CS7641 Machine Learning Assignment I
Supervised Learning Models for Classification

Wang Lu, GTid: 903355610
7th September ,2018

Part I. Introduction

To explore supervised learning techniques in a practical manner and gain more sights about how they perform in different situations, I herein choose two classification problems with distinct properties of data for my analysis. The first problem, finding high income individuals is inspired by Udacity's Finding Donors for CharityML project [1], the respect dataset originates from the UCI Machine Learning Repository. It was first introduced in 1996 by Ron Kohavi and Barry Becker, after being published in the article "Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid" [2]. I simplify the classification task as predicting individuals with income more than 50K USD from demographic statistics, which is imbalanced binary classification with originally 8 categorical features and 5 numerical features on 45K entries. The second dataset is collected and processed from commercial website TMDB[3] via its API, it has a mixture of data types: time series, categories, texts and numbers which allow flexible feature extracting. I am interesting to predict whether a movie is above-average profitable or not based on features such as cast, director, runtime, etc. Essentially, it's a balanced binary classification problem on a relatively small dataset (6K entries) with hybrid features.

I believe the choice of datasets allows an interesting comparison of the 5 models, in particular Multi-Layer Perceptron, Decision Tree, K Nearest Neighbors, Support Vector Machine and Gradient Boosting, on their response to the dataset balance, scale, and feature heterogenicity. Besides, the two classification problems are practically interesting respect to their potential application significance. For example, predicting individuals' income level would help precise targeting of social program and commercial advertising, and inspecting of features negatively contributing to the expected profit benefits better decision making of the $40 billion worth movie industry.

Part II. Preprocessing, Evaluation and Validation Methods

Data Pre-processing

From explorative data visualization on dataset1 (not shown in submission), features [`capital-gain`, `capital-loss`] are found to be heavily right skewed. Thus, logarithmic transformation was applied in order to prevent the input from being impacted by a few outliers [4]. Noticeably, to make sure each feature be treated equally by machine learning models regardless of original values, a max-min normalization without changing the respect distributions was taken on all numerical features across the datasets [5]. Also, one-hot encoding was carried out on all categorical features, enabling downstream their feeding into machine learning models [6]. (Feature engineering and selection on dataset2 is too complicated to be elaborated on in this short paragraph. Please look up respect Jupyter notebook sections for commenting). In the end, total 103 features for problem1 and 79 features for problem2 are finalized for modeling.

Multi-collinearity

There is indeed weak multi-collinearity among a bunch of features in both datasets. For example, the correlation in dataset1 indicated a few nationalities can be speculated from `occupation`, `relationship`, `race`, `marital-status` and `workclass`. However, considering the multi-collinearity impacts mainly on coefficients of features instead of changing model's predictive power itself, these few features were simply left untouched [7,8,9].

Metric Chosen

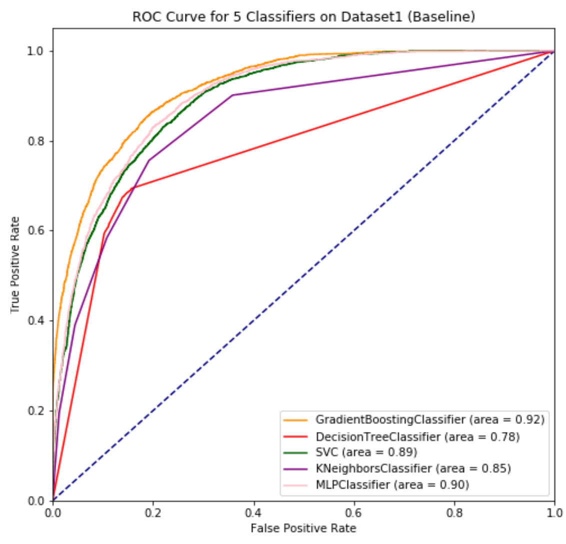
Metrics used in ROC curve are FPR (False Positive Rate), TPR (True Positive Rate) and AUC (Area Under Curve); Metric used in learning curves and model analysis is MSE. (Mean Squared Error) Notice that MSE is calculated from 1.- accuracy_score in this report, this can be justified by the fact that the two numbers are equivalent for the case of binary classification problem.

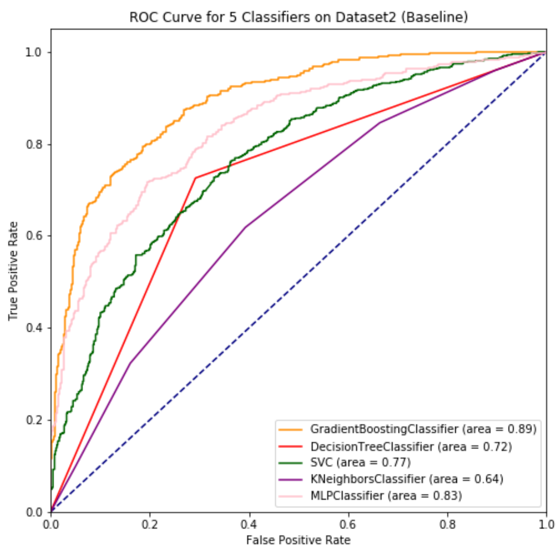
Cross Validation

Instead of K-fold cross validation, I choose the train/test split validation using `StratifiedShuffleSplit`, which make sure the training and testing set are evenly splitted and points in training set are representative enough for estimating the true distribution.

