depth 3 and 15 feature splits. Its 10 cross validated accuracy was 98% and had a testing accuracy of 1. It also had a ROC score of 1. This specific tree had a true positive rate of 1 and false positive of 0.

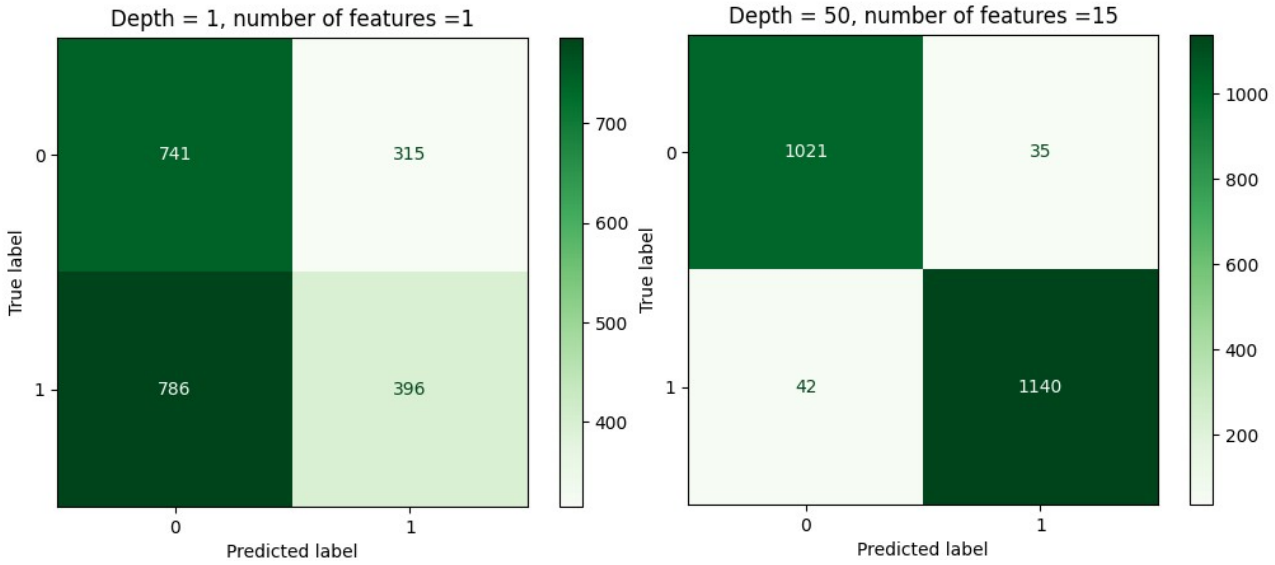


Figure 4: Min setting vs max setting for decision tree before hot-shot

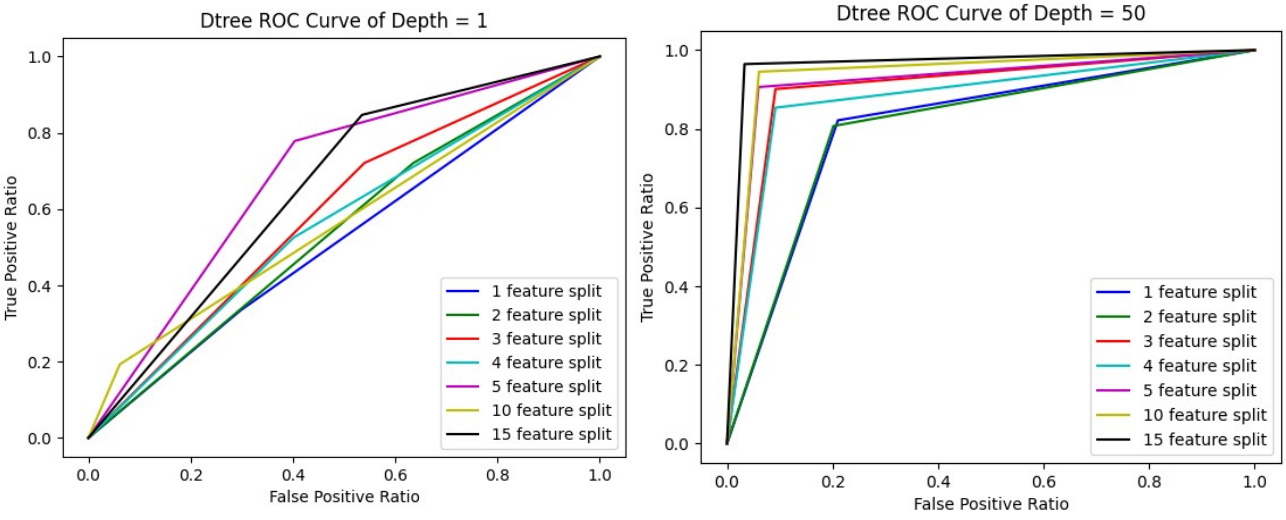


Figure 5: ROC curves for min depth and max depth before hot-shot

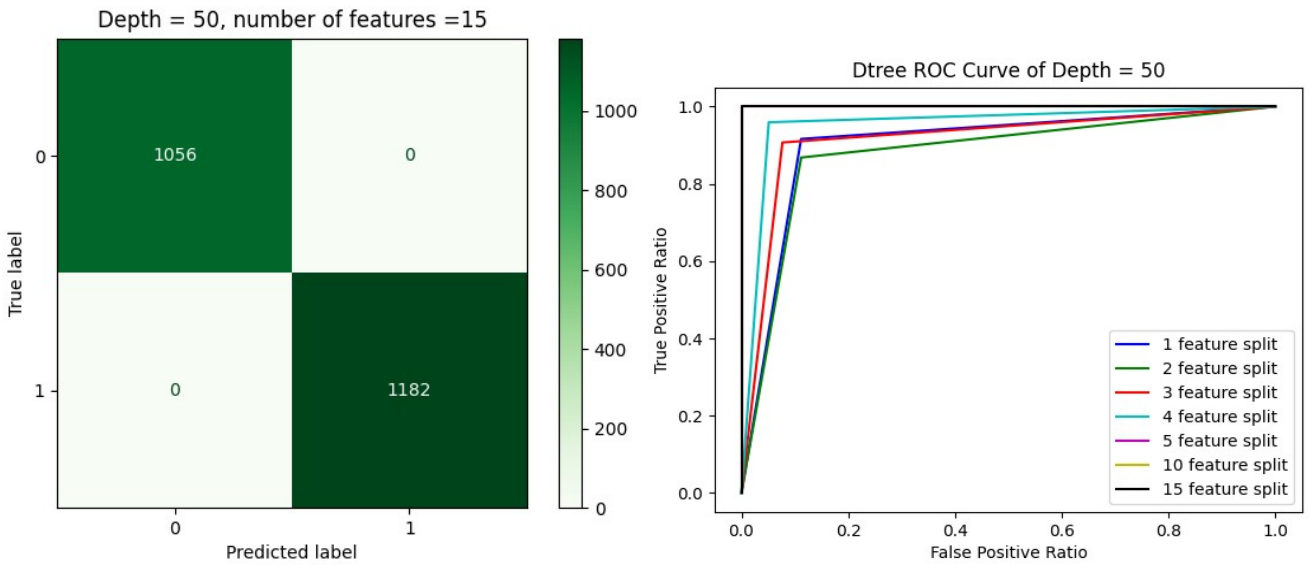


Figure 6: With hot-shot max settings on a decision tree

Bagging

To try out different versions of the bagging classifier I used different depths and bag sizes. The underlying decision tree splits on 1 feature. As with decision trees, we can see that moving to hotshot for the one feature allows for the classifier to learn every case with max feature splits. An interesting result form the hotshot data is 1 depth 10 bag size 100% accuracy classifier that had a 10-fold cross validation average accuracy of 67% and a std deviation there of 13%. I removed the classifiers with very poor accuracy and poor cross validation scores to average the values of interest. Generally the poor performers were 1 bag, and low depth models. The average confusion matrix of usable bagging models were: 1022 true negative, 34 false positive, 33 false negative, and 1149 true positive. The average true positive rate was 97%, false positive rate of 3%, an average ROC score of .99, and a average accuracy of 97%. In figure 9, we can see that the more bags, the better the ROC curve.

The fastest running bagging model with top tier CV score and accuracy was a bagging model with a depth of 10 and bag size of 15. It runs in 1 second, which is the average run time. It its comparable to the higher depth and larger bag size models that take 1.5 seconds. Its 10 cross validated accuracy was 99% and had a testing accuracy of 99%. It also had a ROC score of 1. This specific tree had a true positive rate of 99% and false positive of 1%, and an accuracy of 1.

