Programming Environment:

All machine learning algorithms I used were in sklearn Python package. The dependent packages include numpy, pandas, matplotlib and graphviz. And the python version I used is 3.6.3.

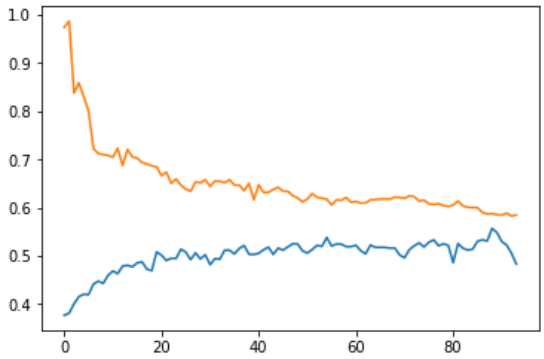
First dataset:

The first data set I chose is the UCI wine quality dataset from Kaggle. This data set consist 3918 wines of which the quality scores ranging from 3 to 9. And there are 11 features for each of those wines, some features like pH, density and alcohol are easy to understand. But for the features related to acidity, there are fixed acidity, volatile acidity, and citric acidity.

I'm very interested about what's the more important acidity feature and what other features influence wine quality most. Also, because this is a multi-class classification problem, I can expect the accuracy of most of the classifier won't be very high.

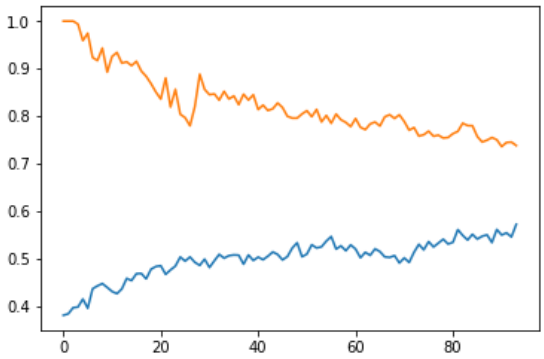
UCI Wine quality:

For the **Decision Tree classifier**, because there is no pruning feature available in sklearn's decision tree function, I use the maximum tree depth parameter as an alternative. After plotting the learning curves of DTs with different maximum depth (Fig. 1-2). The learning curve of depth 5 seems to generalize best and gave an optimal accuracy of around 0.54.



Training and Testing accuracy

Fig. 1 The learning curve of DT with depth 5



Training and Testing accuracy

Fig. 2 The learning curve of DT with depth 10

Also, when I compare Gini index and Entropy as the quality function, Gini index provides a much better accuracy rate at smaller tree depth (1-8). Because I already decided to use a relatively small tree with depth 5, Gini index is obviously the better-quality function.

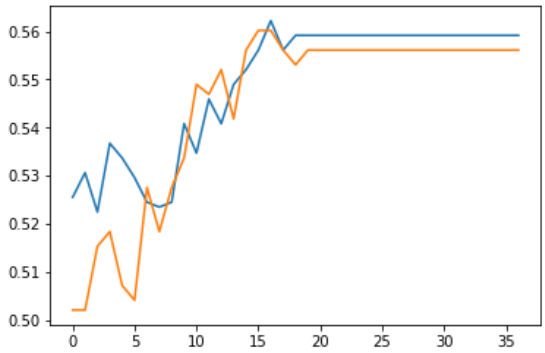


Fig. 3 Compare Gini and Entropy

I also plotted a detail tree structure with depth 3. So that I can find some insights of the different wine features. It's very constant that alcohol level is used by the very first node. Meaning that alcohol level fits with wine qualities very well. And then volatile acidity seems to be the most important acidity feature.