Part III. ROC Curves and Learning Curves

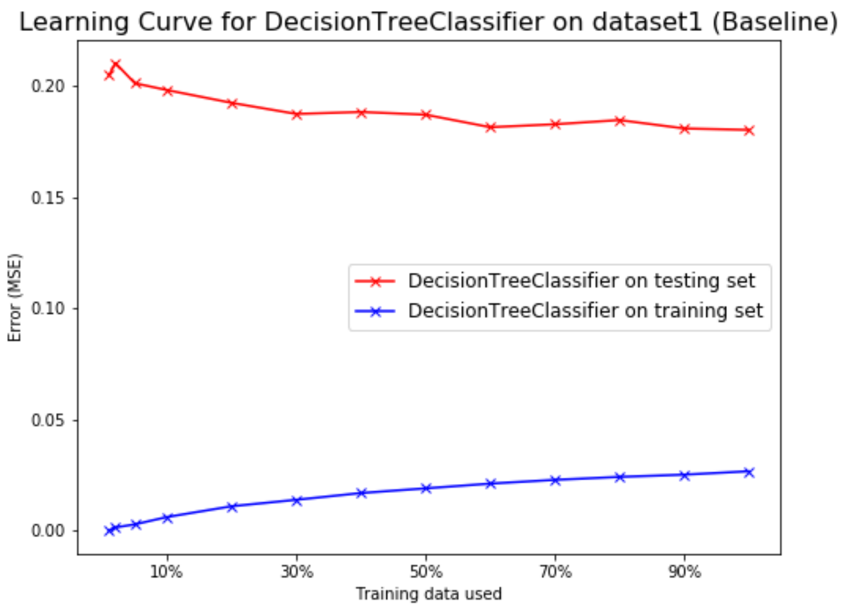
ROC Curves for 5 Baseline Models on 2 Datasets

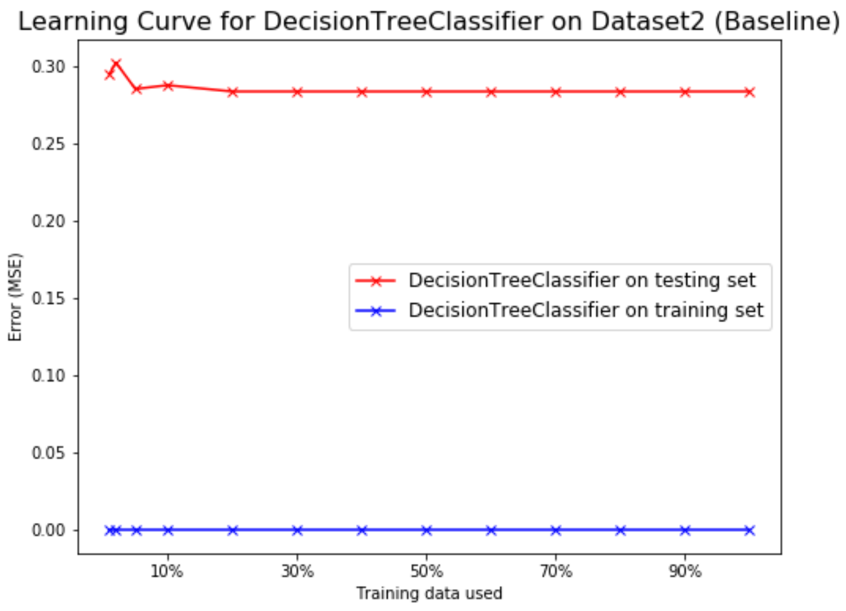




From the ROC curves above, 5 classifiers on problem1 show more consistency and accuracy than on problem2, implying problem1 is perhaps less challenging. However, highest AUC score being 0.92 tells that neither of the 2 problems are too trivial. It is not surprising that overall performance of all classifiers on dataset1 is better, as the first task is provided with more complete data, more training points, and less heterogenous features. And notice that the relative predictive power on two tasks match each other, with `GradientBoostingClassifier` > `MLPClassifier` > `SVC`, and `KNeighborsClassifier` and `DecisionTreeClassifier` struggle the most, perhaps from overfitting. We will see if it is indeed the case by tuning their complexity in later analysis.

Learning Curve for Decision Tree

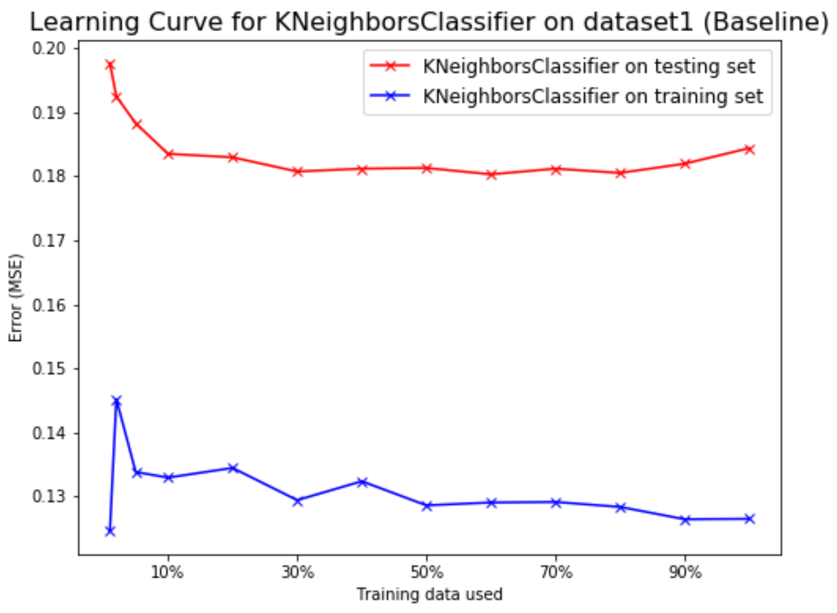


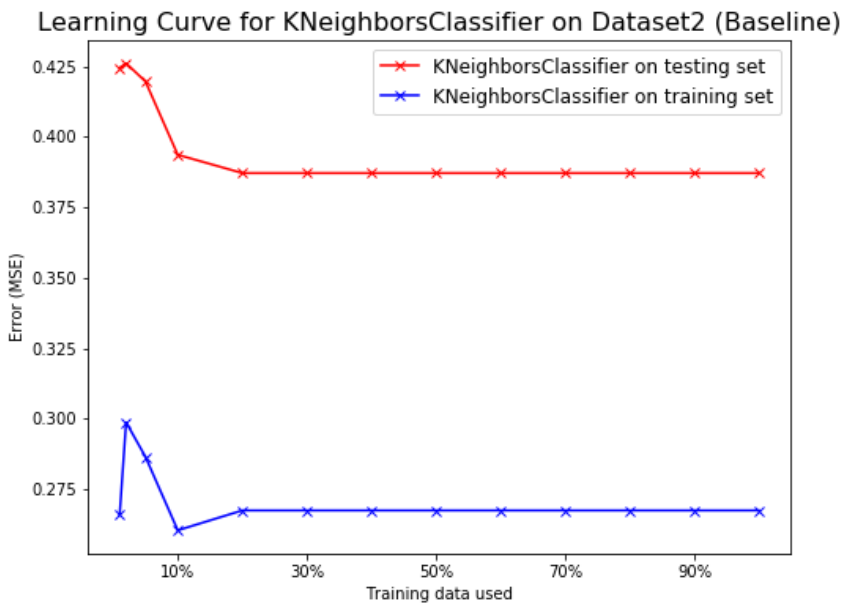


Decision Tree in sci-learn kit by fault use Gini impurity instead of information gain that taught in our class to measure the quality of split [10], and no pruning would be conducted with this setting. It almost for sure that the decision tree built would grow too complex and cause overfitting. (Notice that all method with stochastic processes in this report are seeded with `random_state` = 0, to make sure the reproduction of result)

