Assignment 1
CS-7641
Quynh Pham

Datasets

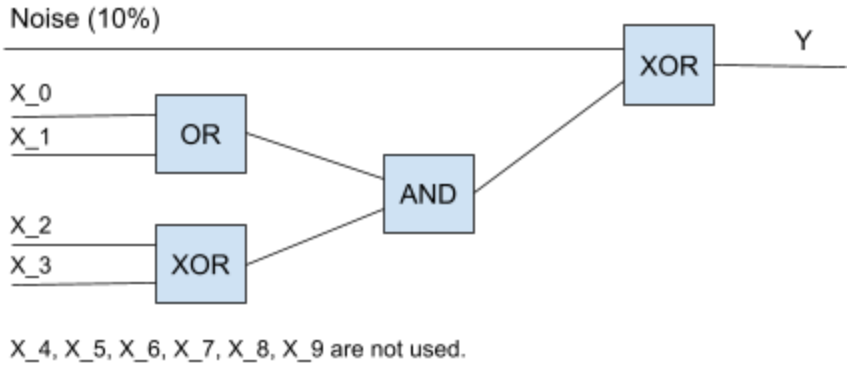
Noisy high-dimensional binary function.

The first dataset is a synthetic binary classifier with many features/dimensions. There are 10 binary features $x_0, x_1, x_2, \ldots, x_9$ and they are all sampled uniformly. We define output to be

$$((x_0 \textbf{ or } x_1) \textbf{ and } (x_2 \textbf{ xor } x_3)) \textbf{ xor } e_{noise}$$

Where $noise$ is a 10% binary error rate. Therefore, The best error one can expect is the noise 0.1. We compare the error rate on the train and test for different train sample sizes. We also look at different variants of classifiers to see when they have better performance and we explain why that happens.

We are excited about this function because it can show the **curse of dimensionality** and since it has xor many algorithms might fail in their naive format.



Red wine quality dataset

https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009
This dataset can be used for classification. It has a quality score that is an integer smaller than 10. So it can be used as either one of regression or multi-level classification. We use it as a classification problem and the error is 0 or 1 if the quality is classified accurately or not. This dataset is widely used and it is multi-level, so we are excited to see performance of our models on this dataset. One thing that we realized is that reducing classification error for this problem is not easy since it is multi-level.

General Results

The test error in all this assignment is colored orange and the train error is colored blue. With very high probability the test error is always higher than train. By more complicated models, the gap between test and train increases and as the number of samples increases, this gap shrinks. Neural networks and

boosting take more time to train than the rest. KNN for our set of problems was not very powerful since the high-dimensionality has a negative effect on finding nearest neighbours in KNN.

Neural Networks

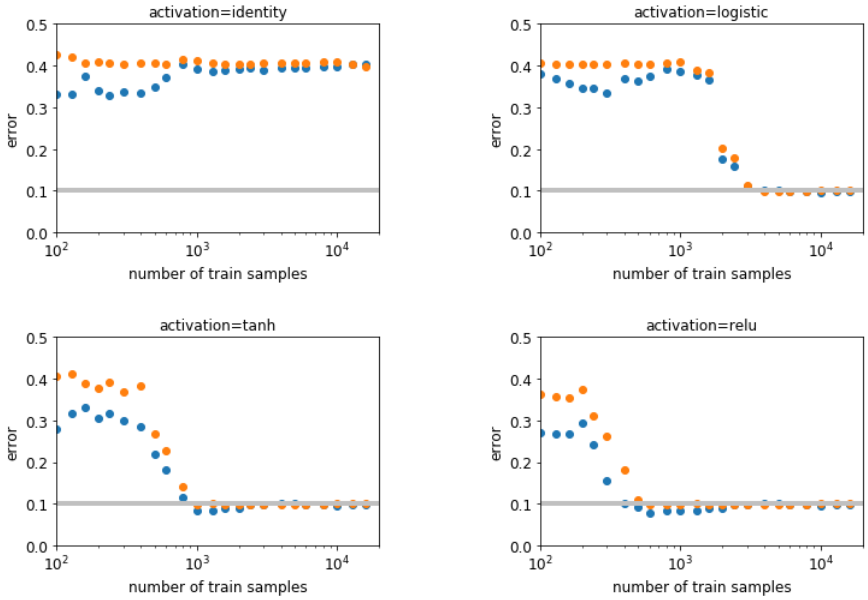
Neural network had longer training time. Scaling up as the maximum number of iterations increased. Performance-wise neural network error achieved the optimal error fast without need of many hidden layer sizes or samples. We looked at different activation functions and hidden-layer-sizes and did some investigation on number of iterations needed. Here are some findings regarding neural networks.