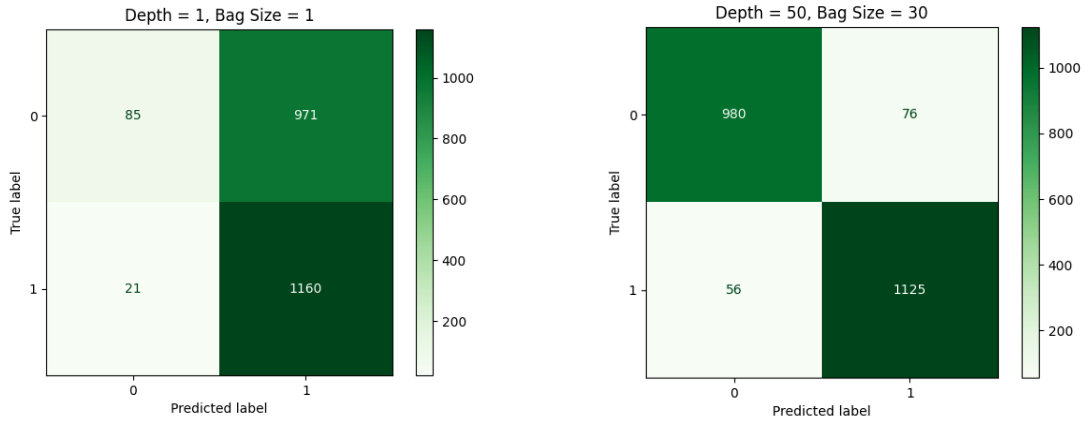


Figure 7: Min setting vs max setting for bagging before hot-shot

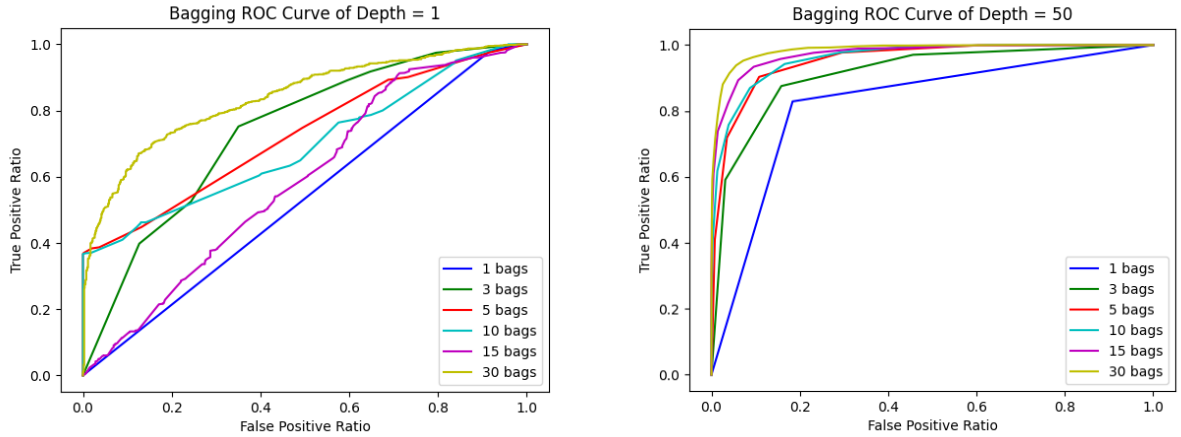


Figure 8: ROC curves for min depth and max depth before hot-shot

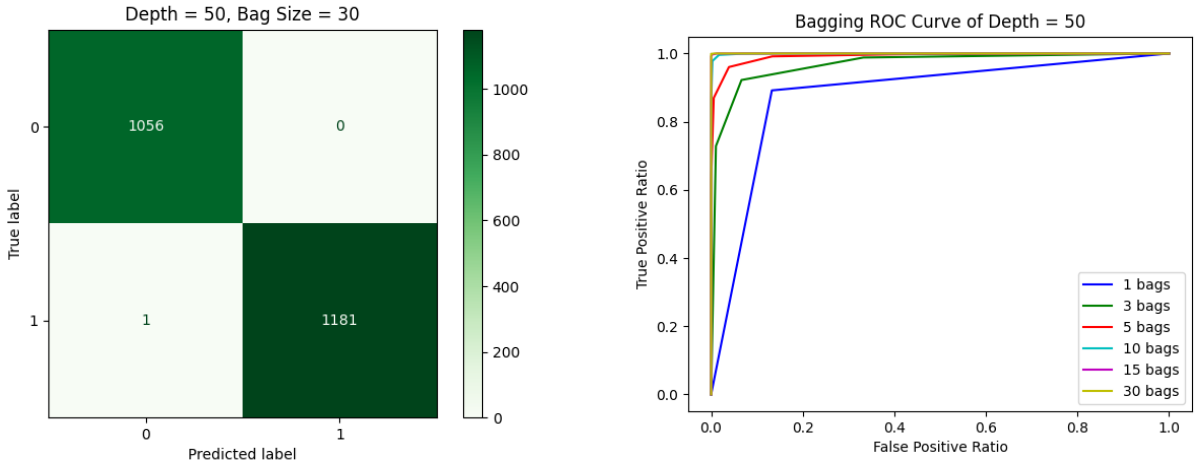


Figure 9: With hot-shot max settings on a bagging model

Boosting

The idea behind boosting is you use weak classifiers such as decision stumps, and focus on their error over a number of new estimators. The smallest group of estimators I used was 10 and then all the way up to 60. Most of the models have decent accuracy, but 3 of the 12 have poor cross validation scores with a high standard deviation. You can see in figure 12 the 2 depth 10 estimator model is poor. I used to a test split of 70% test values and 30% training. On the boosting classifiers the lower parameter ones had issues with cross validation standard deviation being large at .09 to .134. The boosting classifier in figure 10 and 11 is before hot shot and they seem like you could learn at maxed out settings, but after hot shot the roc curve launches to a sharp TPR in most bag sizes. The cross validation calls bag sizes 10 and 20 into question as effective classifiers on this data set.

The average confusion matrix of usable boosting models were: 985 true negative, 71 false positive, 61 false negative, and 1121 true positive. The average true positive rate was 95%, false positive rate of 7%, an average ROC score of .97, and a average accuracy of 94%. In figure 12, we can see that the more bags, the better the ROC curve. The average was brought down by a model, 1 depth, 50 estimators, that had a high cross validation score on the training data, but scored low on the test set.