From the learning curves above, it's obvious that there is a big gap between performance on training and testing set, indicating the model is suffering from high variance. Along with more data points added in training, the decision tree built on dataset1 show a trend of converging while the one on dataset2 didn't. It is perhaps because dataset2 has much 7 times less sample than dataset1, adding more sample is likely to render a converging trend on its learning curve. After checking the attributes of the baseline tree, I found indeed, even the tree on dataset1 has grown up to 53 depth with more than 6 thousand nodes. Decreasing of model complexity by pruning should improve decision tree performance in both cases.

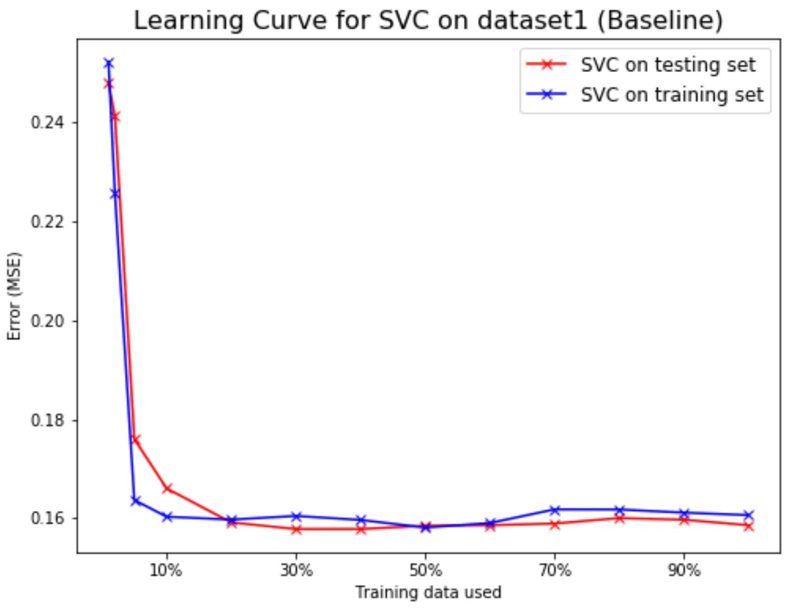
Learning Curve for K Nearest Neighbors

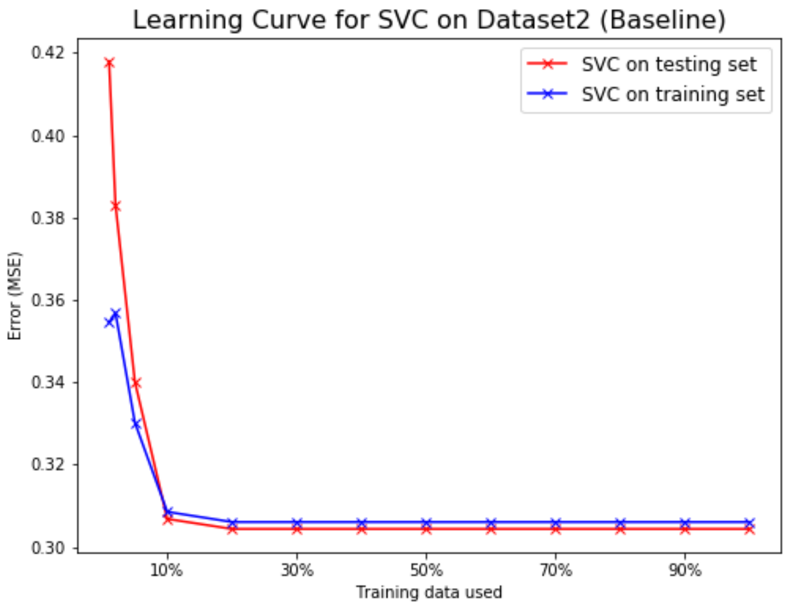




K Nearest Neighbors is a very simple and straightforward method to classify samples with the assumption that samples with similar features should belong to same class [11]. The key factor for KNN decision making is the choice of K, which is how many neighbors should be chosen to vote for the class of a new sample. If K is very small, such as 1, then individual's class is solely decided by the nearest neighbor, thus extremely sensitive to outliers and overfitting. Increasing K value actually reduce complexy of KNN model. Here from the learning curve above, we can see firstly there is a big gap between testing and training set performance and secondly the curves are initially very unsmooth, both implying the current model suffering from high variance and overfitting. Bigger K values should be tried on later models.