Dataset 1

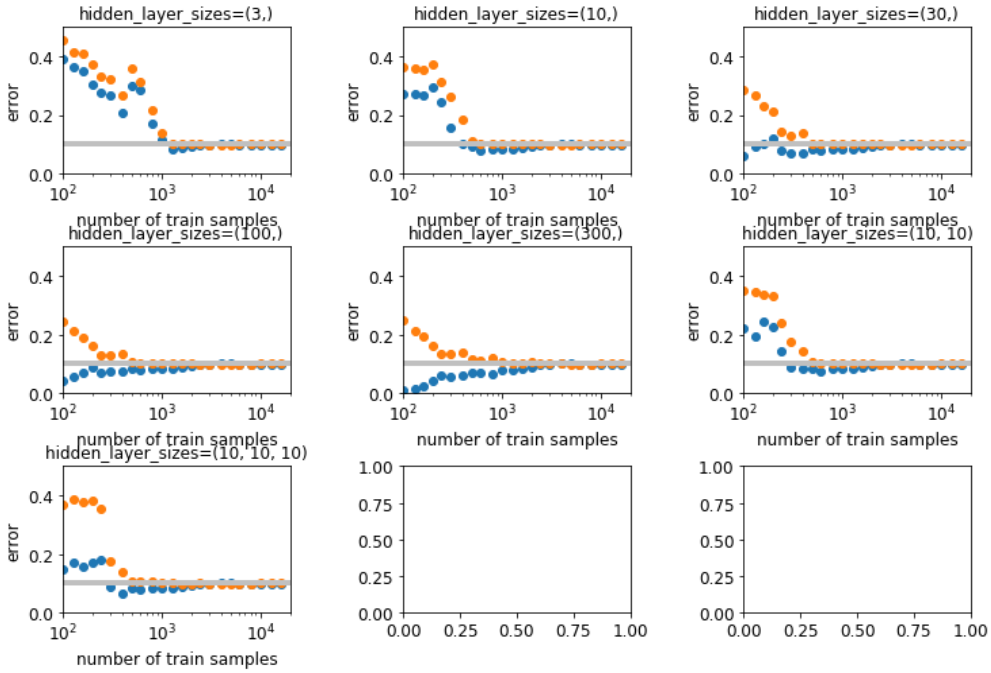
For **hidden layer size >=4** neural networks showed convergence to optimal solution.

- Training time was the highest for the neural network and increased by increasing size or max iterations.
- **Identity activation** cannot classify correctly. ReLU showed the best performance and that might be because the model is logical synthetic function.
- As **hidden_layer_size** increases the test and train gap increases. It also takes more samples to train the **classifier** and fight the overfitting.
- Increasing max_iter decreased the train_error much more compared to test_error. For the correct number of layers in our model it did not cause overfitting.

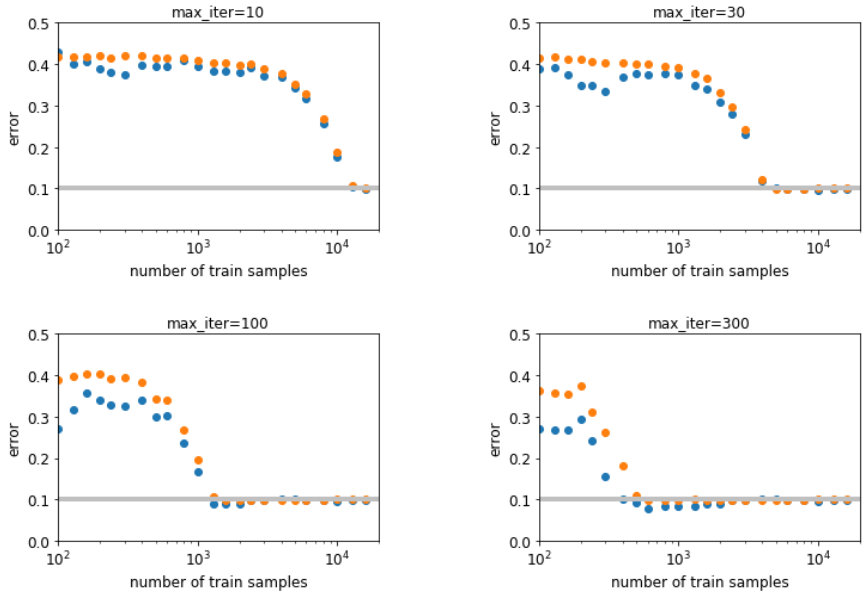
Variant activation
max_iter=300
hidden_layer_size =(10,)



