Supervised Learning Analysis

Decision Tree, Neural Net, adaBoosting, SVM, KNN

• UCI wine quality dataset [Cortez et al., 2009]

Description:

The original datasets from UCI include the dataset for white wine sample and the other dataset for red wine sample. I chose the white wine quality data set. There are 4898 wines is included in the csv file. For each instance, it has eleven attributes. Based on several of attributes are correlated, I need to apply some feature selection. Output variable: quality (score between 0 and 10). Input variables: (figure 1)

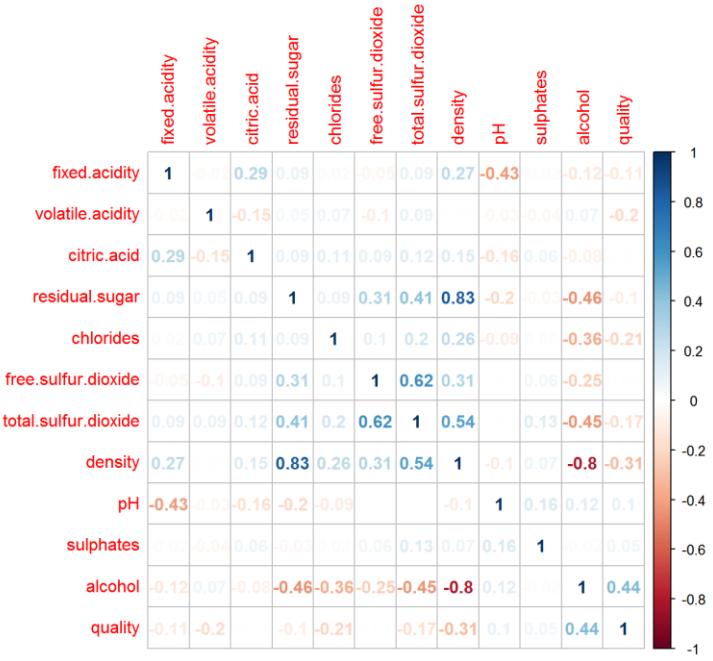
Figure1

1	2	3	4	5	6	7	8	9	10	11
Fixed acidity	Volatile acidity	Citric acid	Residual sugar	Chlorides	Free SO2	Total SO2	Density	pH	Sulphates	alcohol

To decrease the influence of correlations between attributes, I will run prelearning.py which is a program to decide correlations between two attributes. Once I decide which attributes correlated each other, I will only leave one of them. I chose this dataset for several reasons: 1) This dataset is provided by UCI machine learning repository. UCI made this for people who doing research on machine learning. For the dataset itself, it can be viewed as classification or regression tasks. I can expect that there exist some relations between the attributes and the result. 2) I personally feel this dataset is attractive. I've seen sommeliers could tell the quality of a wine. If I could run the machine learning algorithm to figure out which features are the most important factors, that would be a very achievable work.

Before running any learning algorithms, I did some preprocessing on my Wine dataset. The goal is to identify the wine quality based on other attributes. Since the original dataset has a quality scale from 1 to 10, it's hard to let the algorithm do any classification. The reason is that for any learning, we want to get a straight forward feedback like correct or wrong, good or bad, yes or no. The learning result would be better if I set all qualities greater than 5 is 1 and lower than or equal to 5 is 0. The number 0 stands for bad and

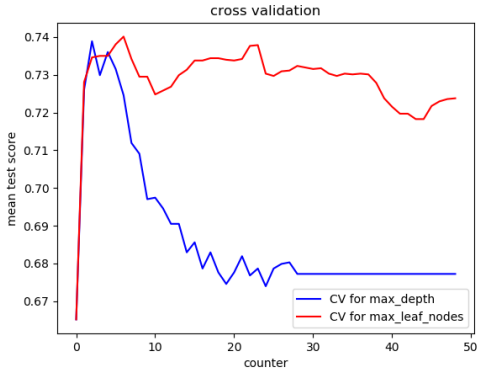
the number 1 stands for good. In this way, whenever I run sci-kit learn, say I input one set of attributes and also input a quality value which could be 0 or 1, then the algorithm could classify this set of attributes to good or bad. That's why it is better than score from 1 to 10 because it means that there could be ten types of qualities. Actually, the score from 1 to 10 also works if I have millions of data because I need that much data to train the algorithm well.