Learning Curve for SVM

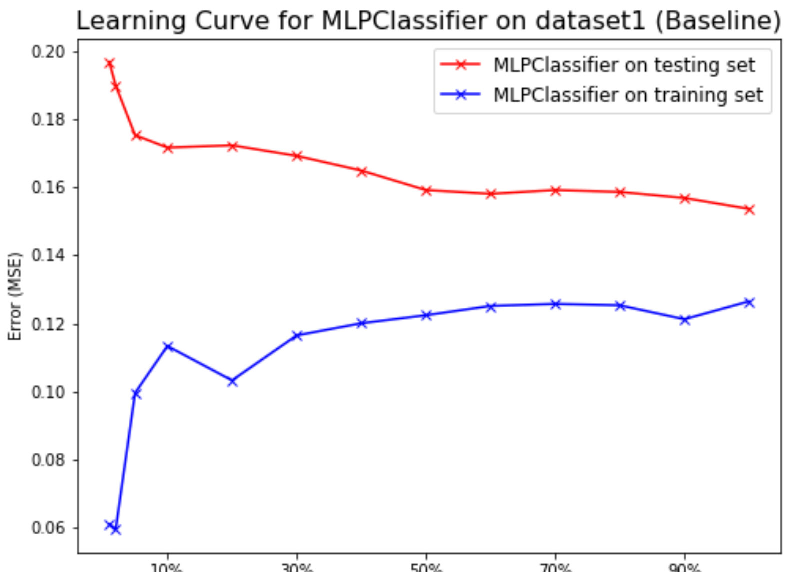


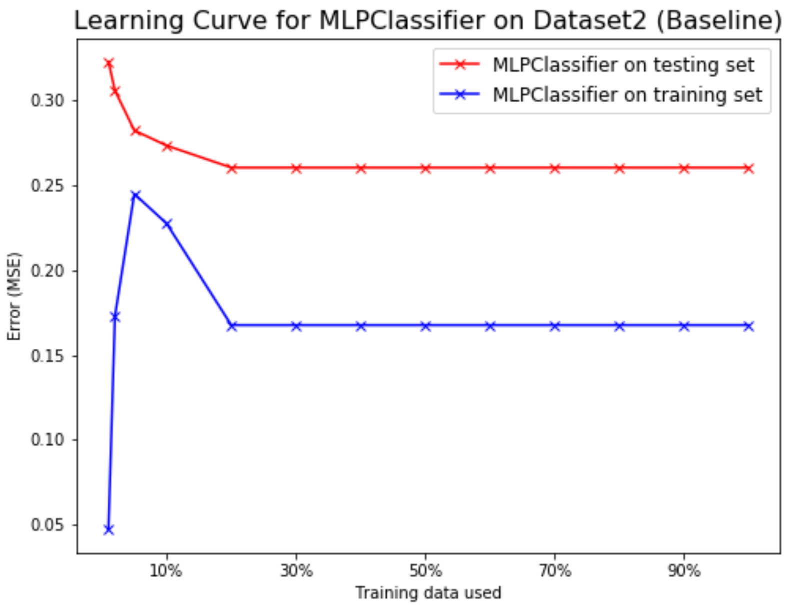


The default SVM model sci-learn provides is with non-linear `rbf` kernel, a very flexible model which would project sample to higher dimension space under linear-inseparable situation, but with the sacrifice of computation time [12]. Learning curves of this model show very different pattern than the previous two: Errors drop dramatically with only 10% of training sample and testing and training sets rival each other for the rest of the time at a level around 0.16 and 0.30 for dataset1 and dataset2, respectively. This pattern implies high bias of the model. Basically, adding more data or increasing training time won't help to improve the performance, we might change the assumptions made about the model instead, perhaps by changing its `kernel`.

Curiously, MSE on training set at 1% training data exposure is the biggest. A plausible explanation might be the uneven internal splitting of training data (unlike testing/training set splitting in previous section, here the internal training data splitting is done by simply slicing instead of stratified shuttle splitting.)

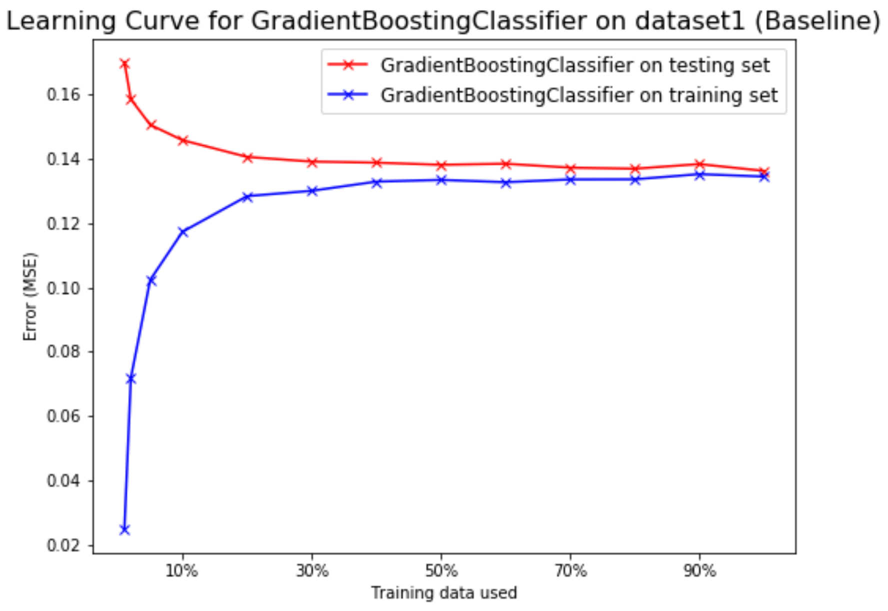
Learning Curve for Multilayer Perceptron

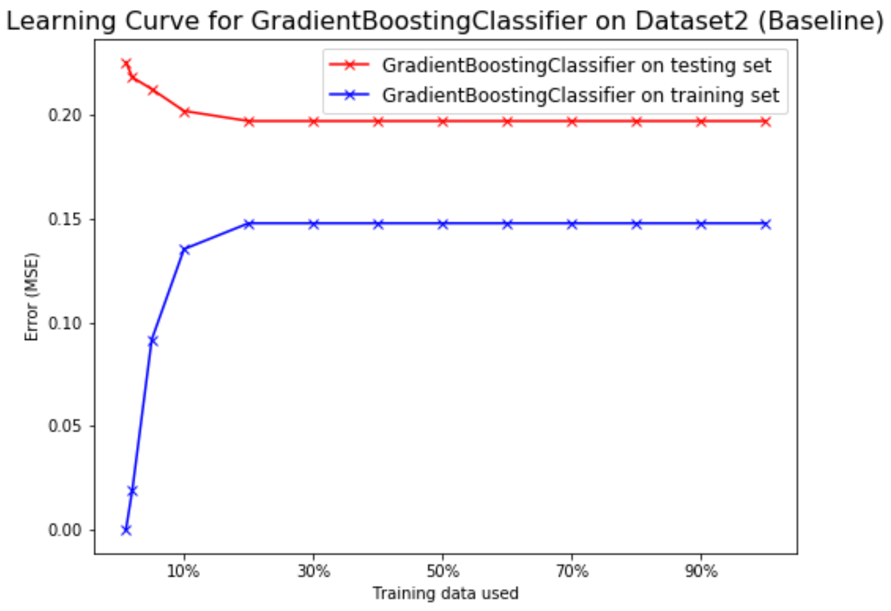




This is a multilayer perception with one hidden layer (100 units) by default [13]. Similar to the decision tree case, MLP on both of datasets suffer from overfitting while the one on dataset2 even worse, implied by the performance difference and the converging trend. Higher regularization power or small number of iterations can be tried to improve the model performance.