activation=relu
max_iter=300
Variant hidden_layer_size



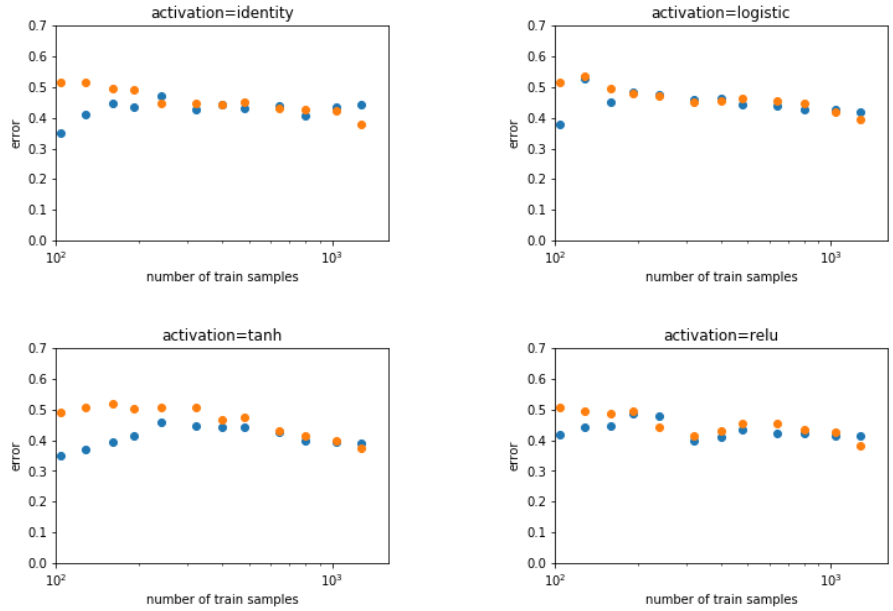
activation=relu
Variant max_iter
hidden_layer_size=10



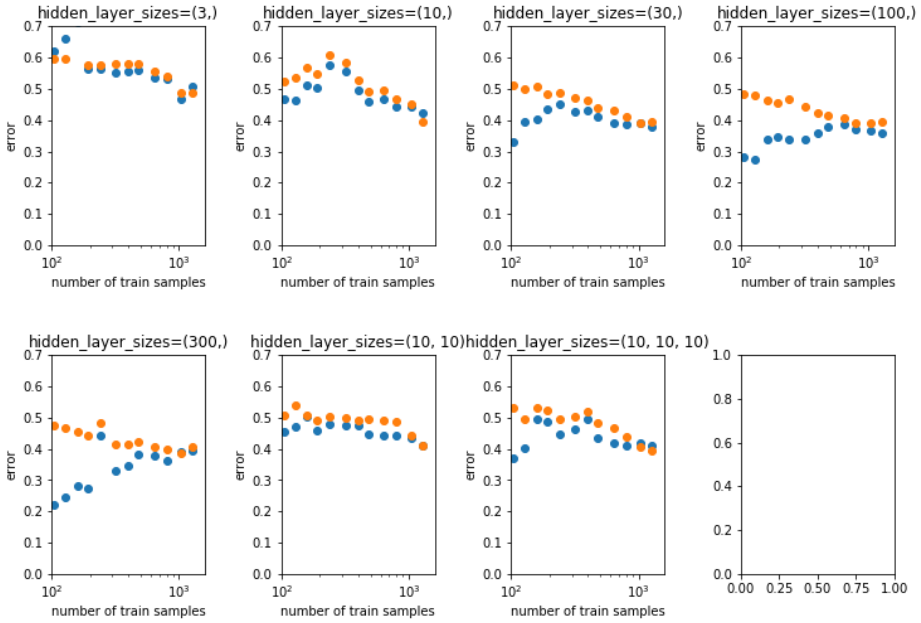
Dataset 2

- All **Activation** were similar but tanh was the most consistent one.
- Single **hidden layer size** = 10 has shown the best performance.
-