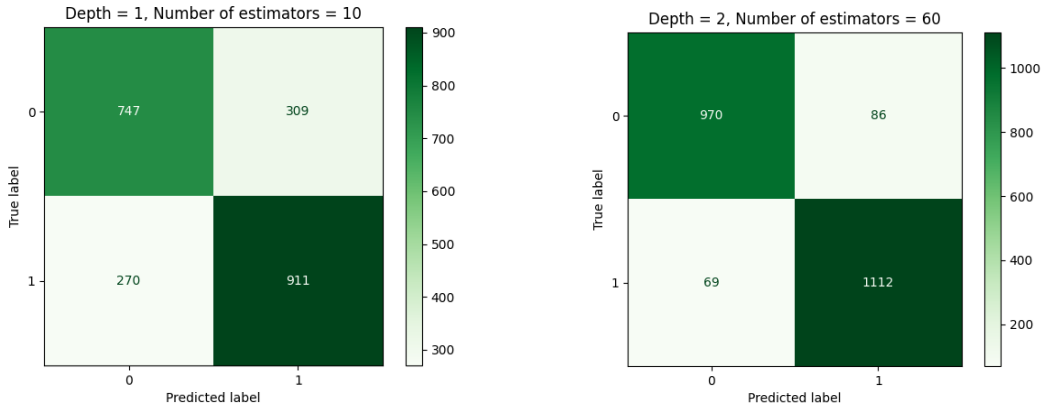


Figure 10: Min setting vs max setting for boosting before hot-shot

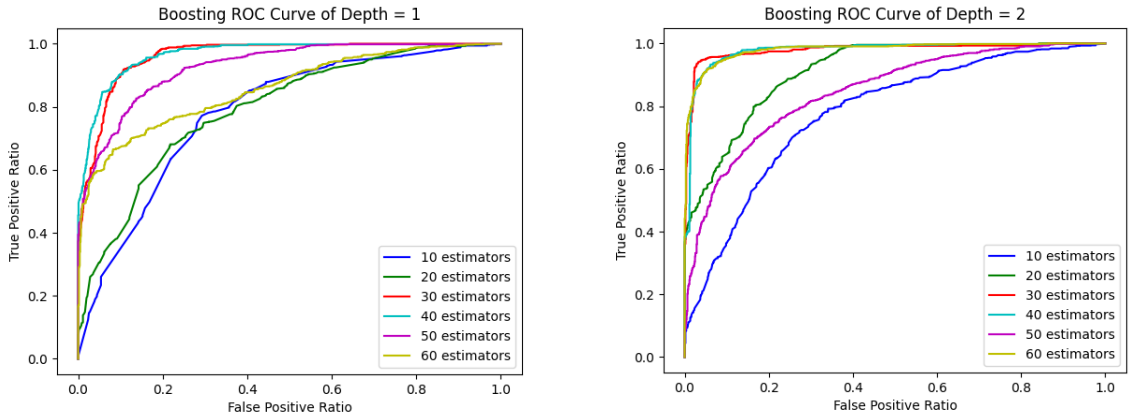


Figure 11: ROC curves for min depth and max depth before hot-shot

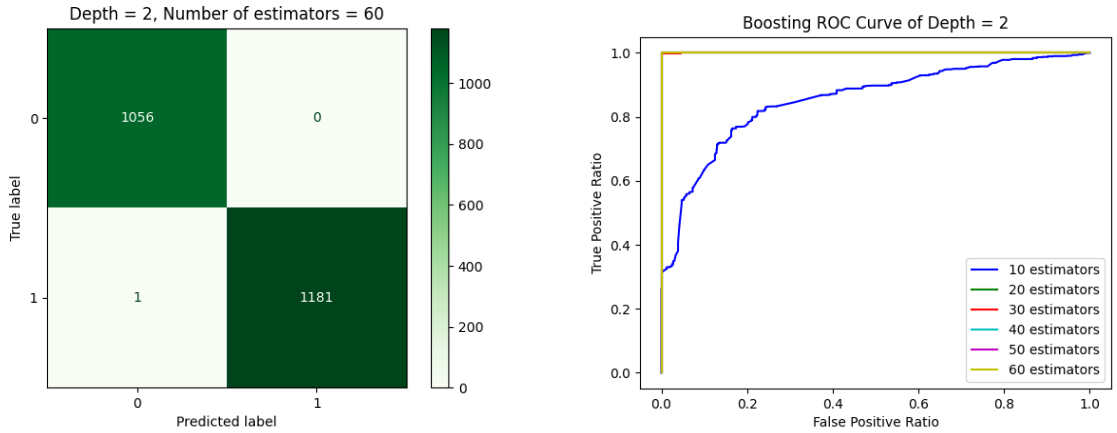


Figure 12: With hot-shot max settings on a boosting model

K Nearest Neighbors

For KNN I changed the algorithm, weight, and K to try and find an ideal setting for the knn classifier, KNeighborsClassifier. Sklearn uses 3 types of algorithms to calculate distance: ball tree, K-D tree, and brute force. K-D tree tries to address the inefficient brute force by encoding aggregate distances and not calculating everyone. Ball tree is meant to be an efficient algorithm for higher dimensions. Sklearn can pick the best one for the data set with the auto setting. The average run time went from shortest to longest as brute with a 1second average, K-D tree with a 2.3s average, and ball tree with a 1.6 second average. I used many different leafs and didn't see an effect on the accuracy so I removed that as a test feature, this is a mistake on my part because it should only effect speed and space which I didn't track. The default leaf count for kd and ball trees is 30, and changing that can affect the speed of the algorithm.