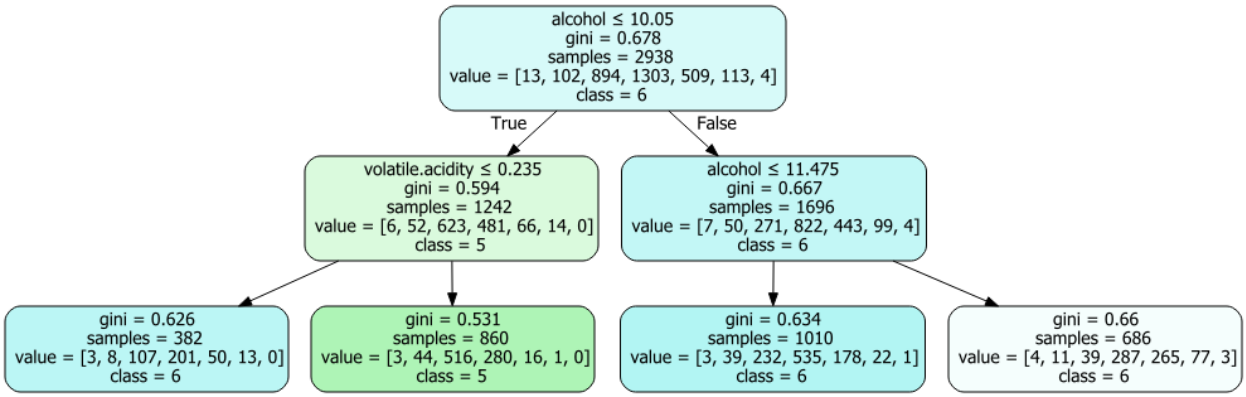


Fig.4 Decision tree of maximum depth 2.

For the **Neural Network classifier**, I used the Multi-Level Perceptron classifier in sklearn. I tried to add 2 (5, 2) to 3 (20, 5, 2) hidden layers, the prediction accuracy didn't improve comparing to 1 hidden layer (5). And the more perceptron I placed on this layer, the accuracy didn't improve along the way. So I chose to use 5 perceptron in one layer, logistic sigmoid activation function and gradient descent solver as the optimal with around 0.46 accuracy.

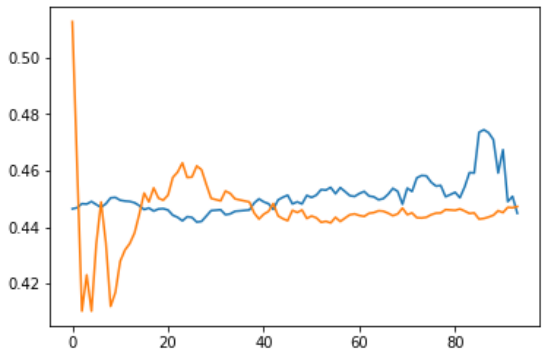


Fig.5 Learning curve of MLP(5,2)

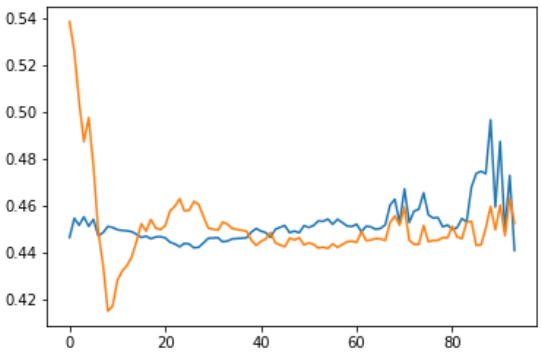


Fig.6 Learning curve of MLP(5)

For the **Boosted Decision Tree classifier**, I'm using the adaBoost classifier in sklearn. Because with the number of weak estimators increases, the testing error rate will decrease for sure, so I arbitrarily chose a n_estimators of 100. Because there is no parameter of maximum tree depth, it's not possible to do a tree pruning. Also the error rates are even higher than basic decision trees.