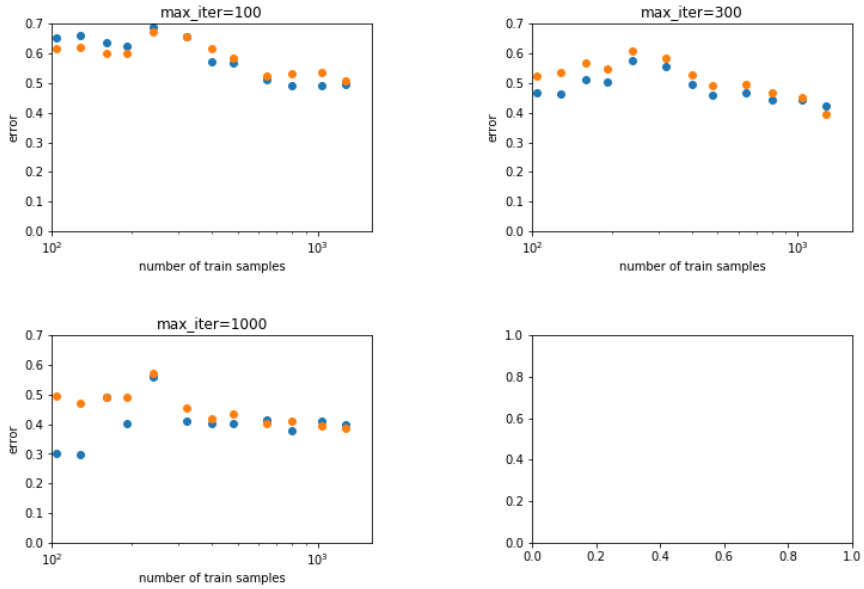
Comparing the activation effect by fixing the rest of the parameters.



Comparing hidden layer size effect by using activation function tanh.



It is clear that 100 iterations is not enough for the model to converge. Tuning the learning rate or increasing the max_iterations would help. 300 iterations with the default parameters shows some level of convergence. Having more data would help to increase the accuracy.

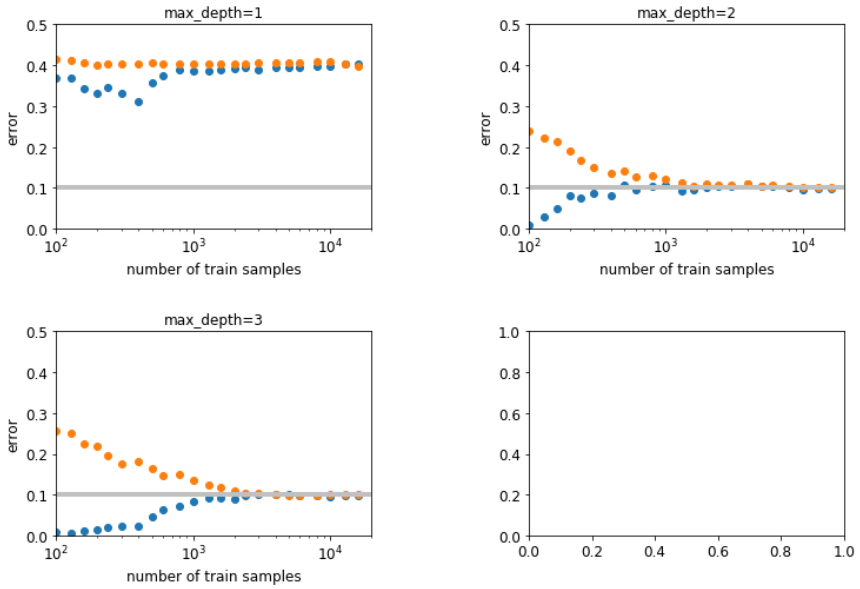


Boosting

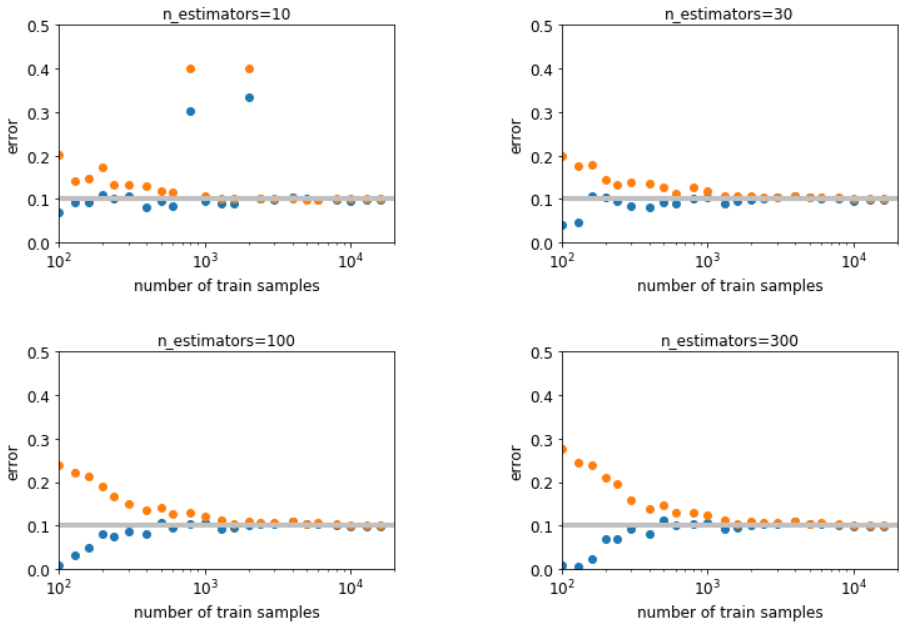
For this section we use the AdaBoost decision tree.