Learning Curve for Gradient Boosting Machine

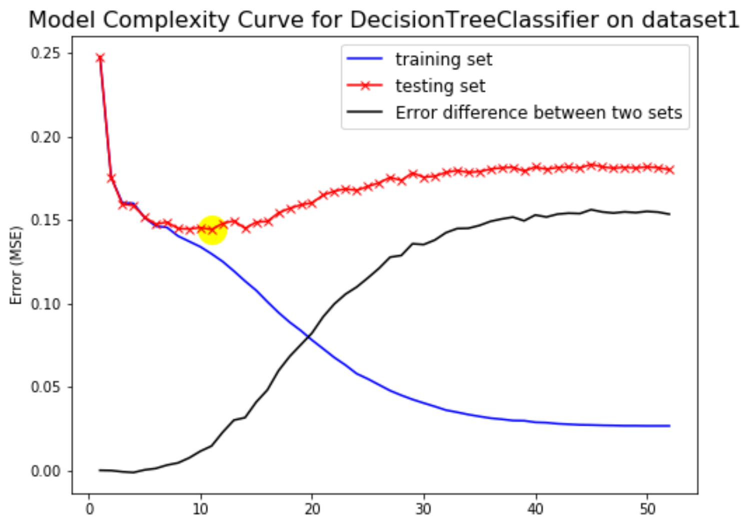


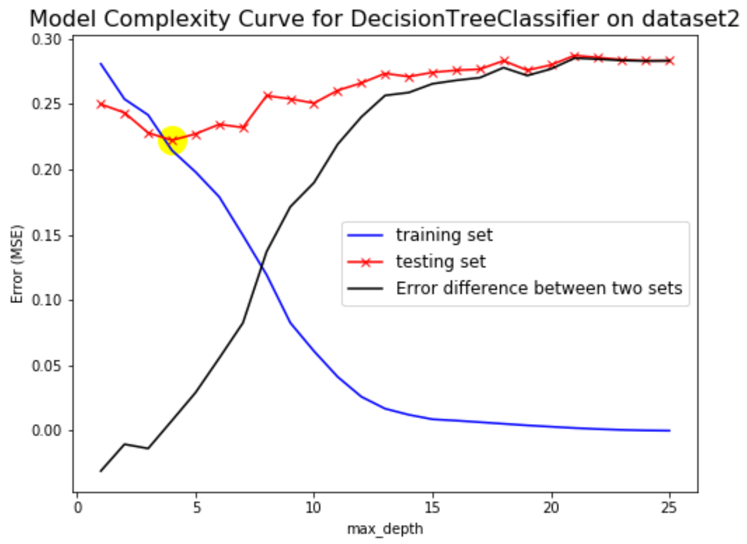


Gradient Boosting Machine is one of the emsembling methods, which in nature is more immune from overfitting than traditional algorithms, and generally gives better performance [14]. Interestingly, the same model trained on the 2 datasets give very different learning patterns for the first time: for dataset1, the model shows a typical pattern of high bias/underfitting; for dataset2, the model gives a typical pattern of high variance/overfitting. Therefore, different strategies should be taken for further improvement. Such as, increasing max_depth for dataset1 and regularizing by adjusting `learning_rate`/ `max_features` / `subsample` parameters on dataset2.

