CS 7641 Assignment 1

Student: Yinglin Li (yli973) Email: allamli@gatech.edu

1. Data Description

1.1. Description of Dataset 1 Digits Recognition:

This dataset was made by NIST dataset (a large written digits dataset) through extracting the normalized bitmaps of handwritten digits from a preprinted form. This dataset contains 5620 instances (3822 training instances and 1798 test instances) and each instance has 8*8=64 attributes with an input of the integers between 0 to 16. The labels of each instance are 0 to 9 which means the corresponding written digits.

1.2. Description of Dataset 2 Magic Gamma Telescope:

The data are MC generated (see below) to simulate registration of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using the imaging technique. So it was somehow to describe the features of the image. The dataset has totally 19020 instances with 11 attributes of real number as input. The labels were 'g' = gamma particle and 'h' = hadron particle.

2. Overview

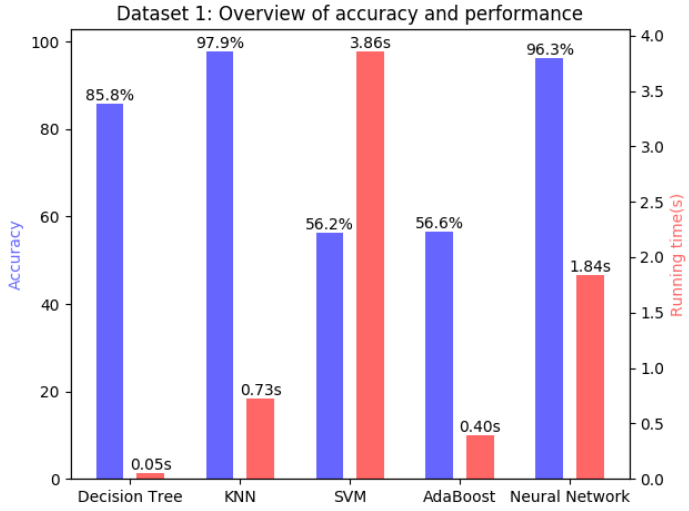

Fig 2.1. Overview of dataset 1

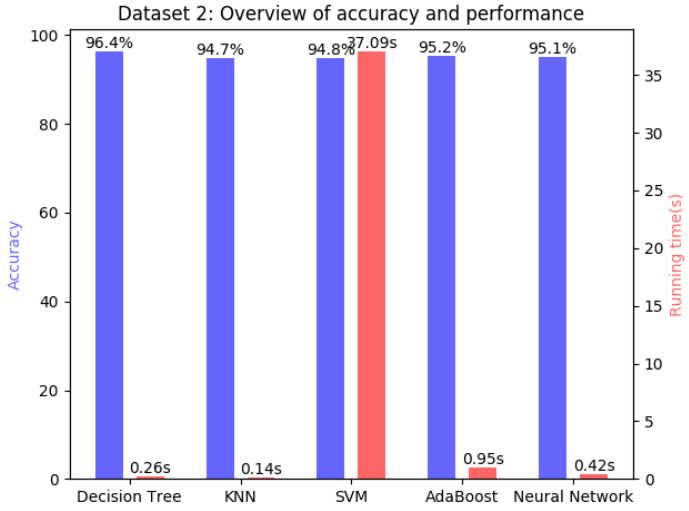

Fig 2.2. Overview of dataset 2

The running environment was with Intel Core i7-6700HQ CPU in scikit-learn of python. For each kind of classifiers, I used the default setting.

In dataset 1, we could see that KNN and Neural Network had the best result with an excellent accuracy and a not bad running time. Decision Tree had the best running time and a medium accuracy. However, the accuracy of SVM and AdaBoost was very bad. Especially with SVM, since there were 64 features of each instance, SVM got a bad performance.

In dataset 2, we could see that all the algorithm could got about 95% of accuracy thanks to the large amount of training instances. However, there were so much training data that SVM's running time was still very slow even if the training instances only had 11 features and 2 kinds of labels. Moreover, in KNN, we could see its performance was better than the performance of dataset 1. That was because KNN's running complexity was related to training instances and its features number.

Detailed tests and experiments of each algorithm will be discussed in each section below.

3. Models Analysis

3.1. Decision Tree:

3.1.1. Bias and Variance:

For each set of training dataset, it would build up a whole different decision tree. So it would cause a great variance. Then, since we built the decision trees by splitting the data into different sets so the decision trees prefer a shorter tree with a good split at the start and the correct answers. But however, decision trees only care about what can be represented by the tree and split by the boolean value.

3.1.2. Model Analysis:

Here we used max depth in x-axis as the complexity measurement. The light green area around the CV curve was the standard deviation of the cross validation, which could represent the variance. And scikit-learn uses an optimized version of the CART algorithm