Dataset 1

For max_depth=1 the learning is unsuccessful since no single classifier can be effective for the dataset we have. But using max_depth=2 allows learning since combination of depth=2 decision_trees can make good weak classifiers.

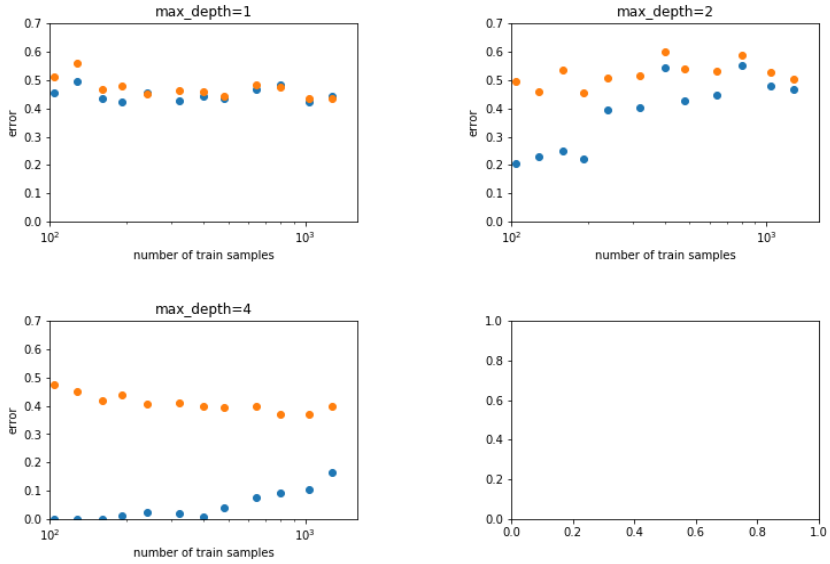


We also also change the number of estimators to see its effect for our model. As number of estimators increases overfitting increases but looks like 30 is a good number of estimators without less amount of overfitting.



Dataset 2

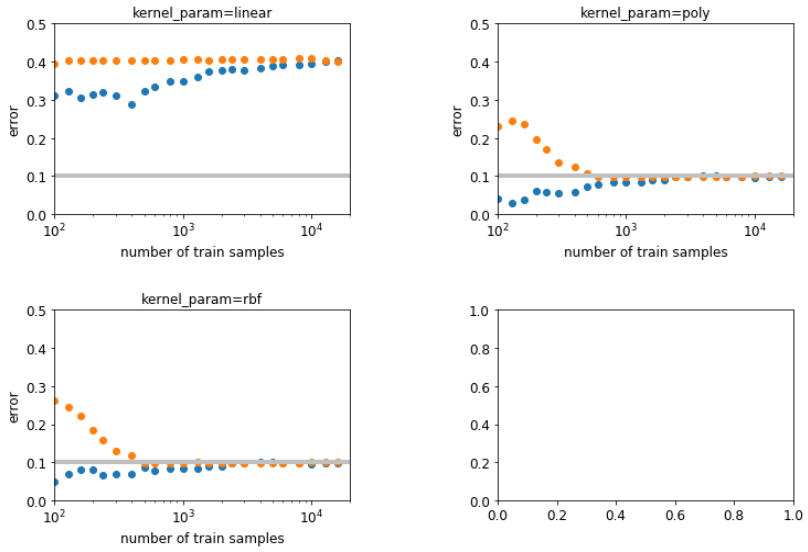
Increasing the depth and nun of estimators caused overfitting and reduced the train error but here is the selected values that optimized the test error for large number of training samples. Max_depth=4 is optimized.