Part IV. Model Complexity Analysis and Optimization

Decision Tree

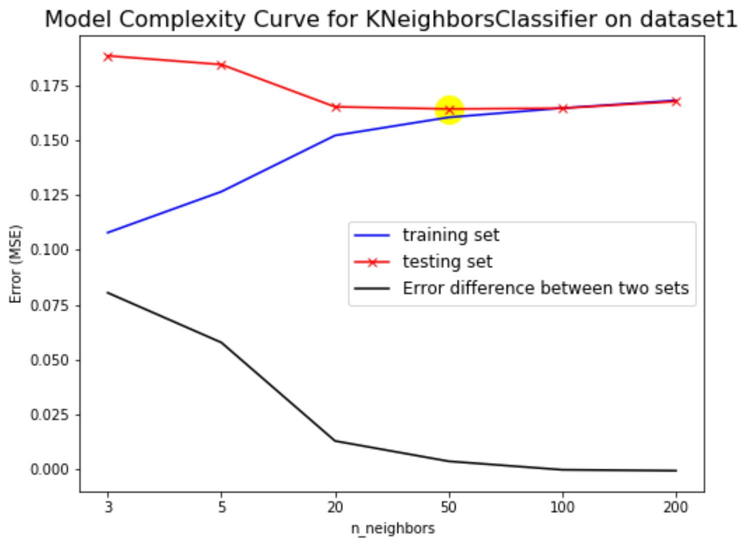


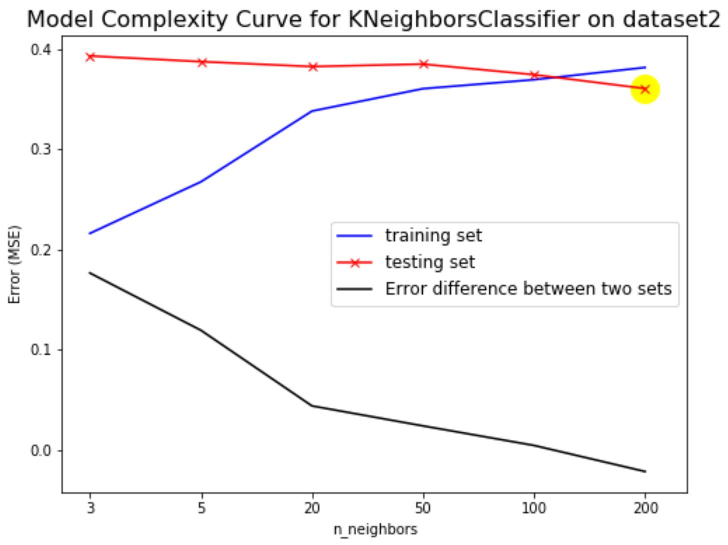


Based on the observations from learning curve section, the decision tree needs to be pruned to improve. Pruning can be realized by changing these parameters: `max_leaf_nodes`, `min_samples_split`, `min_samples_leaf`, `min_impurity_decrease` and `max_depth` [15]. For simplicity, `max_depth` was chosen here. We can see from the curve above that model performance first improves and then worsens as the max_depth, or model complexity goes up. The ideal depth lies somewhere between 1 and unpruned tree, which is 11 and 4 for dataset1 and dataset2.

Besides, decision Tree model runs very fast on both of the datasets. Cross validation is done by StratifiedShuffleSplit on Decision Tree training, as well as on other models throughout the report. Early stopping can be adjusted by `min_impurity_decrease`, specifying a threshold for impurity below which the tree would stop growing there and finalize the corresponding node as a leaf. Thus, this parameter can also be used for pruning.

K Nearest Neighbors

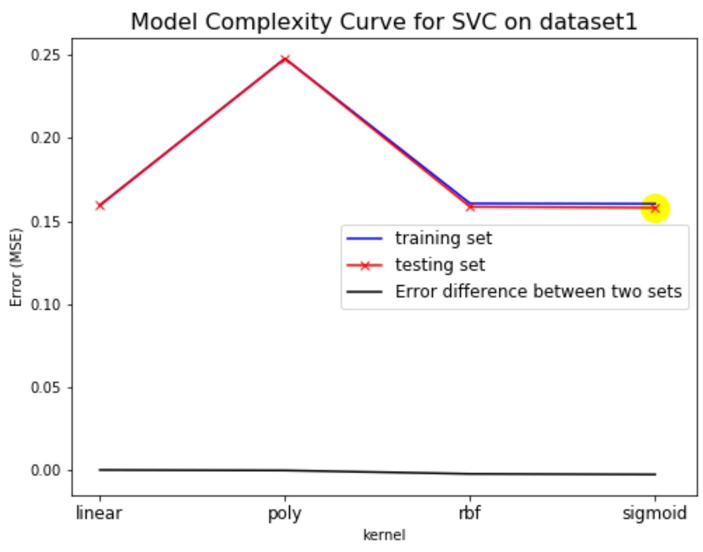


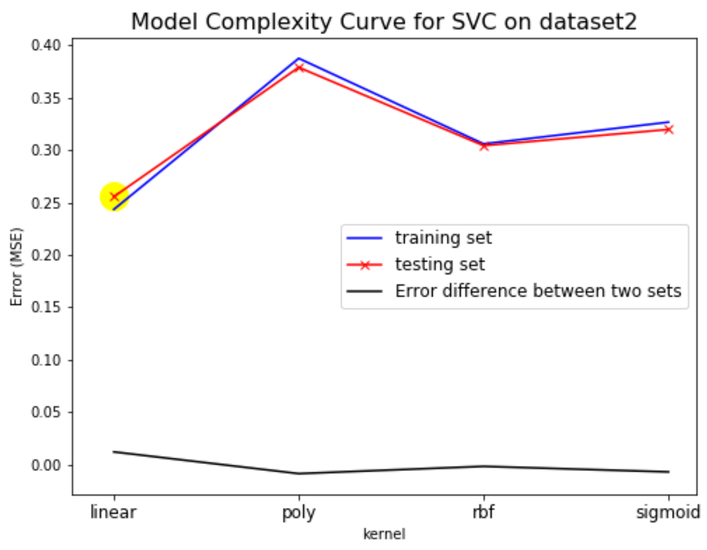


Again, based on the observations on KNN learning curves, we know we need to build models less complex than the default one (K = 5). As `n_neighbors` (K) is the most important parameter controlling KNN model, and a bigger K meaning stronger generalization and more robust to data variance, it's expected increasing K value would give better MSE score. Thus, [3, 5, 20,50,100,200] were chosen for tesing. Notice 3 here is for testing our hypothesis reversely. From the model complexity curve, we found that indeed, MSE drops with K increasing: K = 3 performs poorer than K = 5, while K = 5 poorer than K = 20,50,100 and 200. For dataset1, KNN model suffers from high bias problem since K > 100. Thus, to improve, we should make K = 100. For dataset2, K = 200 allows a better MSE score on testing set than training set, which is strange, K = 100 should be better choice as K = 200 might be an over-generalising model.

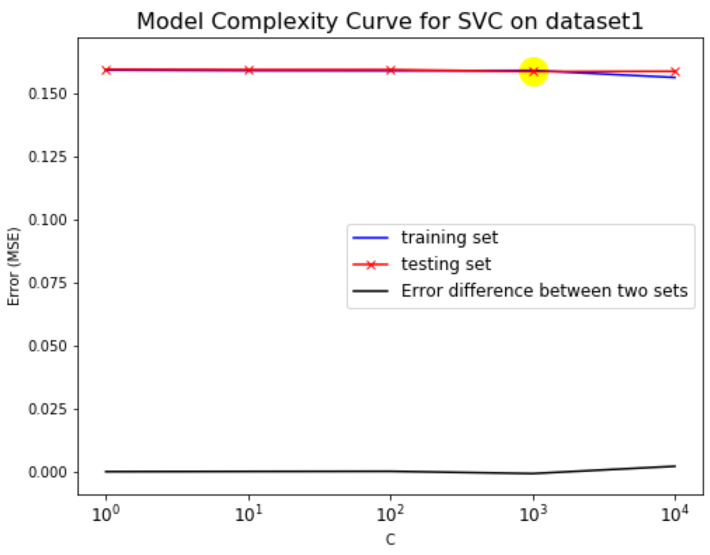
Noticeably, running time for KNN rises up very fast as K value increases. Constructing of KDTree/BallTree is usually efficient, but it's the query processing demanding in computation. With bigger K, the query points stay the same while query task become more difficult as it has to repeatedly finding nearest neighbours in queue for K times.