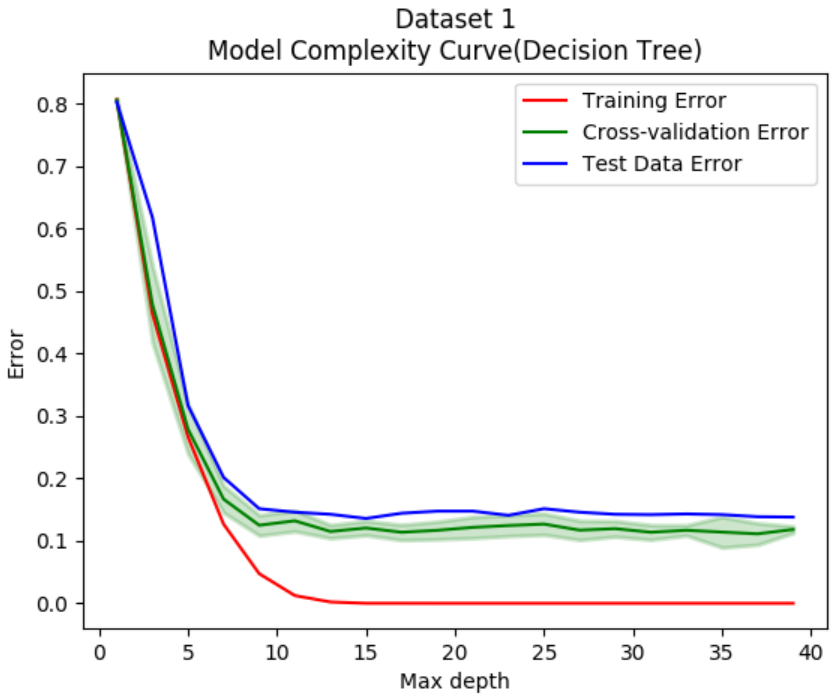

Fig 3.1.1. Model Complexity Curve of decision tree in dataset 1

In dataset 1, we can see when max depth equals to about 13, the error of training and CV(Cross-Validation) didn't decrease any more so the best model should be with the max depth of 13. However, we couldn't see overfitting point here. It was because there were not enough examples for 64 features of each instance to build a very complex trees. However, the test data error was high due to too many test cases, where "many" means the comparison to the training data.

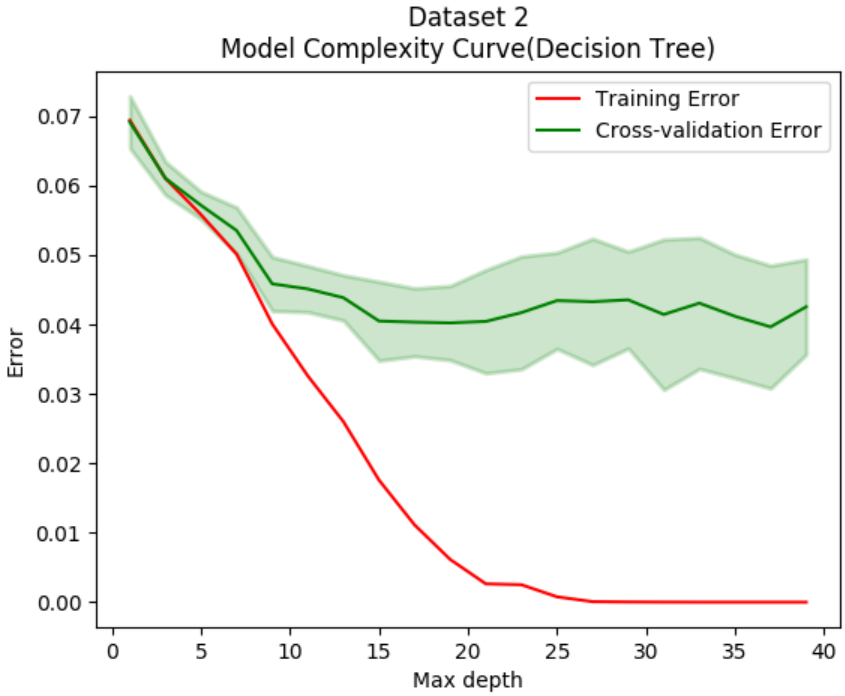

Fig 3.1.2. Model Complexity Curve of decision tree in dataset 2

In dataset 2, we didn't see overfitting point either. That was because the features of dataset 2 really fitted to split the decision trees according to the low error (high accuracy). Since we could see after max depth of 20, both the training error and CV error became stable. So max depth = 20 would be the best model for this dataset.

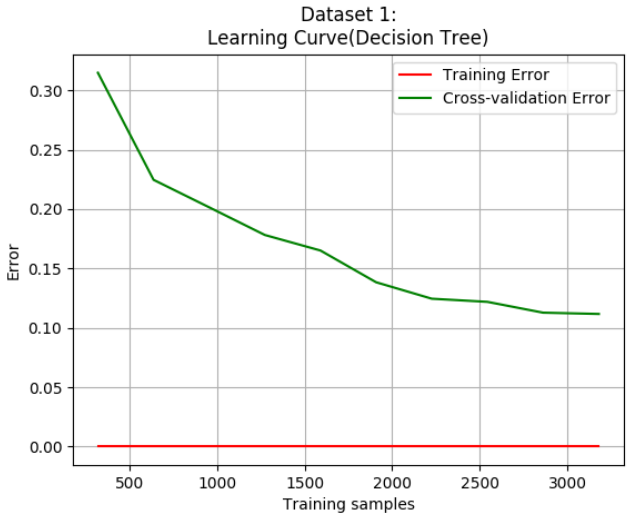

Fig 3.1.3. Learning curve of decision tree in dataset 1

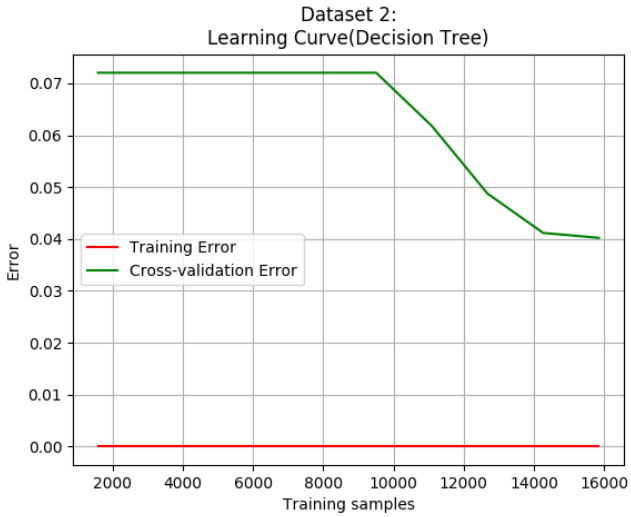

Fig 3.1.4. Learning curve of decision tree in dataset 2

From the learning curves of both dataset 1 and 2, we could find that training error always remained to 0 as training instances increased. Also, the two curves of both graphs didn't meet together, which means we need more training instances to get a better model of decision tree. And that was why we couldn't find the overfitting point in both datasets.