Support Vector Machines

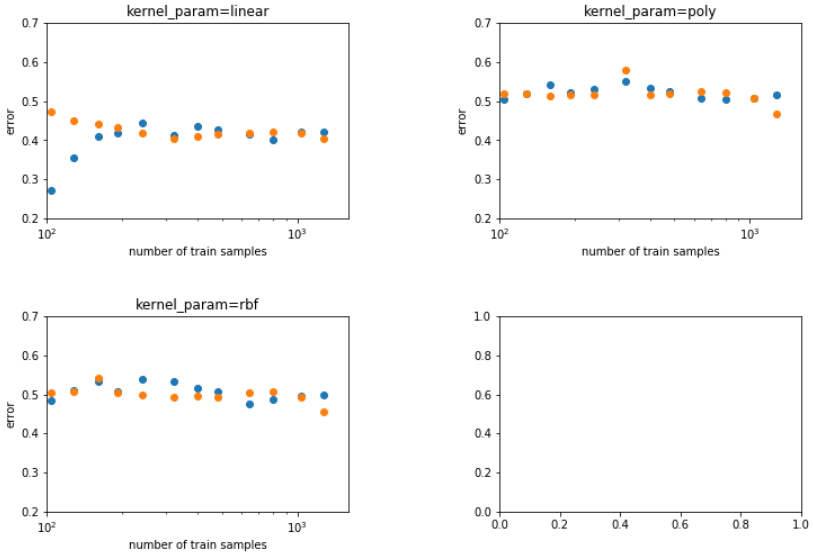
Dataset 1

Linear svm is not able to find a good classifier and the main reason is xor function. But when we move to polynomial with degree 3 then we can achieve optimal error rate. Rbf with the default gamma in sklearn's SVM.



Dataset 2

Unlike the previous dataset, linear svm is the best performing svm classifier for this dataset.

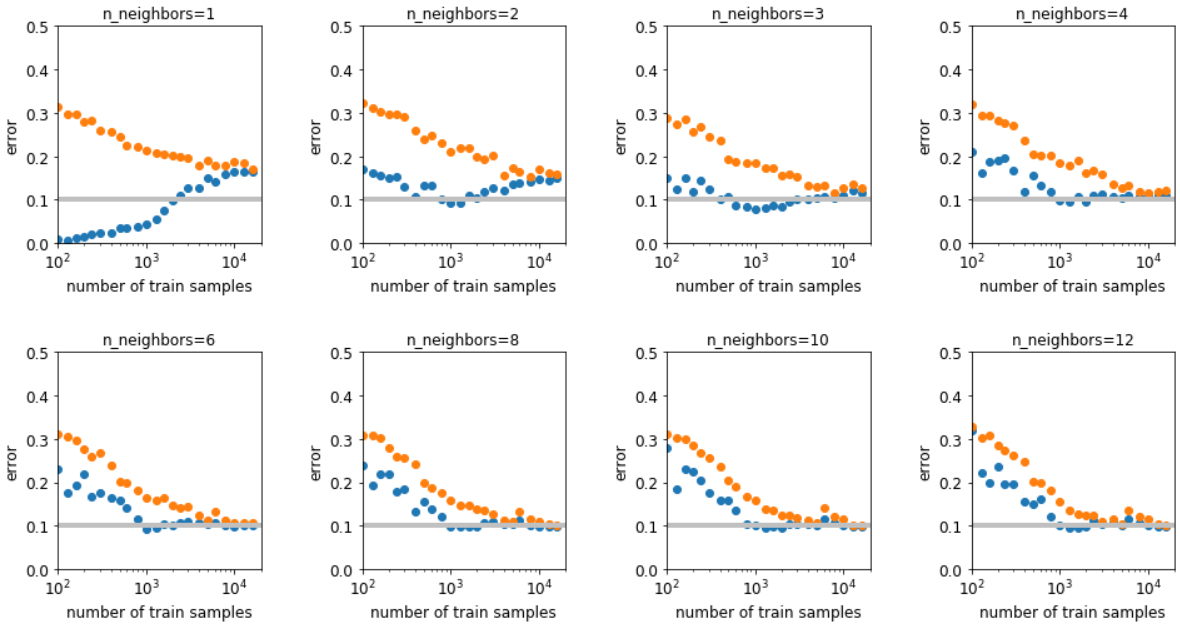


K-nearest neighbors

We can see the effect of the curse of dimensionality for both of datasets.