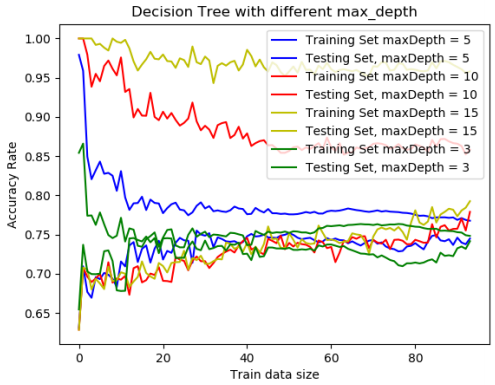
Then I need to test if there are correlations between each attribute. The reason is if there are two attributes have very high correlation, then we only need one of the attributes. For example, A and B has high correlation, we could say because A, then B or because B, then A. Now we want to learn C through A and B. If we run an algorithm with input A and B, this would cost more one classification or regression. I run the prelearning.py and calculating the correlations. Based on that, I would only choose alcohol, fixed.acidity, volatile,acidity, free.sulfur.dioxide, and sulphate as the learning inputs.

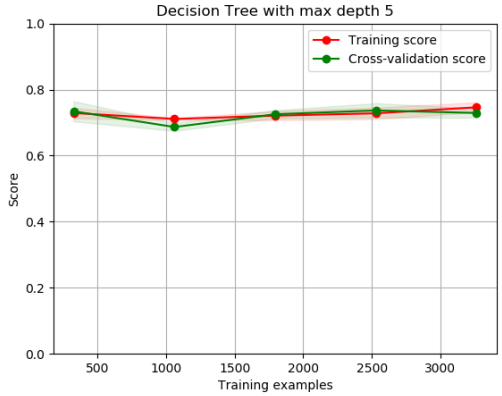
Decision Tree:

First, I run two cross validation for max depth and max leaf Nodes. I choose 'entropy' for criterion. Since this assignment hope students do some pruning on the tree, I basically implement the pruning by limiting the max depth and max leaf nodes. The goal is do not make the tree grow too big which might cause overfitting. Also, during cross validation, I initially split the training set and test set as 3:1, then I give cv = 3 in cross_validate(). In this way, I got the ideal max depth is 5, and max leaf nodes is 5, for best performance.



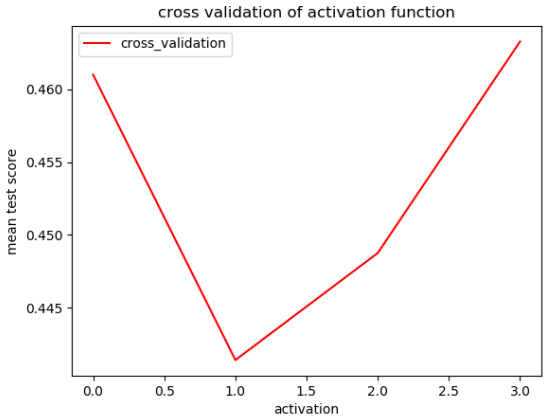
Then, I try different max_depth while defaulting the max_leaf_nodes and try different max_leaf_nodes with defaulting the max_depth. The two graphs show me that for max_leaf_nodes, it actually doesn't matter for 5 nodes, 10 nodes or 15 nodes. The reason could be for the wine dataset, the classification through each level, there are no more than 3 or 5 nodes required. So the max_leaf_nodes can do anything with pruning. On the other hand, I set max_depth to 3, 5, 10, and 15 could cost different test accuracy. If the max_depth gets too high, then it would be overfitting but if the max_depth gets too low, then it could be not enough learning. In this way, I will set depth as 5 which would be good for learning in decision tree. The accuracy score is: 0.7191836734693877 given maxDepth is 5 and test set size is 0.25.

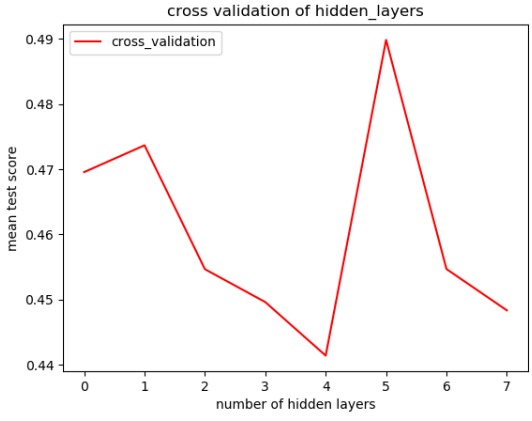


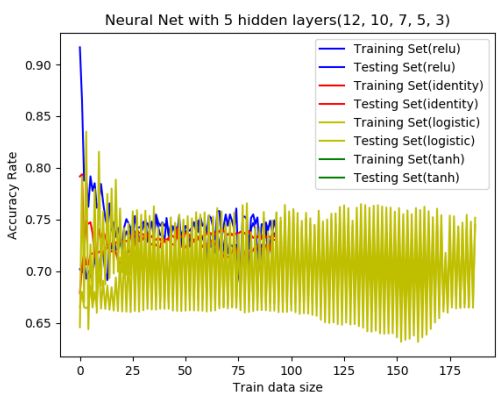


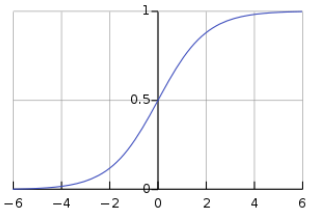
Neural Network:

Since the UCI dataset cannot be linearly classified, I need to add hidden layers. I am running different hidden layers: layers = (12), layers = (12, 9), layers = (12, 9, 6), layers = (12, 10, 8, 5), layers = (12, 10, 7, 5, 3), layers = (12, 10, 8, 6, 5, 3), layers = (12, 10, 8, 6, 5, 4, 3), layers = (12, 11, 9, 6, 5, 4, 3, 2). By doing this, I find the best layers for the UCI dataset. Than I tried different activation function('identity', 'logistic', 'tanh', 'relu') and and relu has better performance.

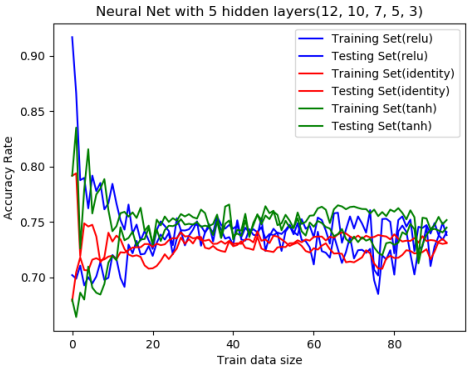




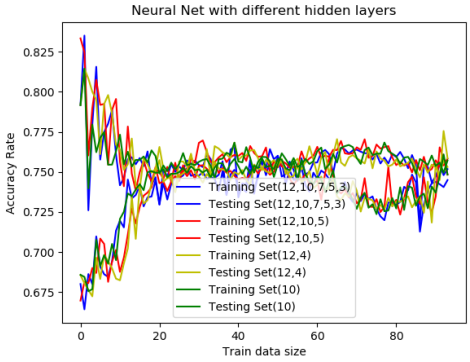




To improve my guessing, I run four types of activation functions while the hidden layers = (12, 10, 7, 5, 3). The reason that the graph using logistic acts like a crazy person is because logistic stands for sigmoid function. The function graph shows that sigmoid always gives positive activation no matter what the input to the perceptron is. In this way, any attributes of the wine would consider to be positive related to the quality. In this way, the learning result is unstable and unreliable. Other activation functions would give negative activation to the neural net and so that when we run the forwarding and back propagation algorithm, the weight for each perceptron will be more reasonable and trustable. So I run the same code again without logistic function to show the rest of three functions on Neural net.

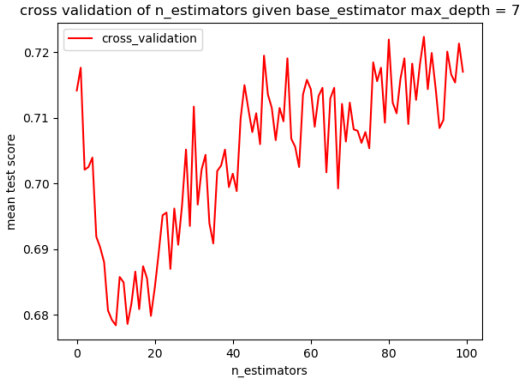


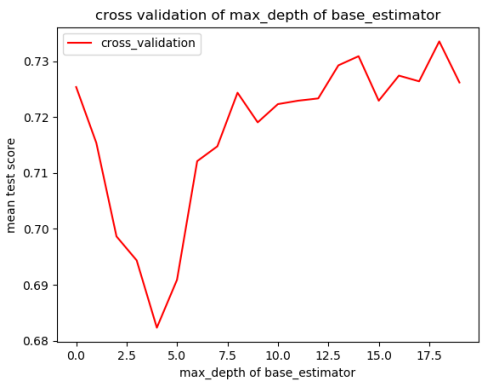
Given test set size is 0.25, I got accuracy of (relu) is 0.7257142857142858 accuracy of (identity) is 0.7216326530612245, and accuracy of (tanh) is 0.7346938775510204. At the end, I tried different hidden layers with activation function tanh. The curves show that increment of hidden layers doesn't better off the learning result. It could because for my six input attributes, a single hidden layer is enough to do the non-linear classification. If I have 20 inputs, the more hidden layers and more perceptron in each layer might be required because there are more needs on classification and on storing information in perceptron.