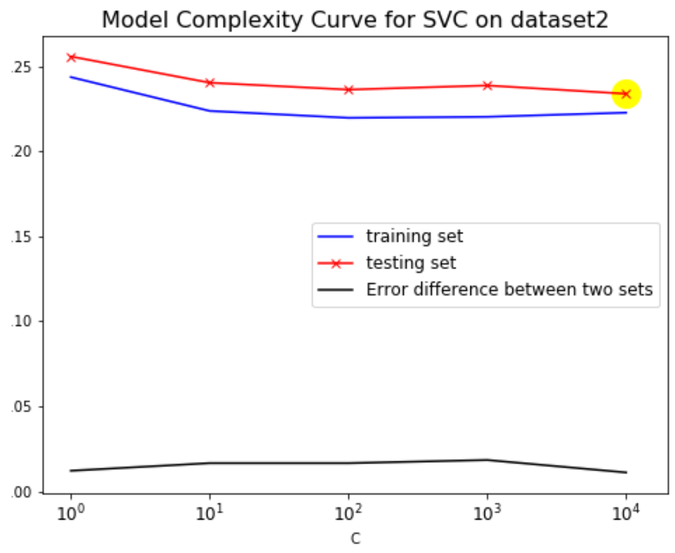
Support Vector Machine





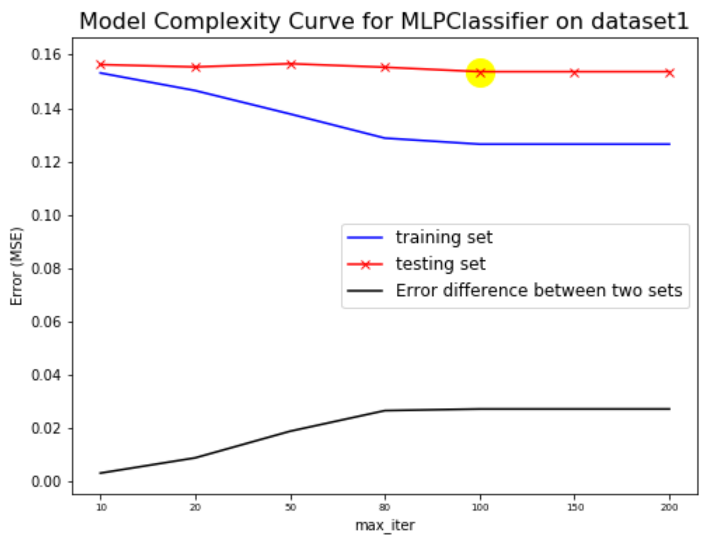
Since the default SVM (rbf) is a high bias model, it's better to first change the algorithm assumption itself to see performance response. For dataset1, we can find that kernel = `linear`/`rbf`/`sigmoid` almost has same MSE score (Sigmoid might have slight advantage). For the sake of efficiency, we can use 'linear' for it. And since linear SVM is the best that outperform the other kernels on dataset2, we can change kernel to `linear` for both datasets and increase C value to make more complex model.

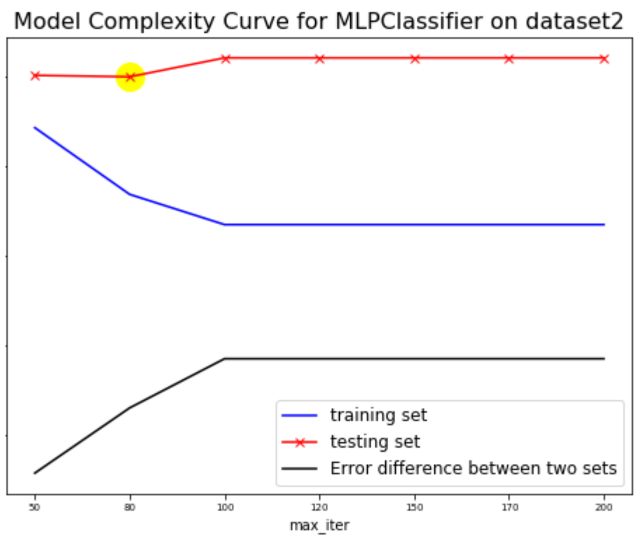




Based on the graph on right, a proper `C` value for linear SVM for dataset1 should be 10000., and for dataset2 be 10000. The very C value penalizes heavily on misclassification thus forces forming of complex models. But curiously, increasing C to even 10000 doesn't seem to increase much of its performance on dataset1. However, the training time has increased significantly for complicated model. Considering the computation cost, C = 1 is chosen for dataset1.