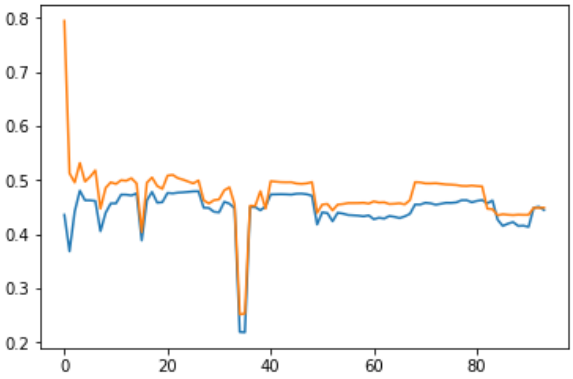


Fig.7 BDT learning curve of 100 estimators

For the **SVM classifier**, I used the C-Support Vector Classification function in sklearn. This function is a wrapper of libsvm package. I tried to use Linear, RBF and Polynomial kernels. The Polynomial kernel can't be fitted within 15 minutes, so I stopped the process and chose RBF kernel since it has a much better prediction accuracy. When I run the learning curve using linear kernel, it also can't finish within 15 minutes and the accuracy score is lower than RBF kernel.

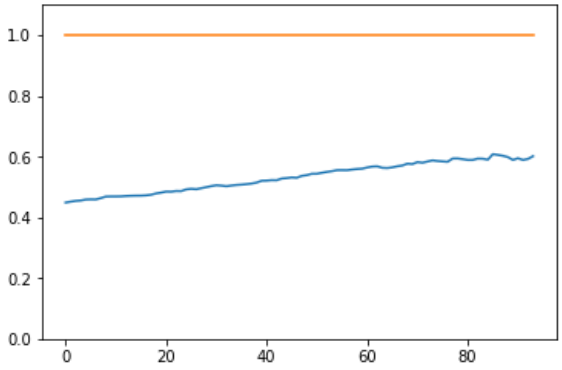


Fig.8 SVM learning curve using RBF kernel

For the **KNN classifier**, I tried K from 1 to 25. The accuracy scores kept increasing along the way. Even if I use 5-fold cross validation, the trend of increasing didn't change. And I tried to plot a learning curve with K=13, the training data accuracy is always 1 (as is expected for an instance based learning method) while testing data accuracy increases. I just chose to use K=13 and the 'optimal' accuracy score is around 0.54.

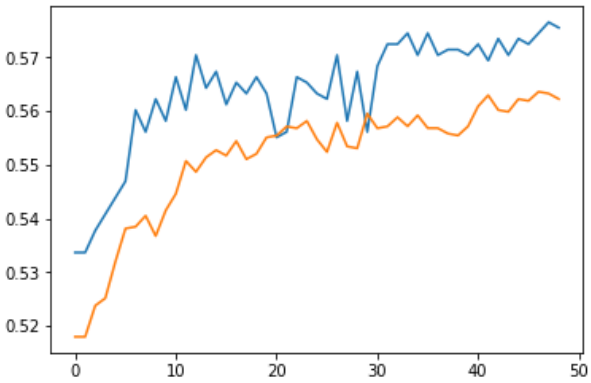


Fig.9 KNN accuracy for different K (Cross Vali)

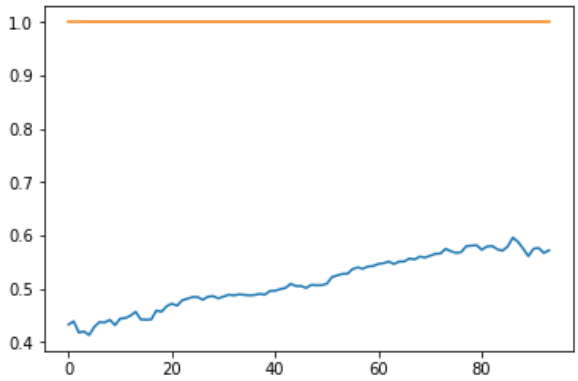


Fig.10 Learning curve of K=13

To sum things up.

1. Why did you get the results you did? Because I'm not familiar with all the possible parameters with some classifiers. The learning curves of some classification doesn't make much sense, which means I really need to do a grid search of a lot of parameters to figure out a reasonable set of parameters. And maybe need to use some other learning curve functions other than plot my own ones.