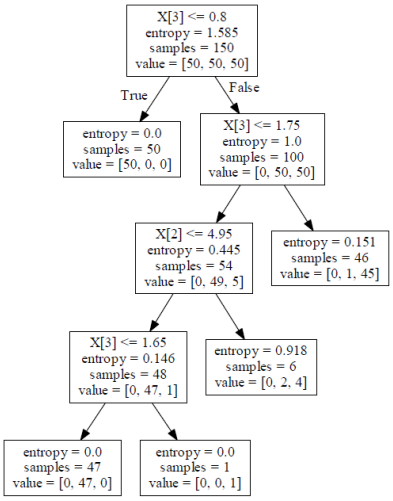
Boosted Decision Tree:

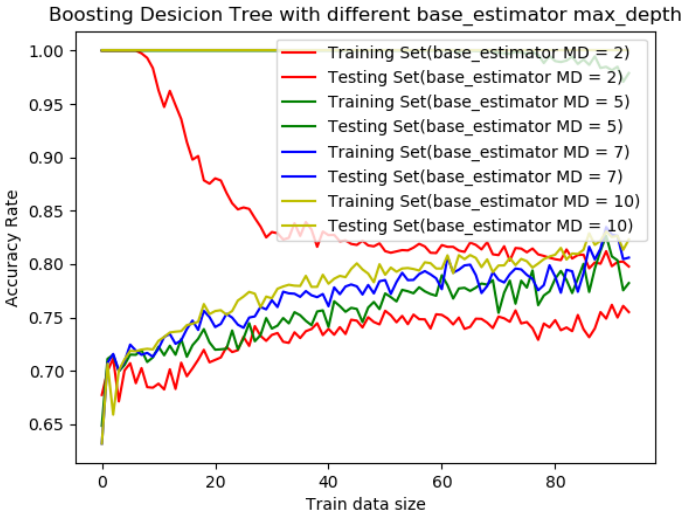
I need to identify a base estimator for the boosting. So I tried different max_depths of base estimator and defaulting the rest of parameters. Then I tried different n_estimators with max_depths of base estimator is 7. I got the two graph:





Honestly, from these two graphs, I couldn't really decide which base estimator and which n_estimator are good hyperparameters. Base estimator stands for each single tree in the forest, and n_estimator stands for the amount of those trees in the forest. Before I run adaBoost, I would make some explanation regarding the two about graphs. For adaBoost, if each base estimator has very large max depth, then each single tree in the whole forest could be almost singly functional as a decision tree. I draw one of the base estimator. Each with this base estimator, the accuracy could be around 0.7109. Say if I create 30 of them, the cross validation score doesn't increase a lot because a few trees would be already good enough for learning. But I still need to run train set and test set the see more information.





From here I also calculated the accuracy rate: (MD: maxDepth of base estimator)

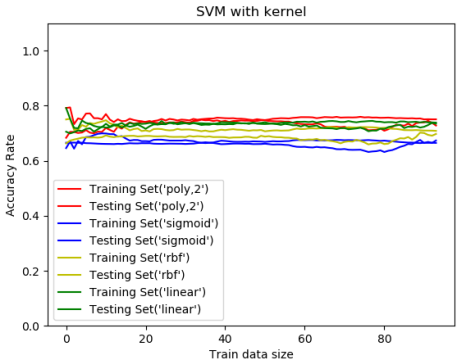
MD = 2	0.7444897959183674
MD = 5	0.7665306122448979
MD = 7	0.7820408163265307
MD = 10	0.8073469387755102

So I would have the test accuracy score boosting to 0.78122 if I set the max depth of base estimator to 7 and defaulting the n_estimator. Then I tried different n_estimators give MD = 7.

So give n_estimator = 55 and MD = 7, the accuracy boosts to 0.7975 which round to 0.8. The result of final training and testing is kinda different from cross validation score. Result could be I am using the entire training set to train the algorithm and testing it which gives me a higher accuracy rate, while I used k-folder training subsets doing the cross validation so that the algorithm could may not be trained well. Now Comparing the adaBoost

Decision tree and the normal decision tree. adaBoost gets 0.8 accuracy while the decision tree only gets around 0.7 accuracy. The result actually matches my expectation. Boosting decision tree almost always has a higher accuracy than decision tree. One way to look at it is that adaBoosting would make many small trees instead of the normal decision tree which only generate one single huge tree. AdaBoosting will build a better model and can cover more cases through bagging and Random forest. For adaBoosting, if an input is misclassified by a hypothesis**, but by combining the whole forest and convert the weak learners, the final model will have a better performance** while the decision tree will just keep increasing the weight of incorrect hypothesis.