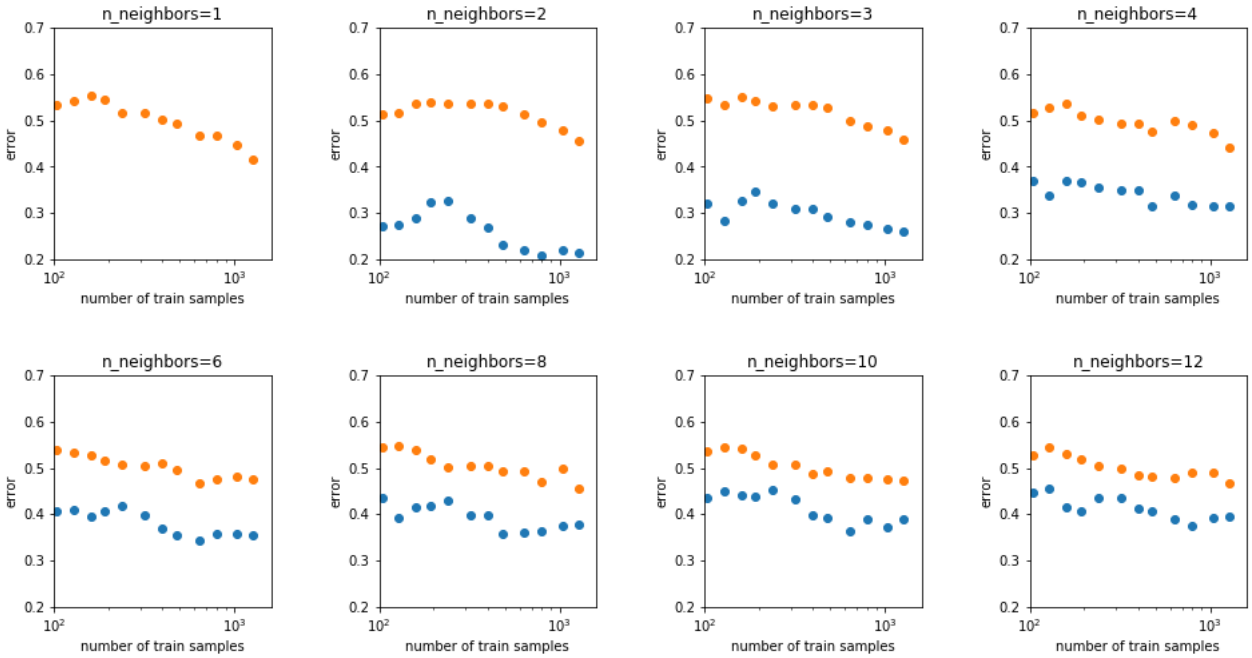
Dataset 1

this classifier. Even though only 4 features are important, it needs many samples to have a good estimate and also only larger k allows good voting power to reduce error. It looks like for datasets where inputs are distributed over many dimensions, the KNN needs some tweaking for parameter k to achieve the optimal error.



Dataset 2

Given that different values have different variance, it might have been good if we would have scaled the dimensions to reduce asymmetry between different features. One thing to note is **Manhattan** distance handles the asymmetry better than **Euclidean** distance and for that purpose we used it in this problem. Also k=1 was the best performing one and as you know train error for that is fully over-fitted to 0.