For the weights there was two built in options: uniform weights and distance weights. The uniform weights were on average equivalent on accuracy, and .26 seconds slower than the distance weights in the brute. The lower values of K (1,2,3) performed worse than the higher ones (5, 10, 15, 30) on the KRKP data set. The higher values of K all had accuracies of 99%. The higher k seemed to correlate to an slight increase on time, but the distance algorithm had the most important effect on that.

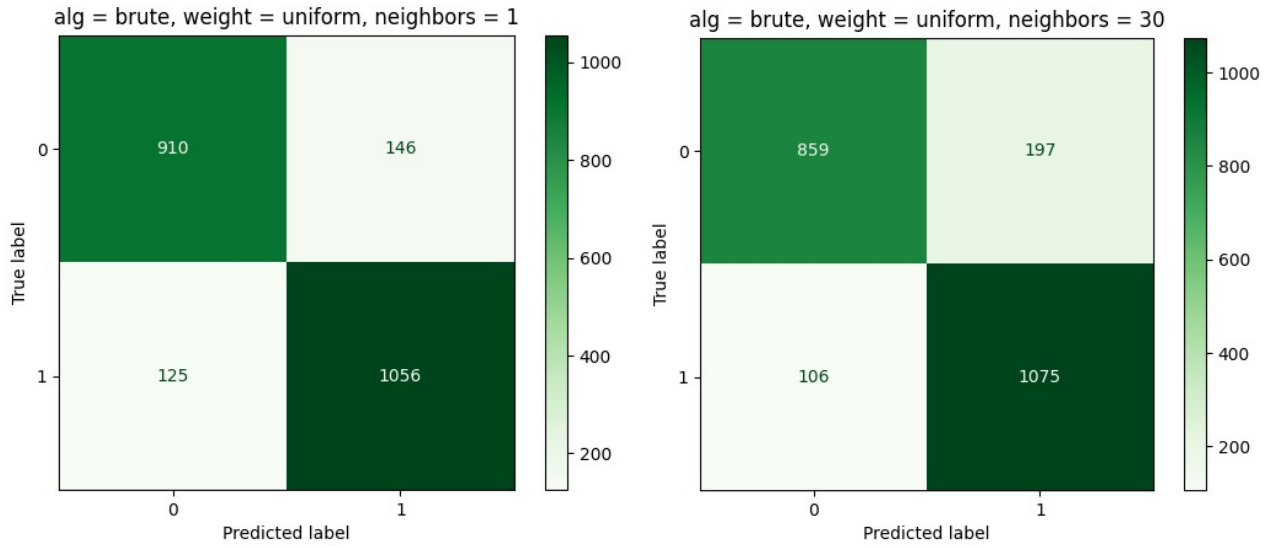


Figure 13: 1- k vs 30- k setting for k-nn before hot-shot, on brute force with uniform weight

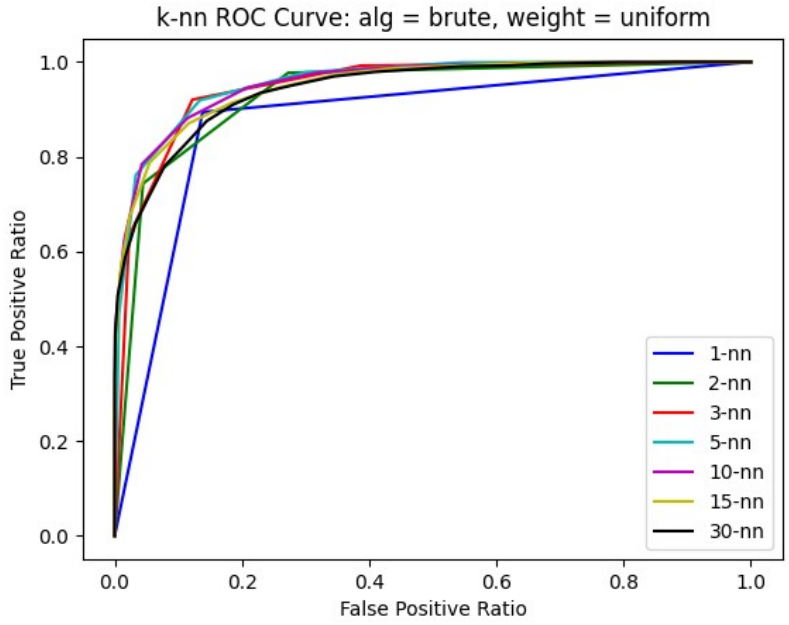


Figure 14: k-nn ROC curve using uniform weight & brute calculator before hot-shot

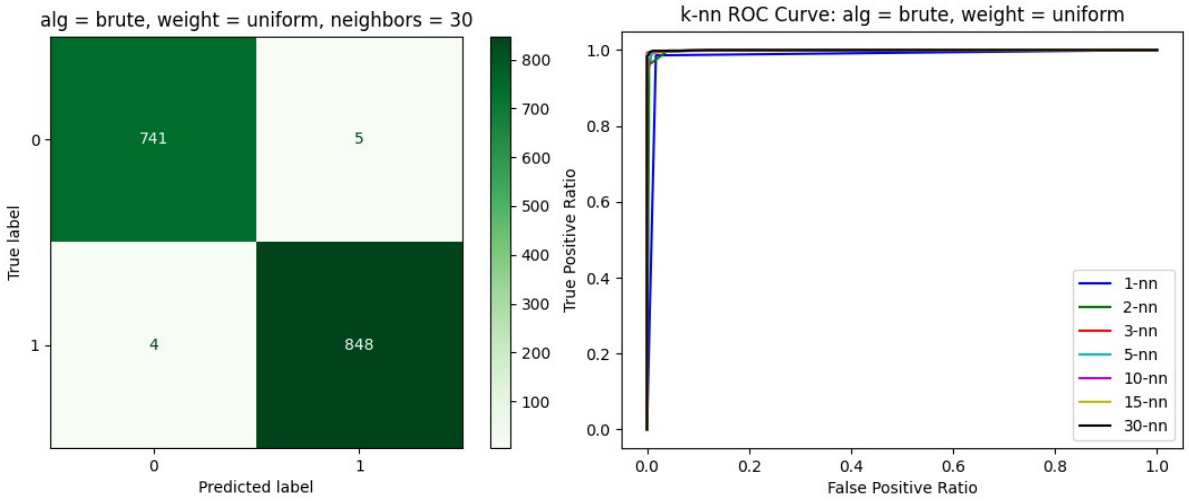


Figure 15: confusion matrix and ROC of hot-shot data

Logistic Regression

I tried out different solvers and C values to have different logistic regression models. The solvers were: newton-cg, lbfgs, liblinear, sag, and saga. The default solver is lbfgs since it is more robust, and works well on small data sets. We learned about sag in class, stochastic average gradient descent which is good for large data sets and feature sets. All the solvers had similar accuracy scores of 100% on the hot shot data. They all had 2 misses (99.9% accuracy) with a small C (.001), except for liblinear that missed 15 (99.3% accuracy) .The biggest stat differentiation between them was average run time. The fastest was liblinear at .55 seconds on average, followed by lbfgs at .67s, newton-cg at .7s, sag at 1.6s and saga at 1.9s.