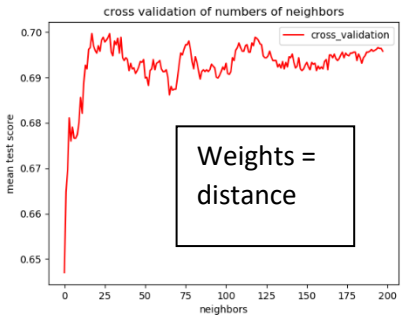
SVM:

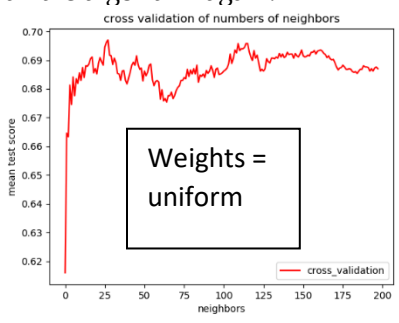


I started with running SVM with different kernel functions. From the graph, generally the algorithm has a better performance using poly kernel, has the accuracy at 0.720816 given test set size is 0.25. Then I want to try different degrees given kernel is poly. I tried to give degree as 3 or 4 but the calculation would take too long. The reason could be the classifying is too complexed. For linear kernel, it generates a linear model but the performance is almost as good as poly kernel. One reason could be the inputs is linearly separatable because the accuracy is very close to poly kernel with degree of 2. RBF uses normal curves around data points and sums them to make the boundary. One thing needs to be mention is that for SVM, the train set and test set has very close performance so that there is less overfitting.

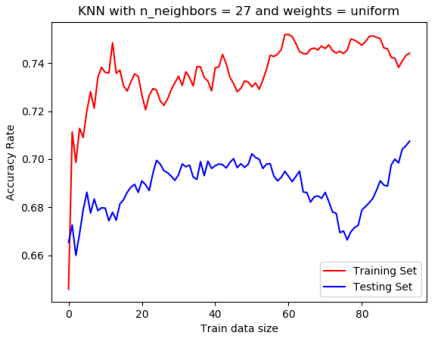
KNN:

Same as the previous four, I start with cross validation for n_neightbors and p value. P1 stands for manhattan distance which has the best performance in cross validation. At the same time, the algorithm has thebest performance when there are 25 neighbors allowed. Then I take the two value to run the algorithm again.

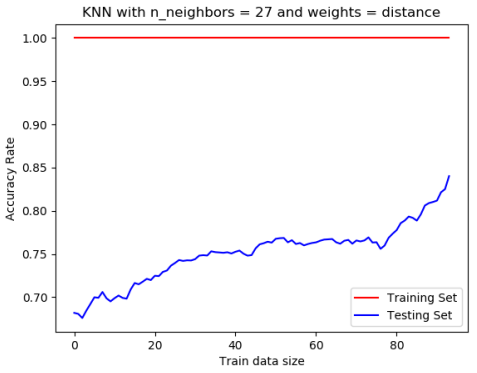




So the variance of using distance is slightly higher than the variance of using uniform. Weights stands for the weight function for all points in each neighborhood.



Comparing KNN with SVM: Since I have six inputs which means the plots in a high dimensional space so that KNN did poorer job than SVM since KNN doing better in low dimensional space. And also as what we get from SVM, the wine dataset could somehow to be classified linearly and since KNN is non-linear classifier so that this may also cost KNN's low performance.



At the end I also train the algorithm with the whole train dataset and test the model with test dataset while I used weights = uniform and weights = distance. The general performance while using distance is really good but it apparently is overfitting because the train accuracy is 100% and the model might be only used in this wine dataset.

• Heart Disease UCI

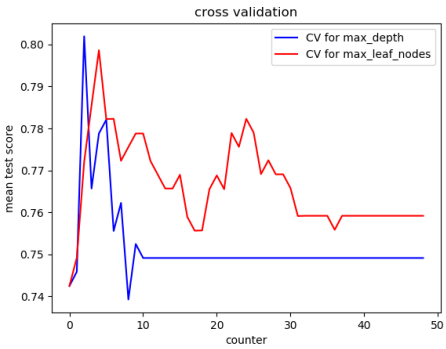
Description:

The original dataset contains 76 attributes but usually people only refer a subset of 14 of them. Attribute information includes: age, sex, chest pain type with 4 different types stands by (0,1,2,3), resting blood pressure, serum cholestoral in mg/dl, fasting blood sugar, resting electrocardiographic results with (values 0,1,2), max heart rate achieved, exercise induced angina, old peak, oldpeak, the slope of the peak exercise ST segment number of major vessels (0-3) colored by flourosopy, thal: (3 = normal; 6 = fixed defect; 7 = reversable defect) I choose this dataset as my second dataset is because I think it's pretty cool to train a model and predict if a person might have heart disease. The Heart Disease dataset is a classification problem.