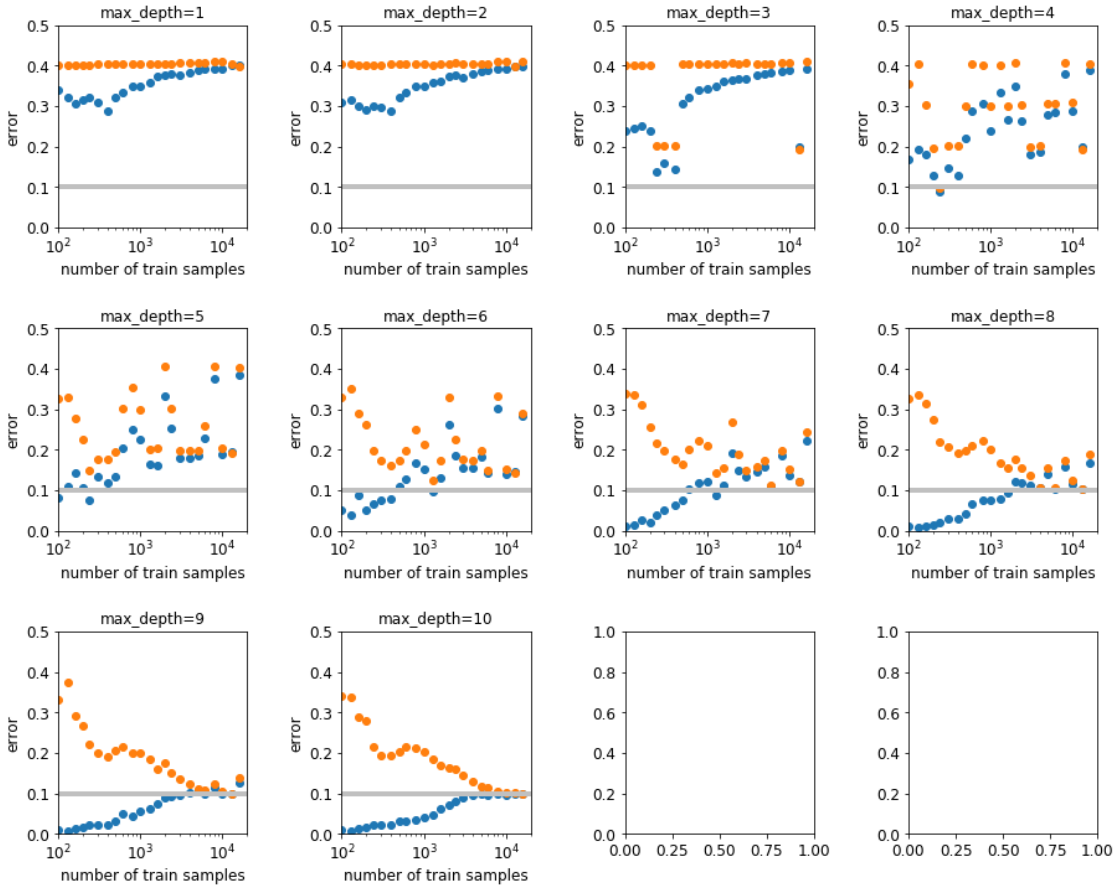
Decision Tree

There are many ways of controlling overfitting and pruning in decision trees. For python sklearn library, one can use max_depth, max_leaf_nodes, min_samples_leaf, etc or combination of these. We used

some of these in our models specifically for dataset 2. We used min_samples_leaf to prune the result as well as using different max_depth.

Dataset 1

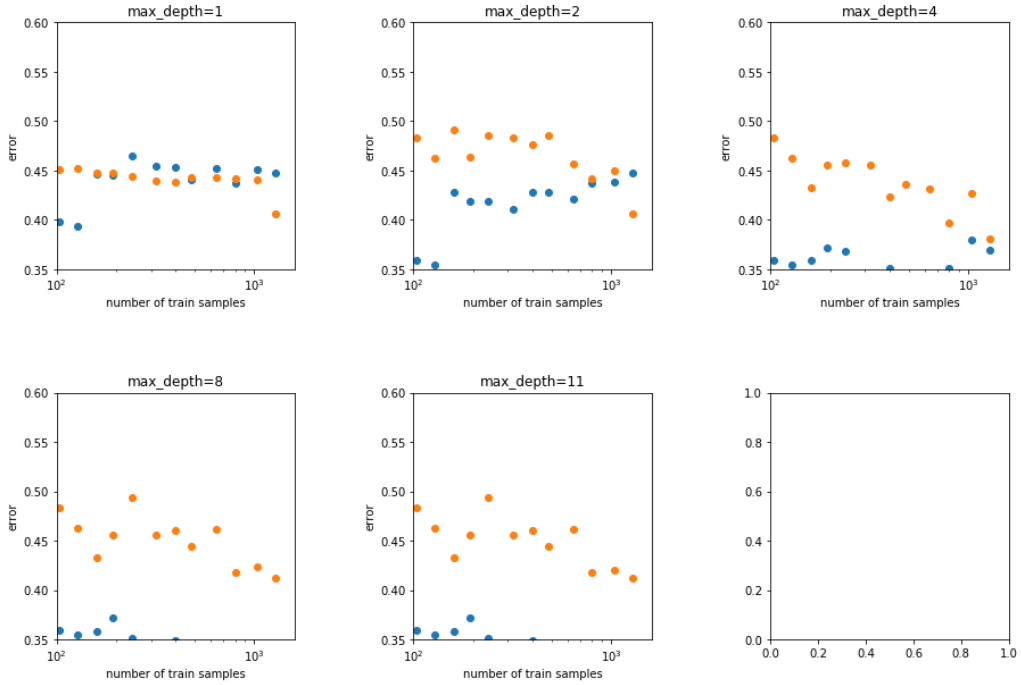
Given the characteristic of this problem and having xor function with fully balanced probability we observed that the decision tree fails with depth less than 10. Decision trees have a tendency to look at per feature gain in each node and sometimes these per feature gains are negligible. On the other hand, if data was unbalanced or each individual feature had more contribution to the result, the decision tree might have been more successful to pick them up in the early depth.



As we can, test and train error reduces as the number of samples increases and train and test has a wide gap specially for larger depth. This is overfitting for small samples and only when the number of samples is very large test error reduces. Clearly XOR function with a balanced set of input caused the decision tree to malfunction and **pruning** in this case would not help.

Dataset 2

One observation is min_samples_leaf=20 had good increase in reducing test error. After optimizing pruning factor max_depth, here are the results. **max_depth=4** is the best performing one.


max_depth=1
max_depth=2
max_depth=4
max_depth=8
max_depth=11
error
number of train samples
0.60
0.55
0.50
0.45
0.40
0.35
10^2
10^3
1.0
0.8
0.6
0.4
0.2
0.0