3.2. K-Nearest Neighbors:

3.2.1. Bias and Variance:

Since for each set of training dataset but the same input data, the neighbors of this input would always be different so it causes the variance for the KNN method. Just as the graphs show, the smaller the k is, the more it would become overfitting and if k==1, it must be overfitting because of the high variance.

As its name means, KNN's key idea is to find the nearest neighbors, to it prefers the data that the near points are "similar" according to a good distance function. The same reason, smoothing functions are preferred by KNN. Last, because it needs to measure the distance between points so all features are better to be matter equally.

3.2.2. Model Analysis:

Here I used 1/k in x-axis instead of k because the smaller k is, the easier to become overfitting. As the result, the scale of x-axis was log(x).

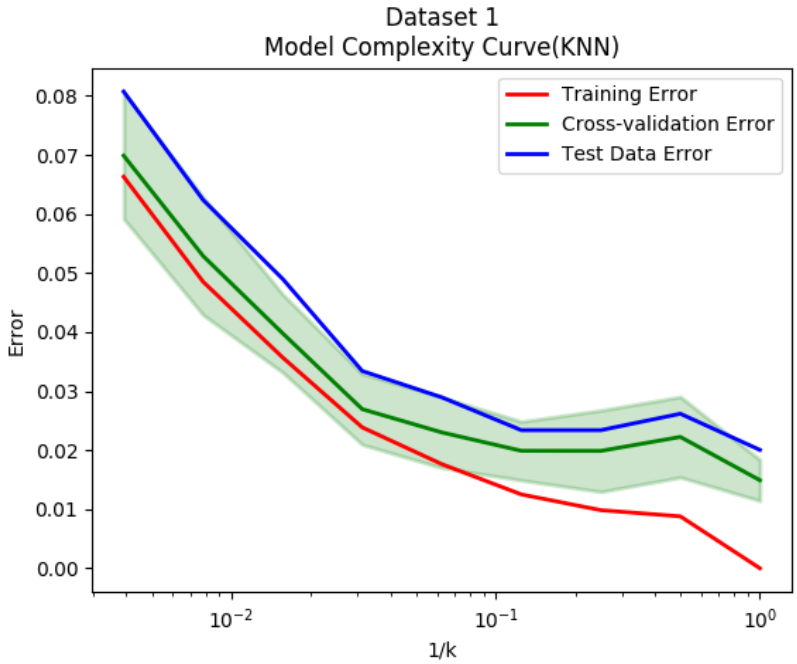

Fig 3.2.1. Model Complexity Curve of KNN in dataset 1

In dataset 1, we didn't find the obvious over-fitting point and we could see that even in k = 1, it had an excellent result. Except the reason of small amount of training instances, I think it was because the clustering of this kind of this dataset was great and each feature could contribute an equal distance valuation. That's why the test error was quite slow even if it was such a large amount of test data. So in my opinion, I would choose k=1 as best model regarding to the distribution area of the data.

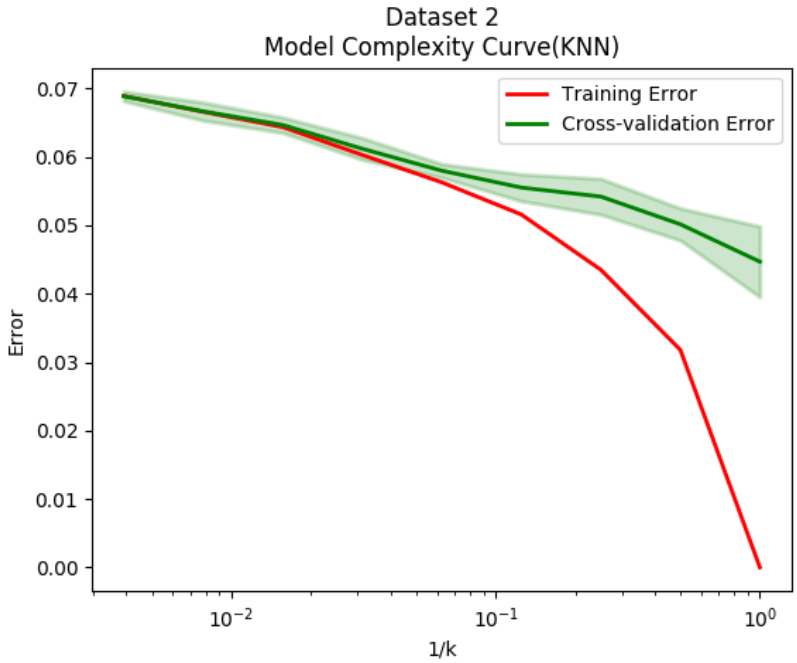

Fig 3.2.2. Model Complexity Curve of KNN in dataset 2

In dataset 2, compared to dataset 1, the divergence of training curve and CV curve was very obvious. And when k=1, the overfitting was the worse. Therefore, we could choose k = 10 as the best model for this dataset.
We can also see the learning curves of both datasets were merging together. Therefore, it proves that KNN model even when k = 1 really fitted to do the classification of data from dataset1 and k = 10 was the best model.

Although the learning curves of both datasets started to merge together, the model complexity curves showed different results in getting the best model and the overfitting point. In my opinion, I think KNN was the one of the best model that could apply to first dataset according to its instances distribution by the 64 features and it also showed a good score in running time and accuracy.