Preprocessing:

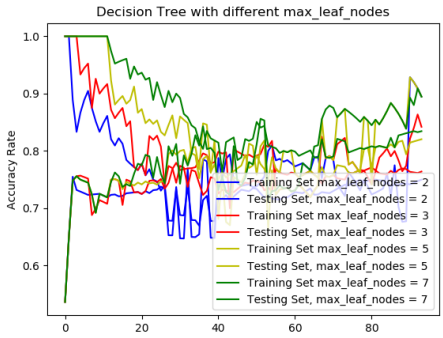
The original dataset I download from Kaggle groups all heart disease tuples together and groups all health tuples together so that if I do not shuffle the rows, then the training will be not as good as I want it to be.

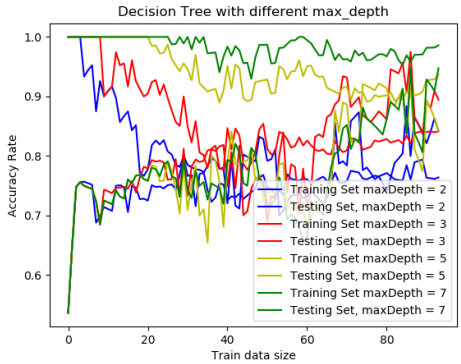
After I shuffle the dataset, I make a experiment with decision tree model. I increase the accuracy from 0.7894736842105263 to 0.8947368421052632 because the dataset is evenly distributed. Unlike the wine quality dataset, which I only use 6 out of 10 attributes for training, the heart disease dataset requires all attribute for training propose because the model should have the ability to detect heart disease as much as possible.



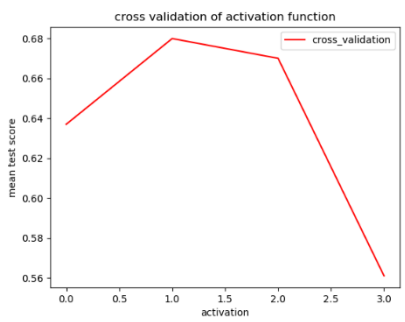
Decision Tree:

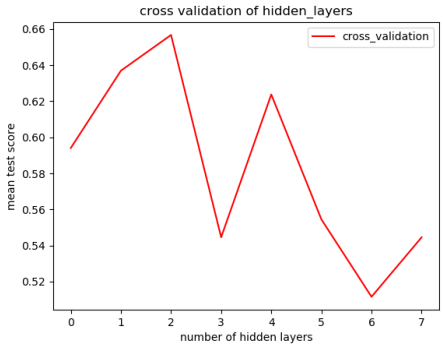
After running cross validation on max_Depth and maxLeaf_Nodes, I got the graph. So I would choose max_Depth as 3 and maxLeafNodes as 4. Then I train the algorithm with entire training set and test with testing set. During training with different max_Depths, I will default max_Leaf_Nodes, and do the same when I train wit different max_Leaf_Nodes. The accuracy difference for max_Depth = 5 is smallest and also accuracy is around 0.9. For max leaf nodes, the algorithm has a better performance when the max_leaf_nodes is 7. For the heart disease dataset, several attributes have different types such as thal has (3 = normal; 6 = fixed defect; 7 = reversable defect), and chest pain type has (1,2, 3, 4). Compare with the wine quality dataset, the heart disease dataset has more complex classifications. Also because it has more input attributes. So if I use max_depth as 5 and max_leaf_nodes as 7, with test set size of 0.25, I will have the accuracy 0.868421052631579. However, with the rest remine same, I set max_leaf_nodes as None, I would have the accuracy 0.9210526315789473. However, without the limit of max_leaf_nodes, the tree could be grow to extremely unbalanced and will cost overfitting. In this way, 0.86 could be a fair enough good performance.