2. How fast were they in terms of wall clock time? For most of the classifiers, the learning process can be finished within seconds. Only SVM with polynomial kernel and boosted decision tree with a large n_estimators (>200) have higher than 15 minutes run times.

3. Would cross validation help? For most of the learning process, I can expect that cross validation will provide more accurate testing error rate. But after tried running CV on ANN and SVM classifiers, I found the training time would be more than 15 minutes for my data set. Thus, I just used a 3:1 training/testing data set for most classifiers. If I were to have a smaller data set, I'll use CV for certain.

4. How much performance was due to the problems you chose? Since I used a 7 class learning problem. It's very much expected that most error rates are very high. But I think an accuracy of 0.60 is okay for such a data set.

5. Which algorithm performed best? How do you define best? I just compare the model accuracy of testing set based on 3/1 training/testing ration. The KNN classification with relative high K (>100) can gave a 0.6 accuracy score and the KNN method is also very easy to understand. It also worth mention that decision tree classifier can provide more insights of the features, we can get some information about alcohol or acidity features of wines. And that is useful for real world learning problems.

Second dataset:

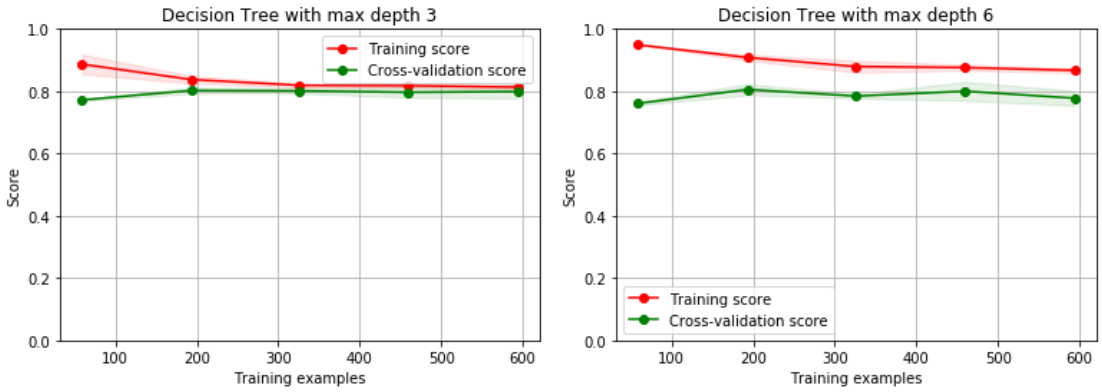
The second dataset I have is the Titanic survival data set from Kaggle. This is a smaller set with 891 person's information and whether they survive the crash or not. The features include ticket class, sex, age, family size, passenger fare, cabin number and port of embarkation. And the goal is to learn a model that can predict the survived or not information best.

Another point worth mention is that the data set I used is one that has been preprocessed from the raw data, so all feature were transformed into numeric classes and NA has been treated as one class together.