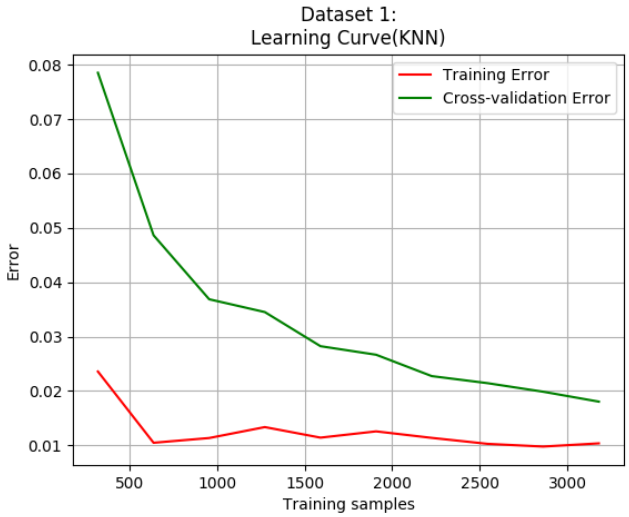

Fig 3.2.3. Learning curve of KNN in dataset 1

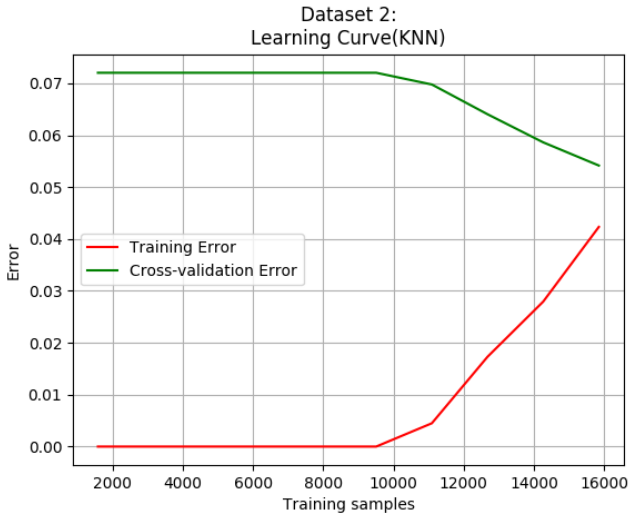

Fig 3.2.4. Learning curve of KNN in dataset 2

3.3. Support Vector Machines

3.3.1. Bias and Variance:

The data of different dataset will get different support vectors to build the machines, which causes the main variance by different training dataset. Also, if you change the kernel types or kernel size, it will produce a new support vector machines.

Since SVM would find the division of the data, so the margins between different labels are the bigger, the better and SVM dislikes the data with unclear margins. As the result, this method is very sensitive to the noise and outliers.

3.3.2. Model Analysis:

In SVM model complexity test, I used the gamma as the complexity parameter. Gamma value was the coefficient of the radial basis function kernel of SVM in scikit-learn.

In dataset 1, we could see the overfitting point was approximately in gamma=$10^{-2.6}$ and it was also the best value of the model. After this point, training curve kept a good result, but test error and CV error increased rapidly.

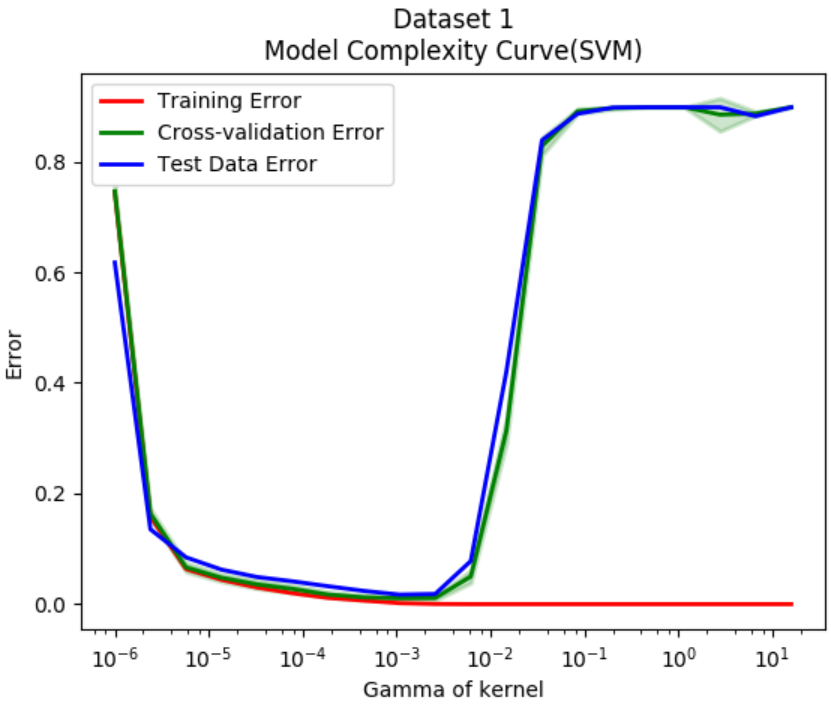

Fig 3.3.1. Model Complexity Curve of SVM in dataset 1

But in learning curve of Fig 3.3.2, we could see that the final result of blue curve with the default gamma had the trend to converge with the training curve. But when gamma=$10^{-2.6}$ (about 0.0025), we could get a CV curve much closer to the training curve and there was a convergence between training curve and CV curve. As the result, we could get a overfitting point when gamma=$10^{-2.6}$.