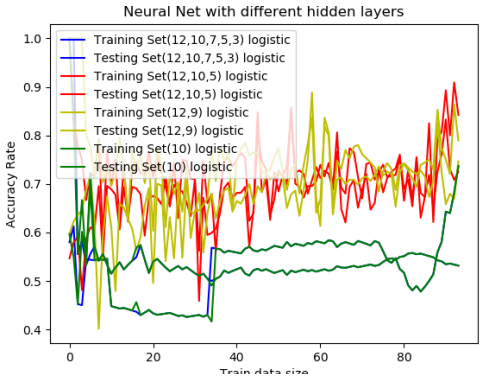


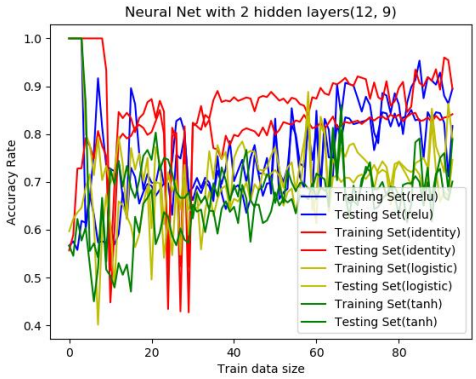
Neural Net:





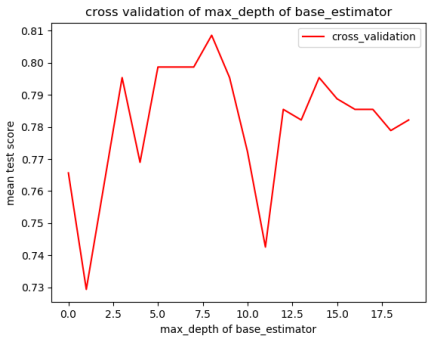
I start with testing different activation functions and different hidden layers. Different from the wine quality dataset, I got the different graphs. For the activation function graph, 0 1 2 3 stands for 'identity', 'logistic', 'tanh', 'relu'. So logistic gets the higher accuracy. The logistic stands for sigmoid function. The reason could because sigmoid always gives some feedback from 0 to 1 no matter what the input is. So that the learning would be quick. For different hidden layers, two hidden layers (12, 9) has the best performance. One reason could be, in cross validation, every training set in n-folder is very small compare with the actually training set. Since the total dataset is about 400 tuples. In this way, for those multi hidden layers, they might not be able to be trained well so that the performance is quite poor. Then given activation as logistic, I run different hidden layers with the entire training set, and hidden layer (12, 9) still gets a pretty good performance. However, if I use hidden layers (12, 9) run different activations, identity would have a good performance. The learning result with (12,9) is kind different from cross validation because I am using the entire training set. Also you can think of because the heart disease dataset is a small dataset instead of wine quality with over 4000 tuples. In spite of this, I still choose (12, 9) and identity to calculate the accuracy: 0.9078947368421053. So with activation as identity, it's a linear function (y = x). The high accuracy might be cost by overfitting or because the problem is linearly classifiable. However, think of the volume of the dataset, it's more likely that the problem is linearly classifiable and that's why the linear activation provides the best performance.

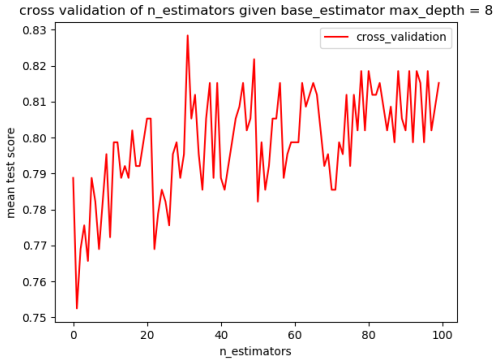




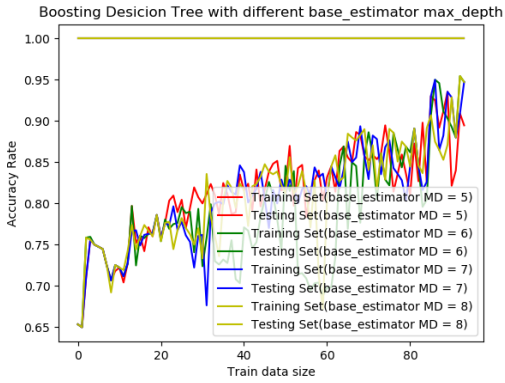
Boosting Decision Tree(adaBoost):

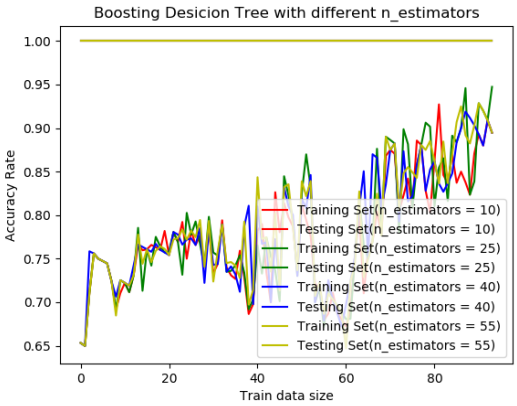
At the beginning, I use cross validation to explore the max_depth of the base estimator. And I choose max_depth as 8, run cross validation for n estimators.