Moving from the data preprocessing of no hotshot on feature katri the logistic regression solver can be seen clearly in figure 17. The ROC curve improved with every increase of C like I expected, but switching to hot shot for the one feature made the ROC curve for every C value to be close to perfect classifiers. In figure 16 there is an example of C going up(regularization going down) and confusion going down.

The average confusion matrix of usable boosting models were: 1055 true negative, 1 false positive, 0 false negative, and 1182 true positive. The average true positive rate was 100%, false positive rate of .1%, an average ROC score of 1, and a average accuracy of 100%. The average 10 fold cross validation accuracy was 99.9%, with the C = .001 models bringing it down from 100%.

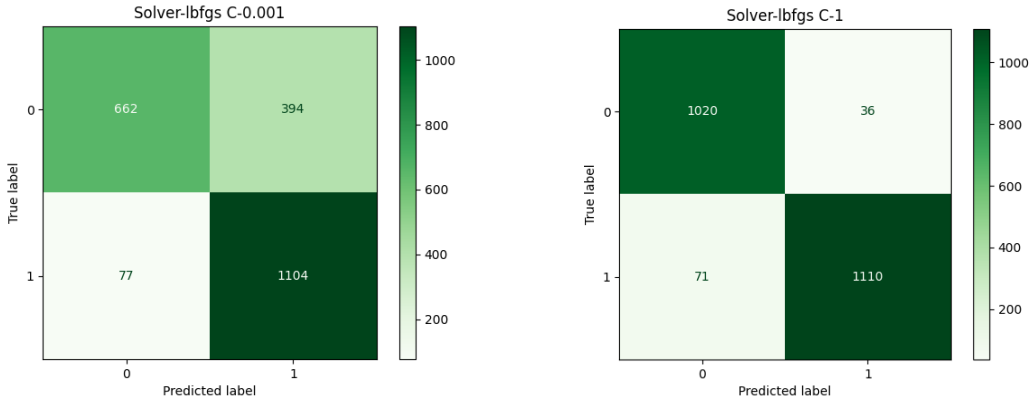


Figure 16: Confusion matrices of lbfgs solver before hot shot

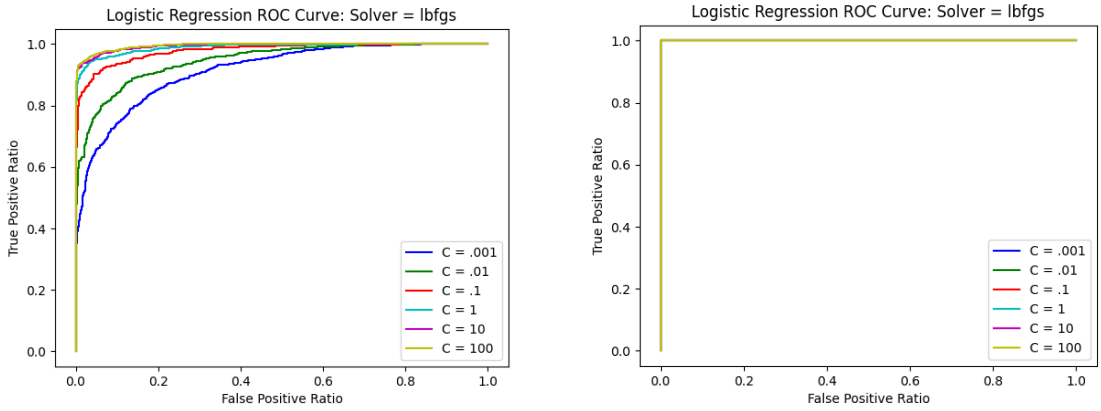


Figure 17: The ROC curve on the left is before hot shot

Multi-Layer Perceptron

The parameters I rotated through for the mlp classifier were the activation and hidden layer sizes. I used identity and tanh as the activation features for the model. Identity is a no operation activation, and tanh is a hypberbolic tan function activation. The solver is the same solver from logistic regression and is lbfgs. I picked this one to keep the model simple. In figure 18 I show why alpha isn't being varied as a test, and the roc curve of all 4 sklearn activation features. Since the 1 layer model worked to 100% accuracy, the multi layer version should do similarly.

The average confusion matrix of usable boosting models were: 1056 true negative, 0 false positive, 0 false negative, and 1182 true positive. The average true positive rate was 100%, false positive rate of 0%, an average ROC score of 1, and a average accuracy of 100%. The average 10 fold cross validation accuracy was 99.9%.

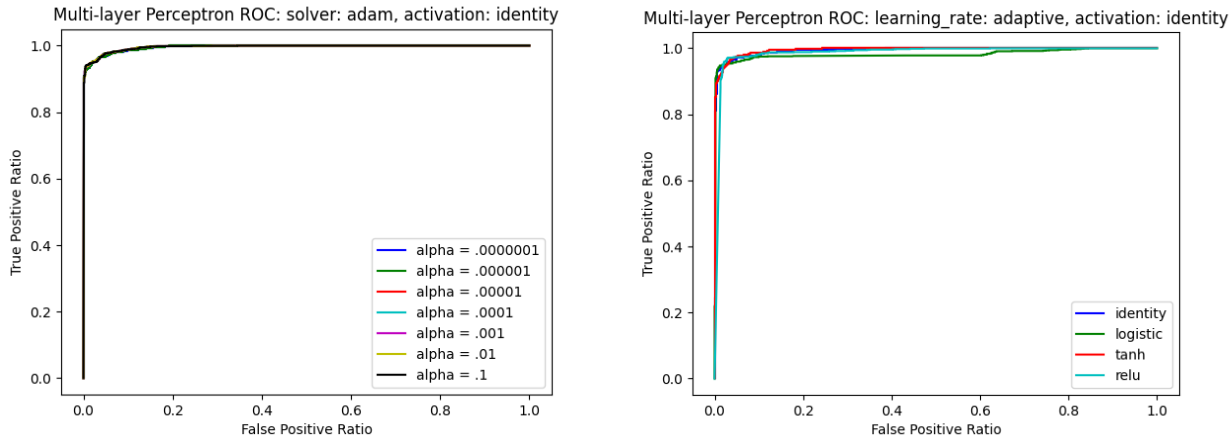


Figure 18: I picked to use the example alpha, since it didn't affect the ROC curve. I used identity and tanh because they preformed the best on the pre hotshot data.