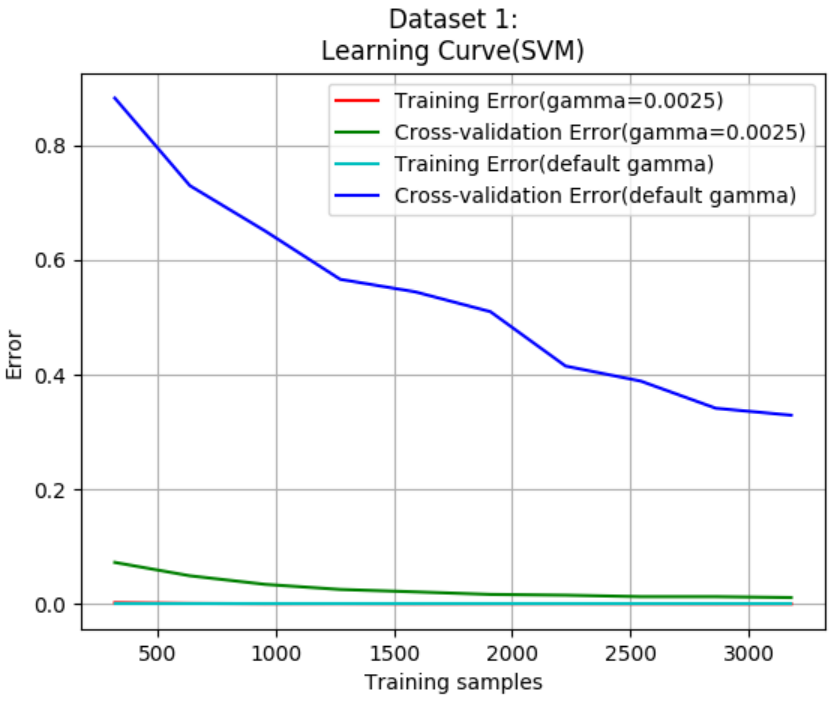

Fig 3.3.2. Learning curve of SVM in dataset 1

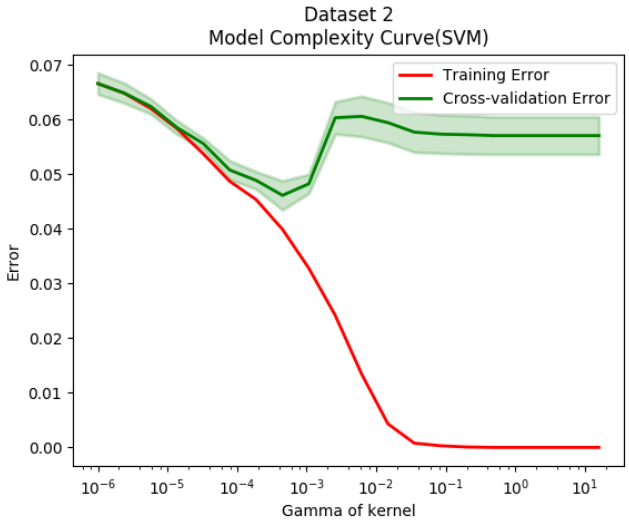

Fig 3.3.3. Complexity Curve of SVM in dataset 2

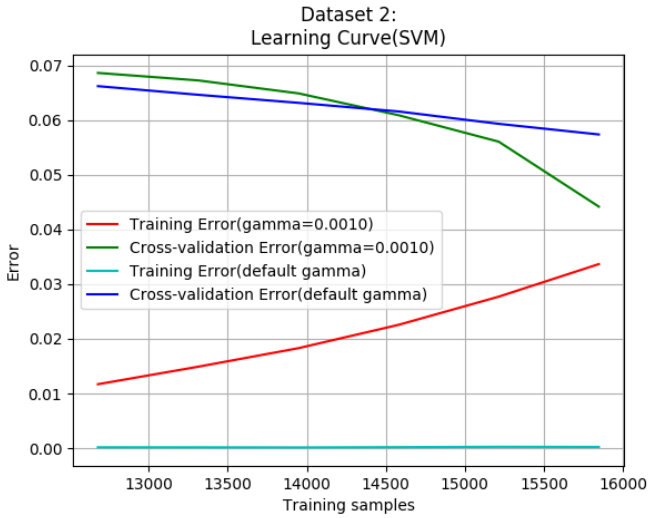

Fig 3.3.4. Learning curve of SVM in dataset 2

In dataset 2, after gamma=10^{-3}, CV error started to increased but decreased after a while. It might be because that the larger gamma value could cover the outliers and noise of the dataset. However, the best value of gamma was 10^{-3} for this dataset using SVM algorithm. In learning curve, as mentioned in dataset1, when gamma was the best value, the training curve and CV curve would converge much more obviously.

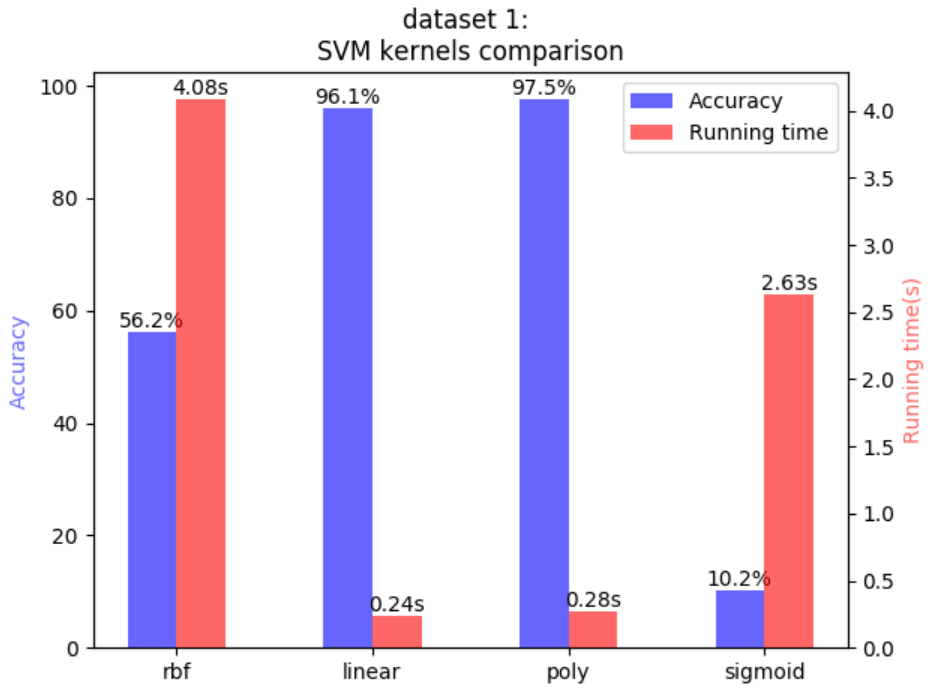

Fig 3.3.5. SVM kernels comparison in dataset 1

Moreover, I compared different types of kernel in SVM for dataset1. We could see that with different kernels, the results would have a great difference. In this case, linear and poly kernel got a great score in accuracy and running time. In conclusion, when using SVM model, it is very important to find the suitable kernel and its best coefficient.