Multilayer Perceptron



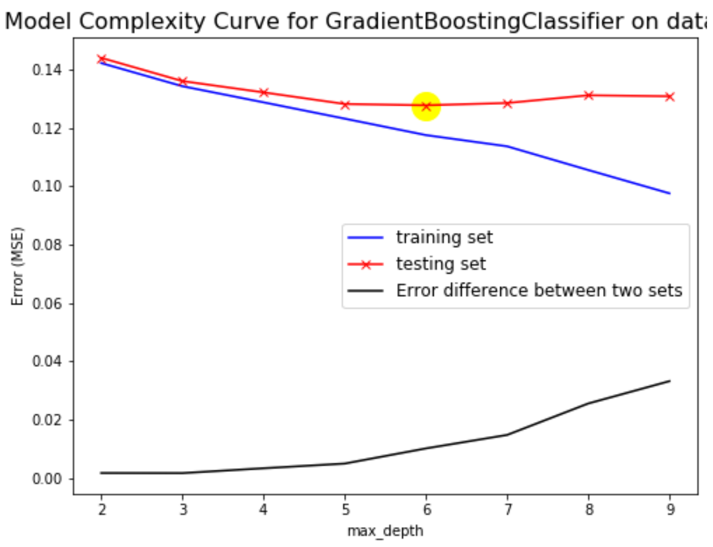


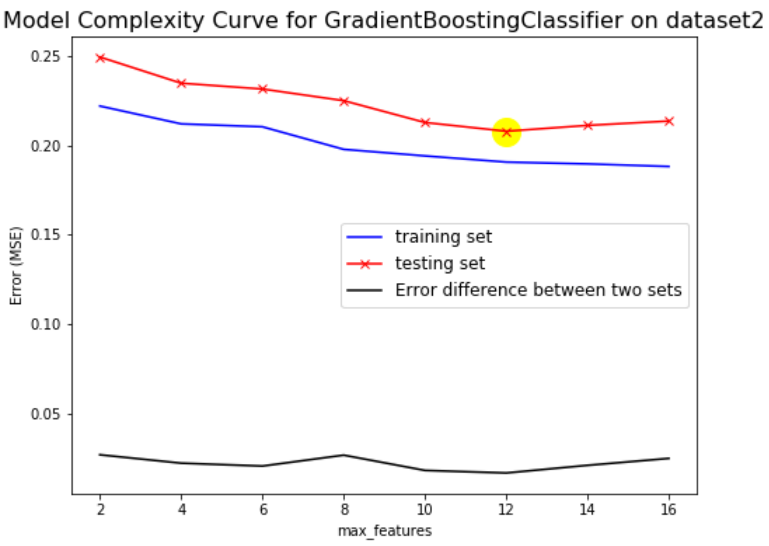
At first, I tried to use different `hidden_layer_sizes` and `alpha` (regularization parameter) to adjust the model complexity of Multilayer Perceptron. However, MLP turns to be quite insensitive to alpha (from 0.001 to 0.01, testing set MSE drops only 0.05). Simply increasing dimensions of hidden layers actually worsen the high variance situation, but curiously, increasing dimensions of hidden layers while reducing the units per layer, improve model performance on both models. However, tuning on these two parameters have only slight effects (results not shown in this report).

By reducing `max_iter`, MLP on dataset2 actually perform better. As shown in the right figure, the error difference between testing and training set has been continuously increasing since max_iter = 50, indicating stronger and stronger overfitting. Though max_iter = 80 is marked as the best performance point, the MSE difference between max_iter = 50 and 80 is actually negligible. It's better to choose max_iter = 50 on dataset2 to prevent overfitting.

For dataset1, pattern of overfitting has shown up since max iterations > 10, indicated by the increasing gap between testing and training error. Herein 10 is chosen for this situation.

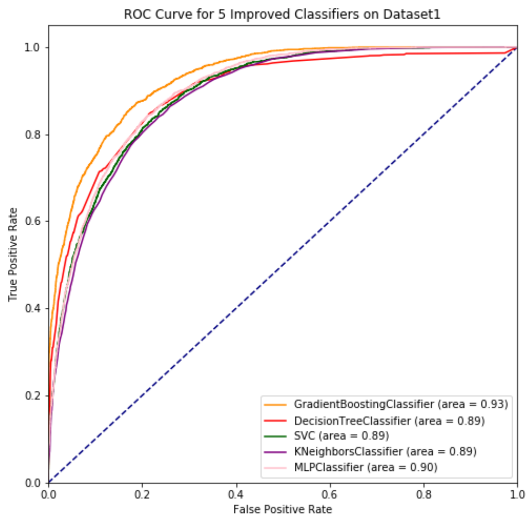
Gradient Boosting Machine

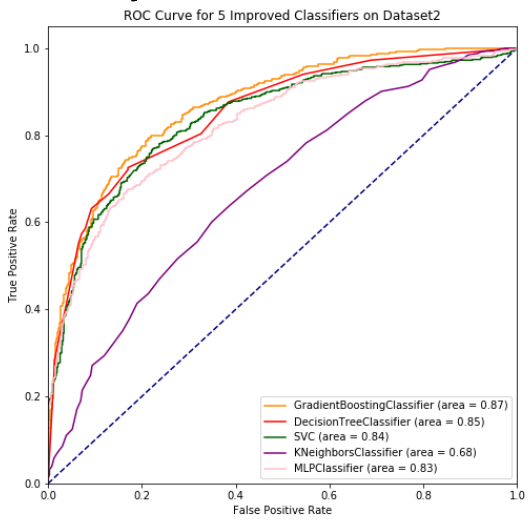




For GBM on dataset2, since it already suffers from overfitting, regularization methods such as decreasing learning rate, max_feature and subsample are taken [16]. Fixing subsample = 0.5 and learning rate = 0.05, by limiting max_features at different values, [2,4,6,8,10,12,14,16], testing MSE gradually drops. Beyond max_features = 12, the model become too complicated that the testing error increases again. Therefore, the best combination of parmater modification discovered for dataset2 is: learning rate = 0.05, subsample = 0.5, max_feature = 12.

Part IV. Summary





On dataset1,
GBM score improved by 0.01;
MLP score improved less than 0.01;
SVC score improved less than 0.01
KNN score improved 0.04
DT improved the most, 0.09.

On dataset2,
GBM score didn't improve;
MLP score improved less than 0.01;
SVC score improved less than 0.07
KNN score improved 0.04;
DT improved the most, 0.13.

It's clearly shown that those classifiers with poor performance at first are also those improved most significantly. After tuning, these improved models have more comparable predicting power. But the only emsembling method GBM is still the one excels. However, notice there are still quite some space for improving since only limited number hyper parameters are tuned in this report.

References:
[1] Udacity-Machine Learning Engineer: Finding Charity Donars
[2] Scaling Up the Accuracy of Naïve-Byes Classification: A Decision Tree Hybird
[3] The Movie Database
[4] Scikit-learn: How to Preprocess data
[5] Scikit-learn: Scaling Features to Range
[6] Scikit-learn: Encoding of Catagorical Features
[7] Why is multicollinearity not checked in modern statistics machine learning?
[8] Machine Learning, Chapter 1. Tom Mitchell
[9] On discriminative vs. generative classifiers: a comparison of logistic regression and naive Bayes.
[10] Scikit-learn: Decision Tree
[11] Scikit-learn: K Nearest Neighbors
[12] Scikit-learn: Support Vecotor Machine
[13] Scikit-learn: Multilayer Perceptron
[14] Scikit-learn: GradientBoostingClassifier
[15] Github: post-pruning for Decision Trees
[16] Regularization of GradientBoostingClassifier