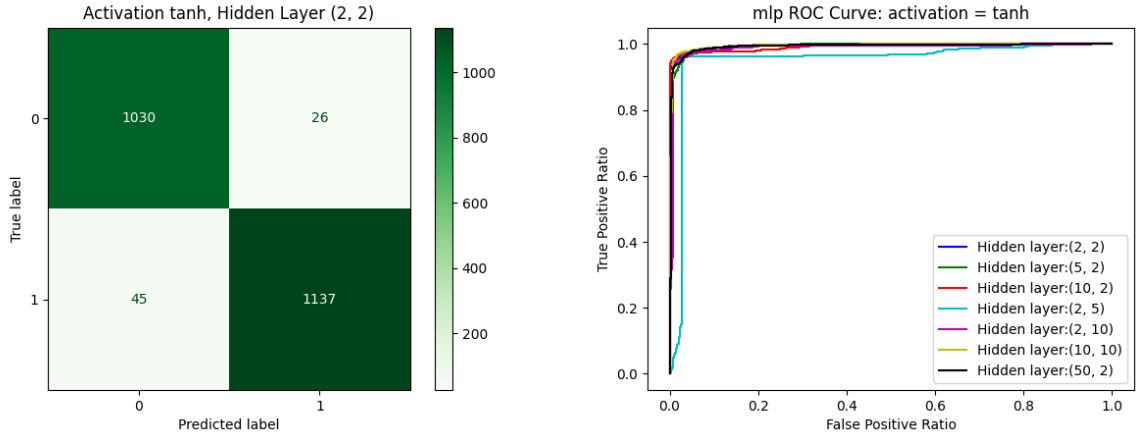


Figure 19: MLP classifier confusion matrix & ROC curve before hotshot

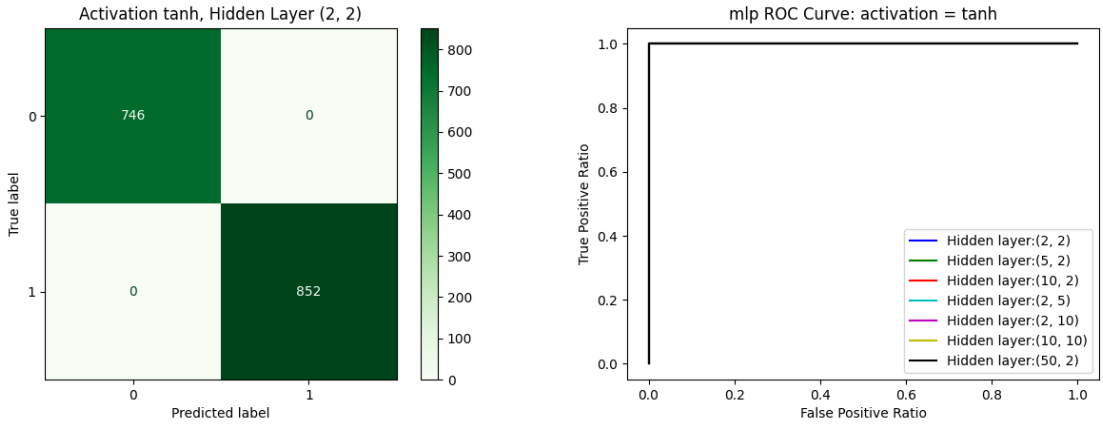


Figure 20: After Hotshot, mlp perfectly identified the data.

Heart Failure Prediction

Decision Tree

The HFP data set I used a 50% test and 50% training split to classify on. The average confusion matrix of decision tree models were: 99 true negative, 2 false positive, 1 false negative, and 38 true positive. The average true positive rate was 96%, false positive rate of 1.6%, an average ROC score of .97, and a average accuracy of 97.8%. The 10 fold cross validation average was 99% with an average standard deviation of 1.5%.

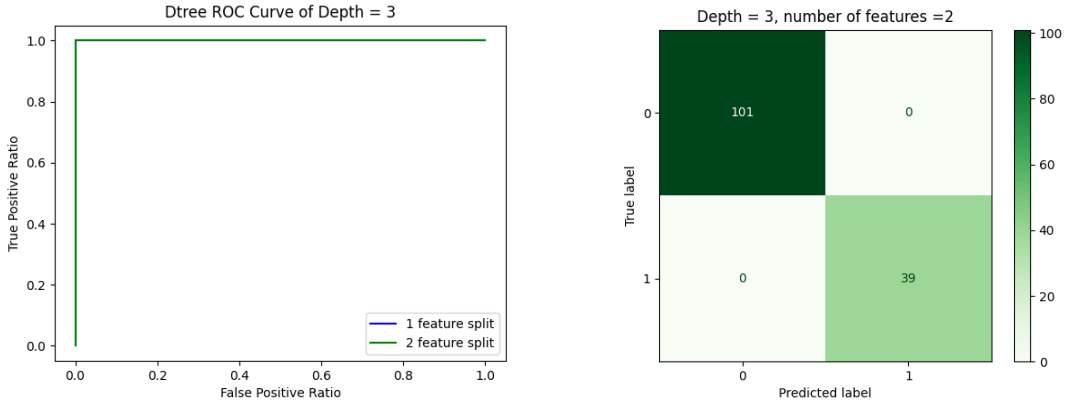


Figure 21: Confusion matrix and ROC curve of Dtree that classifys the data with a high cross validation score.

Bagging

The bagging data needed to clean up the low depth and low bag size models that had high standard deviation in its cross validation. The average confusion matrix of bagging models were: 101 true negative, 0 false positive, 0 false negative, and 39 true positive. The average true positive rate was 99%, false positive rate of .08%, an average ROC score of .9999, and a average accuracy of 99.7%.