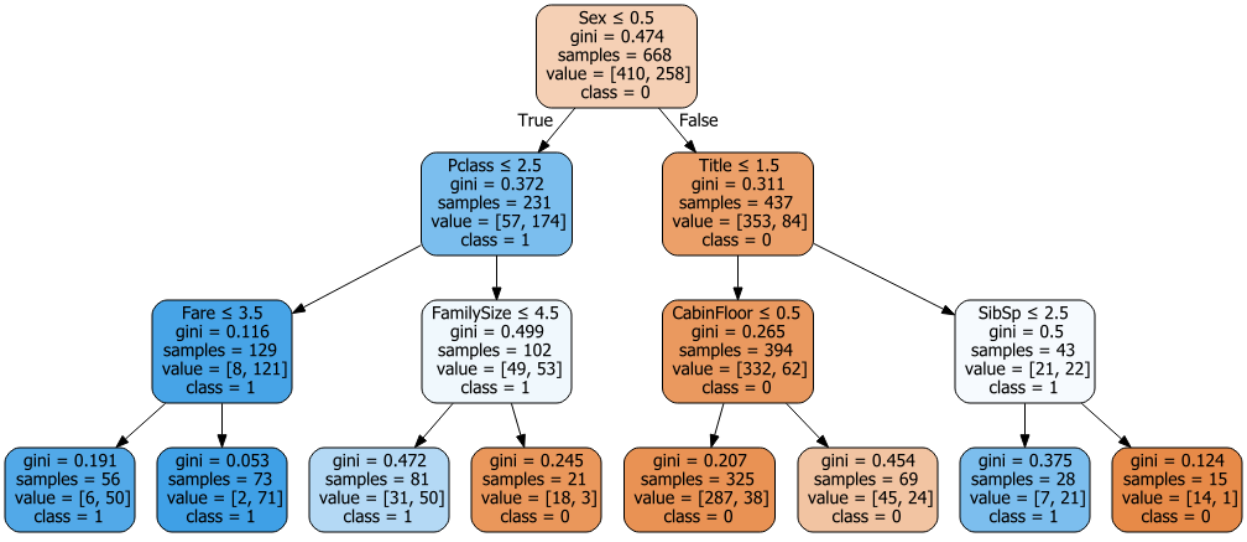
There are two reasons why I chose this simpler data set. First, it's smaller size enable me to do cross validation and more iterations on some algorithms, which means I can get more accurate testing error rates and better models. Second, a two class classification can have much lower error rates, thus all learning curves should look much better.

Titanic survival:

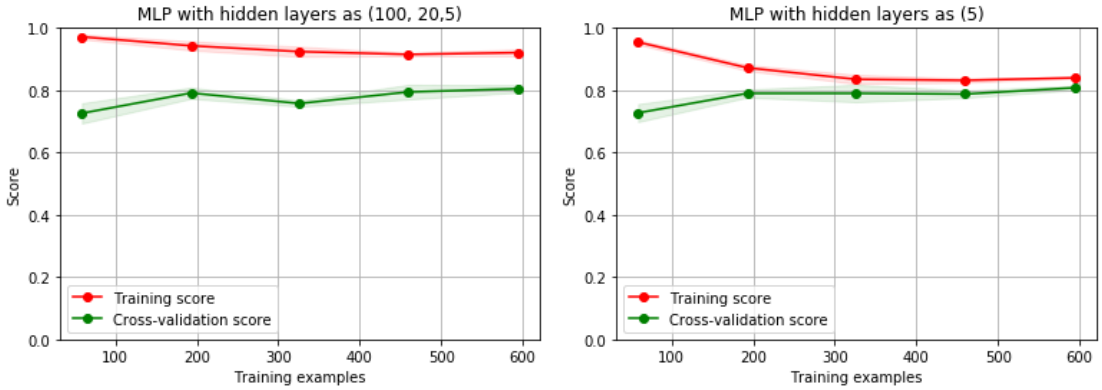
For the **Decision Tree classifier**, I used the same classifier as before. After looking at the learning curves of different tree depths, a simple tree with depth 3 has the best curve shape. The training and testing accuracy nearly come together at around 0.81.



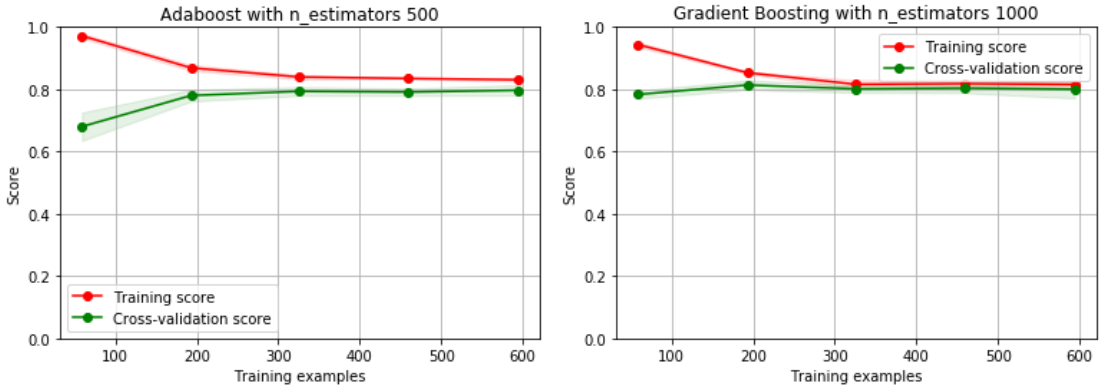
If we take a look at the detail tree structure at depth 3, there are also more information I can learn about the data. The features that fits survival best is sex, ticket class and title. The first two features make sense. As for the title information, it's compiled from the name column and only have two classes for normal (Mr. Mrs, Miss etc.) and well educated (Master Dr etc.). So, this seems to show that for male passengers, the well educated ones have a much high chance of being rescued. I can also do a pruning to the tree to simplify some leaves with the same class, but because my tree is small enough and pretty accurate, I just leave it as is.