3.4. Boost:

3.4.1. Bias and Variance:

Boosting makes a lot of simple classifiers (weak learners) to combine into a complex model (strong learner). With different training as input, the combination result of the weak learners of each iteration would be different and it will cause a huge variance especially when the weak learners don't fit to the data.

The bias of Boost is based on its formulation of the weak learners and Boosting can strengthen this kind of preference or restriction bias. Moreover, boosting is very sensitive to outliers in a dataset, which easily makes error happen during the iteration of combining the weak learners.

3.4.2. Model Analysis:

In this section, the boosting I chose was AdaBoost talked by Charles in the class and used the decision tree as the base estimators.

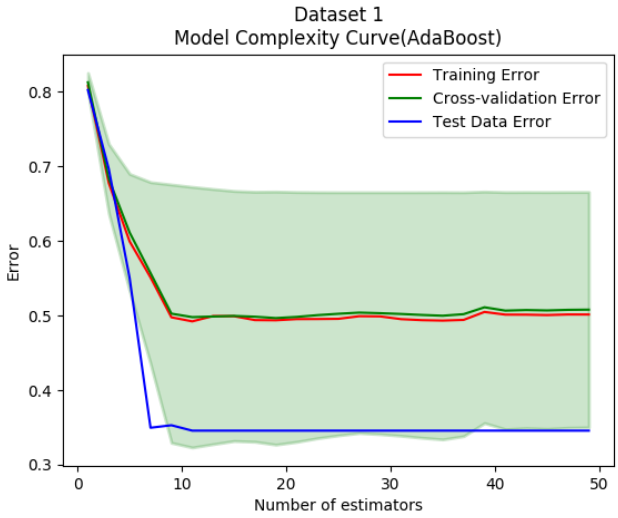

Fig 3.4.1. Complexity Curve of AdaBoost in dataset 1

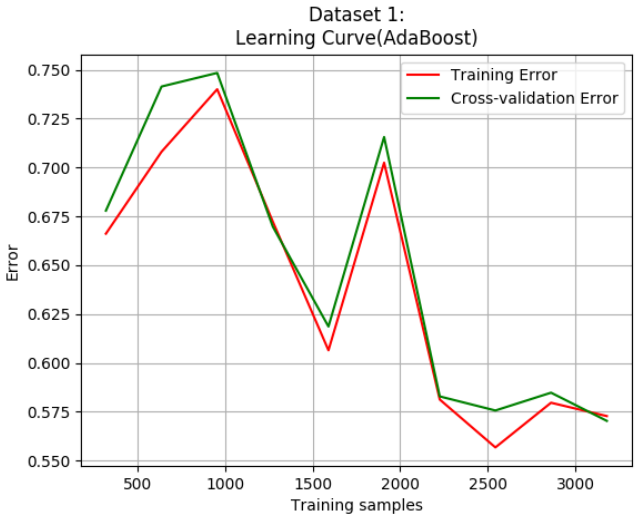

Fig 3.4.2. Learning curve of AdaBoost in dataset 1

It was very interesting to see that a terrible variance occurred in the model complexity curves in dataset1. Moreover, the accuracy was super low comparing to the other models. Sometimes the test instances scored even better than the training instances! But when we look up the learning curve, we could see that both of the curves were decreasing together and very unstable. As the result, we could know that we needed more and more data to make the weak learners to combine a strong enough learner.

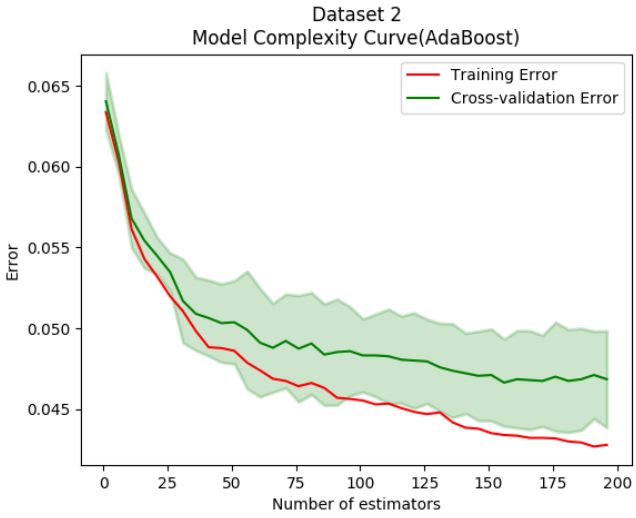

Fig 3.4.3. Complexity Curve of AdaBoost in dataset 2

Fig 3.4.4. Learning curve of AdaBoost in dataset 2

On the contrary, in dataset 2, we could see the real curve with enough training instances that both of the curves decreased gradually. According to the learning curve, the number of estimator of best AdaBoost model should be between 25 and 30.

Moreover, we would not see overfitting in AdaBoost model because of the combination of weak learners could avoid such a situation but we would have an over-high confidence instead.

However, since this model would strengthen the variance in some degree so the variance of both dataset was very high.

3.5. Neural Network:

3.5.1. Bias and Variance:

The objective of Neural Network is to approximate or describe a function according to the training data. As the result, the variance occurs because different data is training a different function.

According to the building process, Neural Network needs small random values of initial weights. Moreover, it prefers simpler explanations to simulate continuous and smoothing function.

Thanks to the help of hidden layers and their units, Neural Network can simulate complex models using lots of units so it seems that there is not much restriction but it is also dangerous to cause overfitting.

3.5.2. Model Analysis:

In Neural Network model, I used hidden layer size as control parameter and all the Neural Network models here only contained 1 hidden layer.