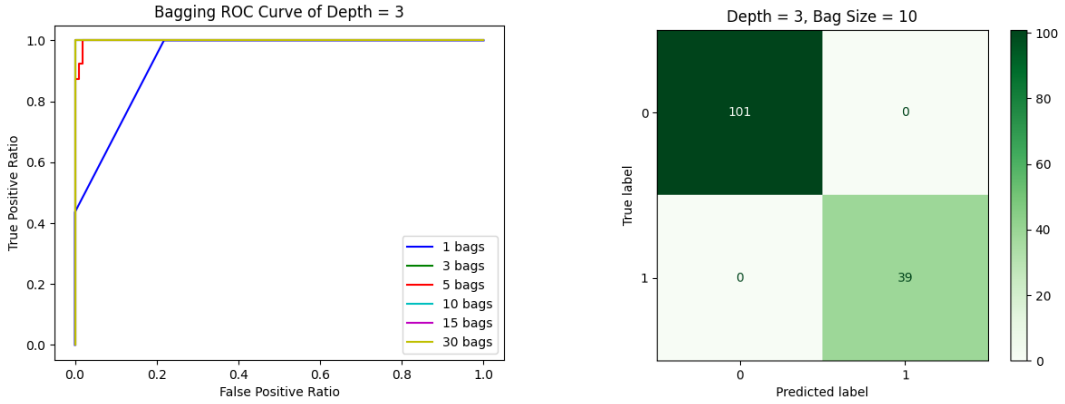


Figure 22: ROC Curve Showing when the classifier starts hitting 100% accuracy and the confusion.

Boosting

The average confusion matrix of boosting models were: 101 true negative, 0 false positive, 0 false negative, and 39 true positive. The average true positive rate was 100%, false positive rate of 0%, an average ROC score of 1, and a average accuracy of 100%.

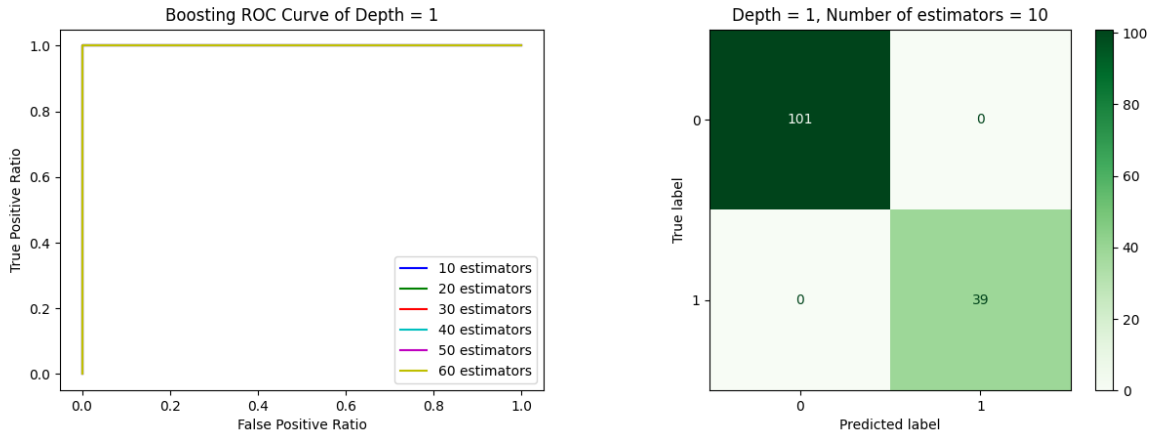


Figure 23: ROC curve and Confusion matrix of boosting classifier on HFP data.

K Nearest Neighbors

The average confusion matrix of knn models were: 101 true negative, 0 false positive, 0 false negative, and 39 true positive. The average true positive rate was 99%, false positive rate of .08%, an average ROC score of .9999, and a average accuracy of 99.7%. The average time for the knn models was .51 seconds.

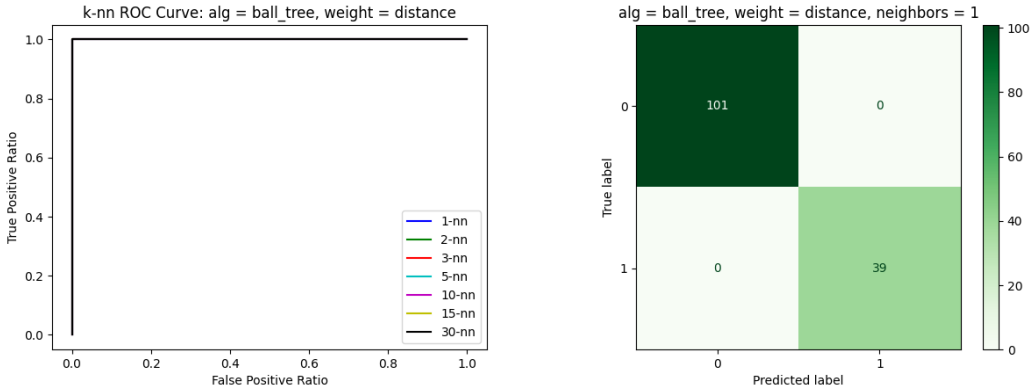


Figure 24: ROC curve and confusion matrix for knn.

Logistic Regression

The C values of .001 and .01 are terrible with any of the solvers and are 20% worse than C of at least .1. I removed those models from the aggregate and then averaged. You can see the confusion matrix example in figure 25.

The average confusion matrix of logistic regression models were: 101 true negative, 0 false positive, 0 false negative, and 39 true positive. The average true positive rate was 100%, false positive rate of 0%, an average ROC score of 1, and a average accuracy of 100%. The average time for the log models was .5 seconds.