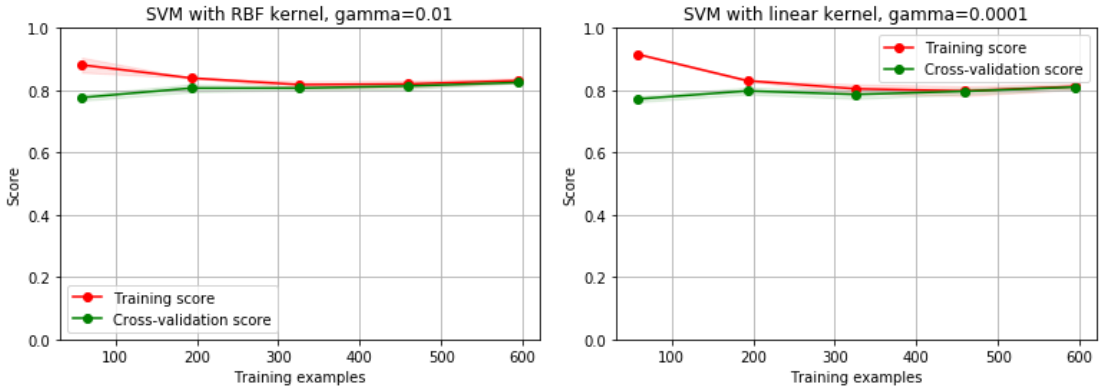
For **Neural Network classifier**, I used the same MLPclassifier function in sklearn. For the solver part, I found the quasi-Newton method converges faster than the basic gradient descent method, it also produces better accuracy score. As for the hidden layers, a complex setup of (100, 20, 5) will introduce too much variance in the model, then the testing score can't reach near the training score. A simple (5) perceptron seems to be sufficient for this data set.