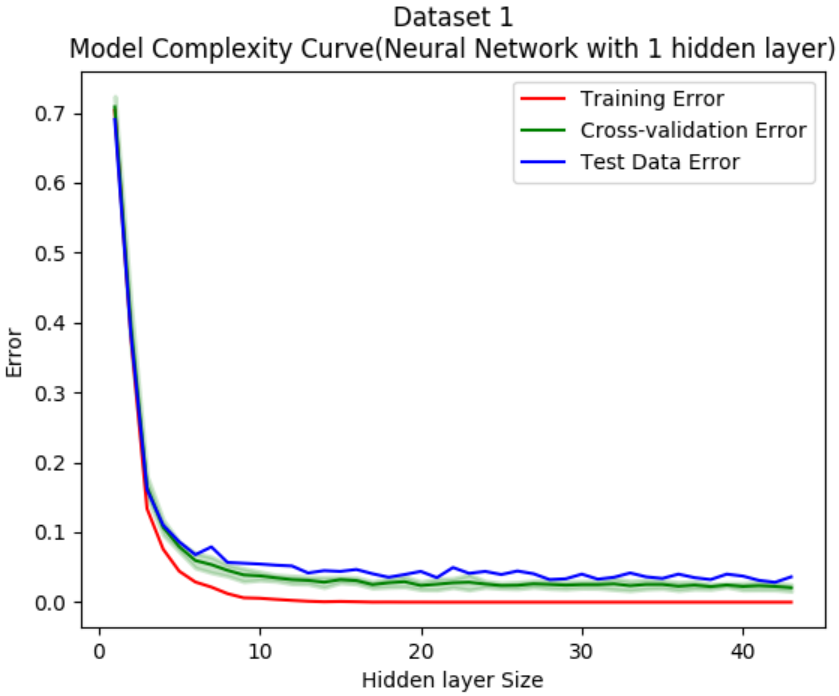

Fig 3.5.1. Complexity Curve of Neural Network in dataset 1

In the model complexity curves of both dataset, we didn't see the overfitting point.

For dataset 1, when the size was larger than about 10, both of training error curve and CV error curve became stable so the hidden layer size of 10 would be the best for this neural network model.

It was the same that when hidden layer size was for the dataset 2 to get the best neural network.

The reason why we couldn't find the was that the algorithm implemented by scikit-learn had set a restriction for iteration that when the training error started to converge, the algorithm would stop to avoid overfitting. As the result, overfitting point would not appear in this 2 graphs.

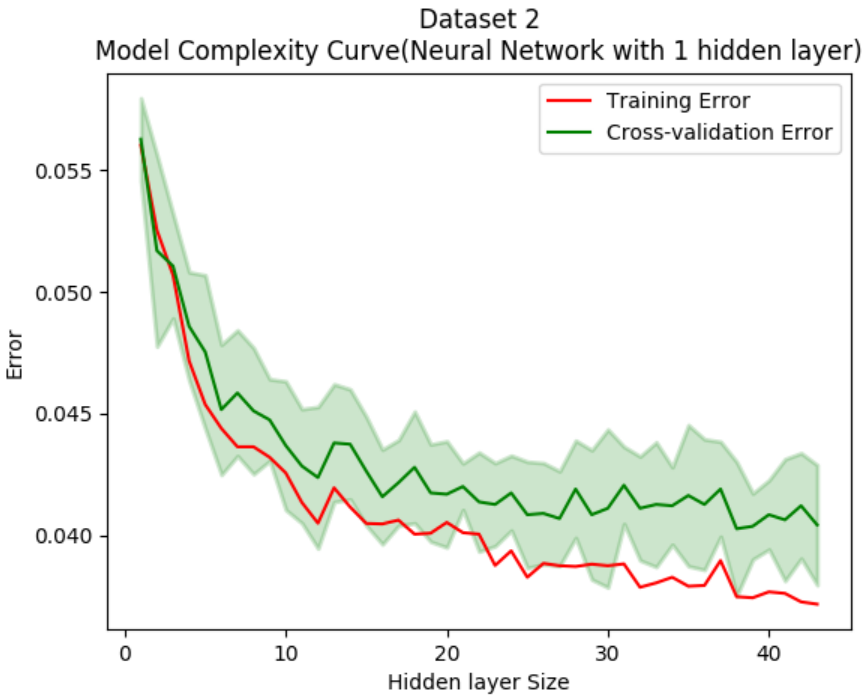

Fig 3.5.2. Complexity Curve of Neural Network in dataset 2

Therefore, in the graphs of learning curve, we could see the two curves had started to merge together in both of the 2 dataset but we still could not find an overfitting points in the graphs of model complexity curves.

However, because of the preference of Neural Network, the Neural Network model tends to be the most stable model comparing to the other 4 models. In addition, the Neural Network model has a great amount of applications in Computer Vision because it could often draw the characteristic of a series of images by considering all the pixels equally.

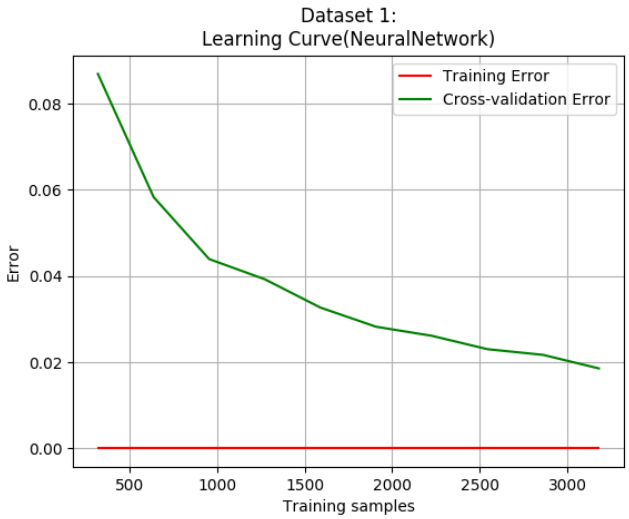

Fig 3.5.3. Learning Curve of Neural Network in dataset 1

Fig 3.5.4. Learning Curve of Neural Network in dataset 2

4. Receiver Operating Characteristic (ROC)

Since only dataset 2 was a binary classification problem, I only made the ROC curves in all the five models of this dataset:

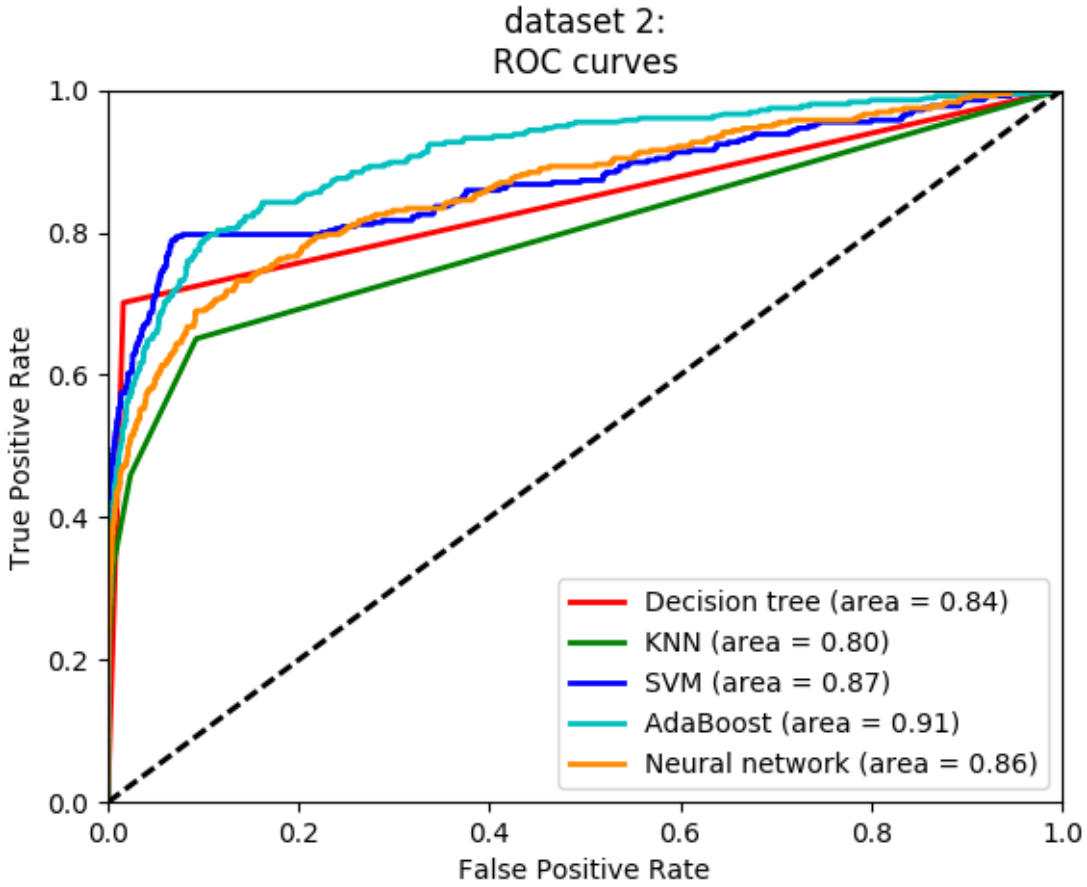

Fig 4.1. ROC curves of 5 models in dataset 2

In the figure, we could see that Adaboost model got the best score with the most area. That was because Adaboost could strengthen the confidence with enough data to get a correct classification. Then Neural Network, SVM and Decision tree had a good performance to get the correct classification. But the ORC curve of Decision tree was sharper at the start, which means that it got a good performance when classifying the true samples. At last, KNN got an area of 0.8 and the whole curve was below the other 4 models. So comparing to other models, KNN had more test samples with the regression results close to 0.5.

5. Summary and Arguments

5.1. Summary

Decision Tree: Easy to understand. Got a not bad accuracy. Both the running time and prediction time was very fast. The simpler and shorter the tree, the better the model. But it was easy to overfitting when there were many training instances.

KNN: A lazy learning model with nearly no training time. The prediction time highly depends on the size of training instances and their features number. And it had a great accuracy when dealing with multi-labels problem. However, it needs a great space to store all the training instances and it's very to become unbalanced.

SVM: All we need is to find the margins. So margins are very important. Prefer the data with large and clear margins. Unfriendly to the noise and outlier instances. Moreover, it would become very very slow when training a great amount of instances.

Boosting: It uses many iterations to combine the weak learners into a strong learner. Can decrease bias but may strengthen the variance. Has a good performance when dealing with the binary classification problem. Can avoid overfitting but may have over confidence of the data.

Neural Network: The most widely applied models in different kinds of area. Can approximate closely complex non-linear function. Easy to implement and learn the data. Has a mechanism by itself to avoid overfitting. Noise would hardly have an effect. However, it needs a great amount of training instances to get a strong neural network so it could still be easy to have overfitting.

5.2. Arguments and problems

The first dataset came from a famous dataset in Computer Vision, MNIST, where the original instances have 28*28=784 features! I have tried to run it, but however, it took me so much time to finish one training test. At last, I used this processed dataset instead. After data preprocessing, there are only 64 features. However, it could still get a not bad result in classification.

Dataset 2 are also processed data from imagines but the different point is that it extracts the real features, such as height, width of the images and there are only 11 features for each instance. Surprisingly, it had a great performance in the accuracy in all the 5 models.

Therefore, I am curious about the machine learning methods to extract the features from a series images which can decrease the input dimensions and improve the accuracy. And in Computer Vision, we usually use filters or mathematical methods to strengthen the features but it doesn't help to decrease the dimension. So I want to know whether there are good ideas to extract the features.