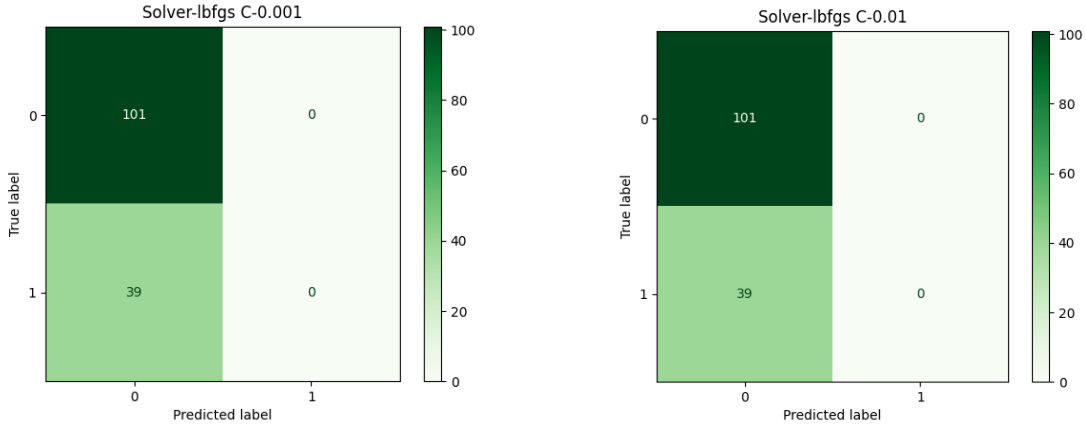


Figure 25: Examples of the models I removed from consideration

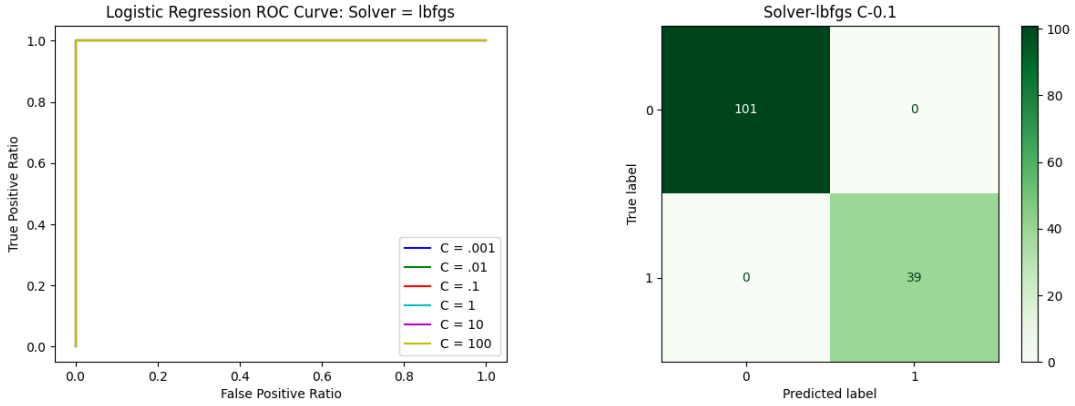


Figure 26: ROC curve and confusion matrix for Logistic regression using lbfgs.

Multi-Layer Perceptron

The mlp models I tried all learned the classification perfectly. The average confusion matrix of mlp models were: 101 true negative, 0 false positive, 0 false negative, and 39 true positive. The average true positive rate was 100%, false positive rate of 0%, an average ROC score of 1, and a average accuracy of 100%. The average time for the mlp models was .59 seconds.

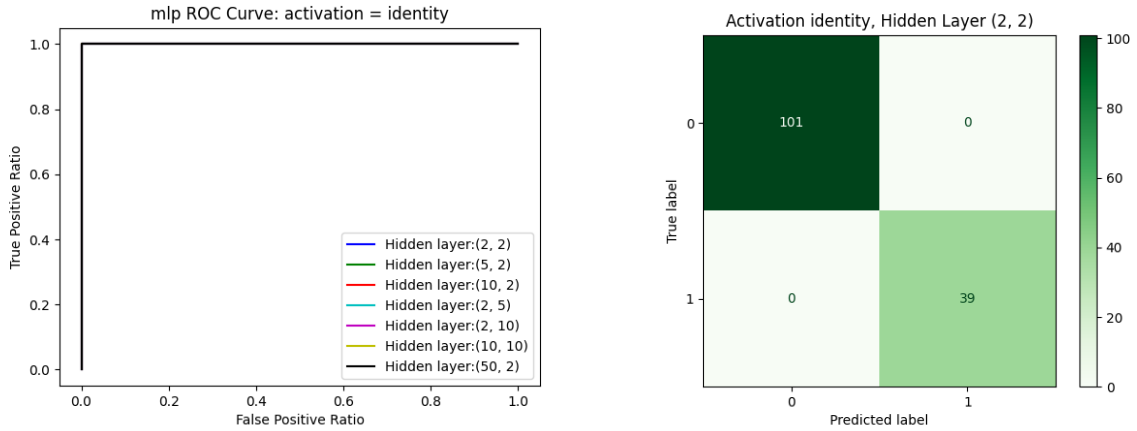


Figure 27: ROC curve and confusion matrix of mlp that learned the classifier to 100%

Algorithm Performance

King Rook Vs. King Pawn A7

The performance of the algorithms is night and day difference when the non-continuous feature was being treated as such. Many of the algorithms were in the 70-80% accuracy range and classifying the data fairly well. The decision tree classifier before hot shot can be seen in figure 5, and it worked at classifying the data at max settings. After hot shot it worked and actually could get 100% accuracy on the data at lower parameter settings. Bagging worked on 5 or more bags at max depth as can be seen in figure 8. After hotshot it improved to getting close to 100% accuracy at that same settings. The boosting worked with depth of 2 and estimators 30, 40 and 60 shown in figure 11, but like the others it could learn the classifier to 100% accuracy with the hotshot change. Knn can be seen as an okay classifier with a sharp ROC curve in figure 14, but it is not 100%. Hot shot improves knn on the KRKP data to be close to 100% accuracy and nearly a 1 area under the ROC curve. The logistic regression with higher C (C >= 1) values were good at classifying which can be seen in figure 17. Hotshot improves this classifier to 100%. MLP as a classifier was good even before that data got processed properly with the ROC in figure 19 showing off how good it was at classifying the data. It improves with hotshot to 100% capable.

All the classifiers with all sorts of setting had the capable to classify data from the KRKP data set to nearly 100%. The only differentiation stat I tracked would be model and classify time. Decision trees were the fastest and the others took 2 times as long. The results in the attached plots and the program output show how important it is to preprocess data properly. In the KRKP I messed up by using a non continuous feature as continuous messing with all the models.

Heart Failure Prediction

The algorithms all do very well when the features got reduced down to erum_creatinine and ejection_fraction. When the HFP had all the features running through, both the linear regression and mlp would not converge to the tolerance in a reasonable amount of steps. After the feature trimming to just 2, the algorithms converged quickly. Since most of them in this state could hit 100% accuracy the last sort of comparable stat to rank them by would be time to model and classify. On average the fastest classifier for the HFP data-set is the logistic regression at .501 seconds. This is followed closely by knn at .512 seconds. Theses two were the fastest and also the most accurate when you removed the logistic regression models that had a small C. Bagging did terribly with low depth and bag size. Log regression and MLP didn't converge with the whole data set, which forced preprocessing to select the best features or a lesser tolerance. Boosting performed well even at a low number of estimators, only having a slight cross validation standard deviation (2%) while training. Knn didn't show a big difference on time between the algorithms like the KRKP data set. The end result is that all the classifying algorithms can do the the job, but the data needs to be preprocessed so that they can correctly.

Program README

To run the pre hotshot version that outputs files to KRKP/{classifier}Plots

python3 chessClassifier.py

To run the hotshot version that out puts files to HotShot/{classifier}Plots

python3 hotshot.py

To run the heart failure data that outputs files to HFP/{classifier}Plots

python3 heartFailureClassifier.py

References

Data Sets

https://archive.ics.uci.edu/ml/datasets/Chess+%28King-Rook+vs.+King-Pawn%29

https://www.kaggle.com/andrewmvd/heart-failure-clinical-data

https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-020-1023-5

Classifiers

https://scikit-learn.org/stable/modules/neighbors.html

https://scikit-learn.org/stable/modules/tree.html

https://scikit-learn.org/stable/modules/ensemble.html

https://scikit-learn.org/stable/modules/neural_networks_supervised.html

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

Preprocessing

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.zscore.html

Metrics

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.plot_confusion_matrix.html

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.plot_precision_recall_curve.html