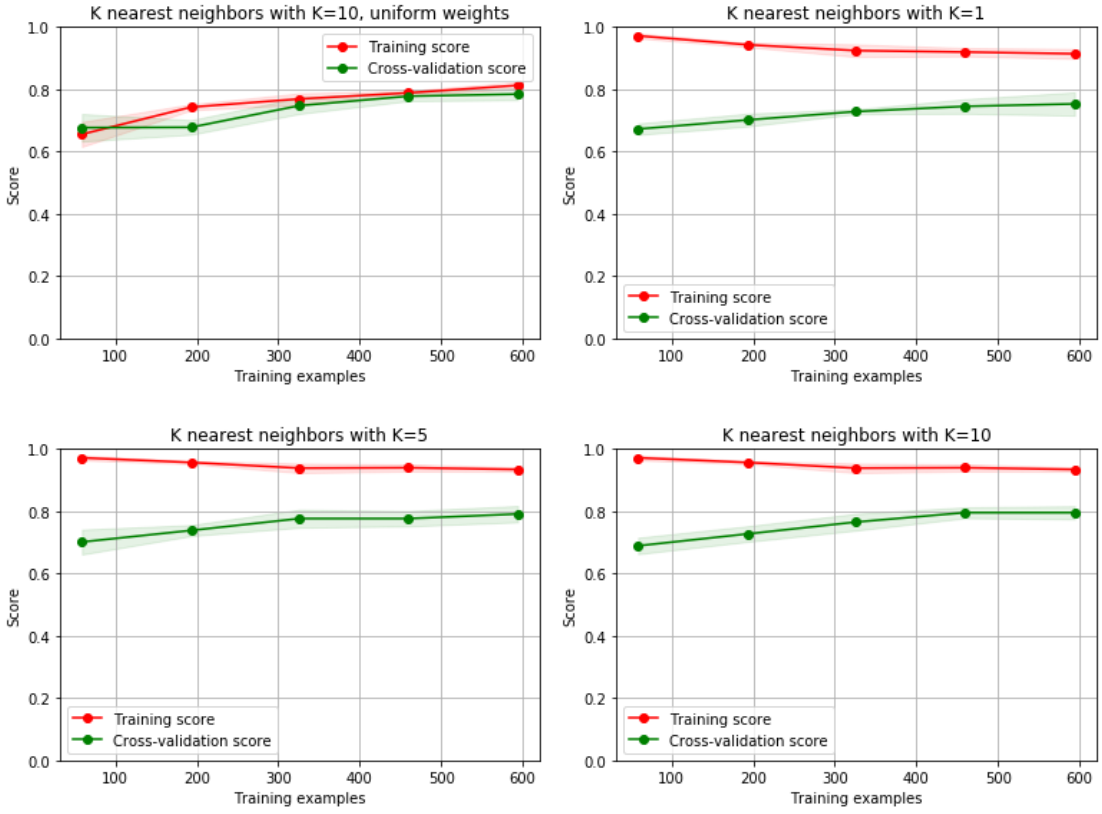
For the **Boosted DT classifier**, I compared the results of AdaBoost and GradientBoosting classifiers. The Adaboost converges quite slowly and doesn't provide any control over the tree structure, thus the testing score is always smaller than training score due to overfitting. As for Gradient boosting classifier, it supports the change of tree size and leaf nodes amount. Thus, I can control the tree structure similarly to doing pruning. After setting the maximum tree depth to 3 and maximum leaf nodes to 5, the learning curve converges very well.



For the **SVM classifier**, because there are multiple hyperparameters in the model. I did a grid search of kernel, penalty, and gamma to select a better model. The RBF model with C=10 and gamma=0.01 gave a very similar performance of another optimal linear model. Also, for the polynomial kernel, I can't finish any fitting with a C larger than 1, so it's not possible to do a grid search on that.



For the **KNN classifier**, if we use uniform weights for all the K points, the model is underfitted because of too small training score. After changing to distance weighted K point weights, the learning curve makes more sense. And with a larger K=10, the testing score improved to around 0.80. But the models always look to be overfitted. Also, all distance metric is Euclidean distance because Manhattan distance doesn't give meaningful learning curves here.



To sum things up:

1. Why did you get the results you did? This simpler classification data set allows better model fitting and obviously better testing accuracy.

2. How fast were they in terms of wall clock time? For most of the classifiers, the learning process can be finished within seconds. Only SVM with polynomial kernel and boosted decision tree with a large n_estimators (>200) have higher than 5 minutes run times.

3. Would cross validation help? After change to use cross validation for all my learning processes, not only the learning curves looks smoother, the final testing scores are also more accurate. Given a data set with a feasible size, cross validation will certainly help the learning.

4. How much performance was due to the problems you chose? This is a simple two class classification problem. And we can imagine many features should be related to survival, like sex, fare class etc. My testing accuracy is at most 0.81, but I can't find a model that can perform better, so I consider that's due to data set or maybe the preprocessing methods.

5. Which algorithm performed best? How do you define best? In general, SVM and Gradient Boosted DT provided the highest cross validation scores of around 0.81. Training set scores also converge to testing scores very well, that's another way to identify good models I used. If I take run time into consideration, SVM model with RBF kernel is faster than boosted trees. So I'll take SVM as the best algorithm for this data set.