The running result shows that max_depth of base_estimator from 5 to 9 provides a good performance. However, n_estimators after 30 basically gives a good performance. To find the better hyperparameters, I run two tests using the entire training set. So '7' cannot do because the variance is too big but 8 looks fine. While the max_depth is 8, try different n_estimator. The variance of (n_estimator = 55) is smallest. So Give that, I calculate the accuracy 0.9078947368421053. In this time, the boosting decision tree didn't do better than the normal decision tree. The first reason I can think of is the volume of the dataset. Since adaBoost will build a forest and combine the trees to form a model, it always require a lot of data to train. Given a small size dataset, the model could be not trained enough. At the same time, the decision tree only generate one tree which fairly works with this size of dataset.

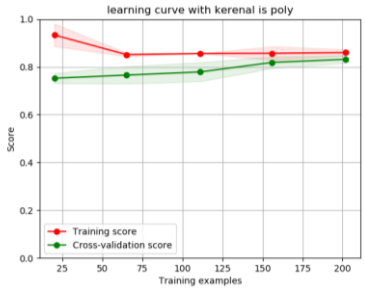


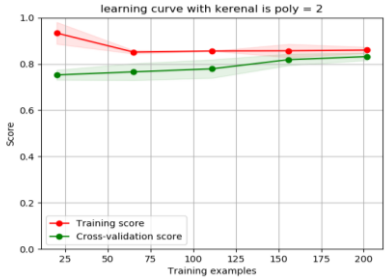


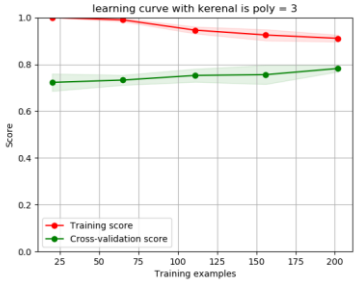
SVM:

I decide to try something new at the one. So I start from calculate the accuracy rate with different kernels while the test set size is 0.25. From the table, clearly that the heart disease dataset should be treated as a linearly classifiable question, which also proved the former assumption in Neural Net. Now, I would just focus on the situation when the kernel is linear and when the kernel is poly. I want to find the optimal kernel function. Instead of using cross validation, I just using plot_learning_curve. From the learning curve, there three options actually have the very similar performance. In this way, the accuracy given test set size of

Poly(degree = 2)	0.881578947368421
Sigmoid	0.5789473684210527
Rbf	0.618421052631579
llinear	0.881578947368421



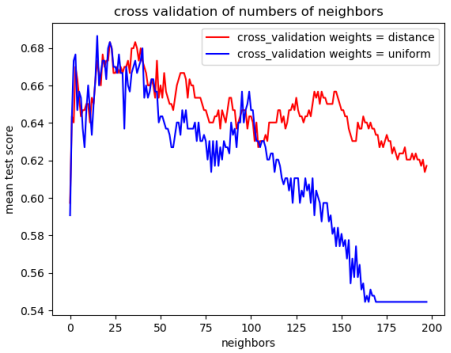


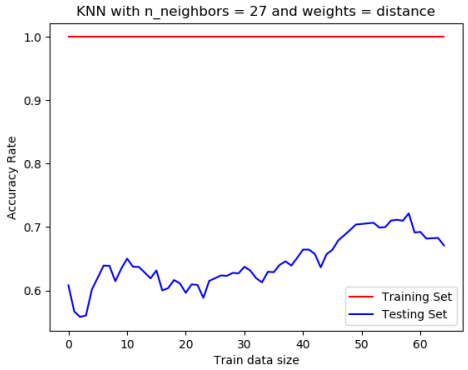


0.25 and kernel using poly = 2 is around 0.88. It's a fine performance compare to the best accuracy(0.92) that decision tree can achieve. Without the limitation of max_depth, the decision tree can have 0.92 accuracy. But I aim to mainly compare SVM with KNN so that I will run KNN first then make some analysis using SVM and KNN.

KNN:

As usual, I start with running cross validation. As we see the accuracy is slightly higher when the weights = distance. Then I would choose n_neighbors = 27. Using these two hyperparameters, I run the learning algorithm again with entire training set. So the accuracy in the best performance is around 0.7. Now,





KNN has the accuracy significantly lower than SVM. One way to look at it is that KNN is non-linear classifier so that it always preforms better regrading to the non-linear classification problems. Since I choose linear kernel function in SVM so that SVM has a higher accuracy. Actually, for rbf and sigmoid kernel functions in SVM, it has the similar accuracy with KNN.