CS7641 Assignment 1

Sai Chen

**Section 1. Descriptions of the datasets**

In this analysis, two datasets are used to compare the performance of various machine learning methods: hand-written digits dataset and Wisconsin breast cancer dataset. Both are built in datasets that can be directed loaded within scikit-learn python package. The reasons of choosing the two datasets are listed below.

1. Hand-written digits dataset is a widely used standard toy dataset for evaluating different machine learning algorithms. The center goal of mining this dataset is to build efficient and accurate algorithms to characterize digits written by human beings, initially from the need of recognizing information in standard mailing services. Besides the practical use of this dataset, it also provides a good platform for comparing the performance of different algorithms on extracting local information in images. Unlike learning problems with weakly correlated attributes (or covariates), being able to deconvolve spatial correlations of features have significant importance in building a successful algorithm to recognize patterns in images. Based on this assumption, it would be interesting to know how tree-based methods perform on this dataset compared to other methods, since decision tree has a linear structure in the classifying process.
2. In recent decades, machine learning has seen its broad range of applications in disease diagnosis based on medical records. The Wisconsin breast cancer dataset contains features computed from images of breast specimens and the matching diagnostic conclusions. Since I’m extremely interested in utilizing machine learning methods to analyze medical records, this dataset offers a good opportunity to evaluate the performances of different learning methods on making decisions in the context of disease diagnosis.
3. These two datasets together represent two different categories in learning problems. The breast cancer dataset is more like classical non-linear regression problem in which several potential causal attributes as well as confounding factors are given for each class label. On the other hand, each single image pixel is unlikely to be the real “attribute” used to recognize class labels. Instead the algorithm has to somehow be able to pick up the key information from the pixel patches. It is then interesting to compare performances of a single algorithm on evaluating two different types of datasets and to know the applicable range of the algorithm.
4. Simplicity is another important reason for picking the two datasets. Both sets have restricted while descent amount of training data that can be used for a non-trivial amount of cross-validation. The features in each dataset are pre-processed and contain no missing values. By this simplicity, it saves me time from looking for good strategies on data processing so that I could focus on evaluating approaches that are computationally intensive. Originally, I used the complete MNIST digits dataset and another 20-year-long heart disease dataset for this assignment and I cannot even finish a single round of grid search for good parameters using cross validation. Even though data pre-processing is very important in real problems, I still benefit from using the two “toy” datasets with a deeper level of computations.

The complete descriptions of the two datasets can be found on the scikit-learn webpages. Below I will list some basic statistics on the size of the two datasets.

**Digits recognition**: this dataset contains **5620** training samples. Each sample is an **8x8** pixel-wide image, with value of each pixel ranging from **0** to **16**. For each sample, a class label is given to indicate which digit out of 0-9 this image represents. This is a multi-class classification problem. The sample size for each class is given in Table 1 below.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| digit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| size | 178 | 182 | 177 | 183 | 181 | 182 | 181 | 179 | 174 | 180 |

Table 1. sample size for each class in digits dataset

**Wisconsin breast cancer**: this dataset contains 569 training samples. Each sample contains 30 numeric predictive attributes that are computed from images of breast mass specimen. A class label is given to each sample that indicates whether the associated individual has breast cancer. There are **212** malignant samples and **357** benign samples, respectively. The summary statistics could be found on the scikit-learn website.

**Section 2. Packages and methods**

I used scikit-learn (version 0.21.2) package for this project. All computations are performed in python (version >= 3.6.8). Usage docs are provided for all functions written by me.

I performed 5 learning algorithms on each dataset: a simple one-layer linear neural network, K nearest neighbor with different K values, a simple decision tree using Gini impurity for splitting with various levels of pre-pruning (early stopping) and post-pruning (simple reduced error pruning), an adaboost ensemble method using the previous fitted decision tree as the base learner, support vector machine using two different kernels (“linear” and “polynomial”). The details are described below in the **result** section.

The basic evaluation approach for each method is plotting the learning curve, namely prediction accuracy against size of the input training samples. For the two iterative fitting algorithms neural network and SVM, a plot of prediction accuracy against iteration numbers is generated for each method. The **prediction accuracy** is measured as the proportion of correct classifications upon test dataset (generated by cross validation). For digits recognition dataset, the accuracy is summed over all classes. Specifically, the **score()** attribute function in each scikit-learn learner object is used to generate the accuracy result.

The learning curve of accuracy v.s. size is generated using the scikit-learn function **learning\_curve()**. This function takes a parameter **cv** to perform cross validation. The prediction accuracy score over each sample size is calculated in a K-fold manner, in which the mean of the K different accuracy scores are averaged to generate the final result. Standard error is also calculated in a similar manner. Since the breast cancer dataset has <1,000 samples, the first sample size used is >50% of total size and K is set to 3 for most methods, to avoid cases in which training samples contain only one class label.

The learning curve of accuracy v.s. iteration numbers is generated using a homemade function. Since I did not find ways to extract accuracy history during model fitting, I used a brute-force method to calculate the accuracy score over each specified interval. I set the **max\_iter** parameter to various numbers and compute accuracy of the model under each iteration number. To reduce randomness during fitting, I set **random\_state** to a fixed number. However, this is definitely not ideal and cannot avoid all the randomness in the fitting. The final result should be interpreted in the point of view of average behavior and only approximately represent conclusion from a single run with different numbers of iterations.

There are other types of analysis generated specifically for each model, which will be discussed in the **result** section.

**Section 3. Result**

**Section 3.1. Breast cancer dataset**

Below I will go through my analysis on the Wisconsin breast cancer dataset

***Section 3.1.1. Neural Network***

A one layer neural network (NN in below) is fitted to the dataset using the function **MLPClassifier().** NNs with two or three layers and similar numbers of nodes in each layer have been tested and in fact decreased the prediction accuracy over test sets (data not shown here for saving space). To find the best number of nodes, I used k-fold cross validation (k = 3) (Figure 1). The model accuracy is not affected from 25 to 60 nodes. For simplicity, I used 25 nodes for the below analysis.

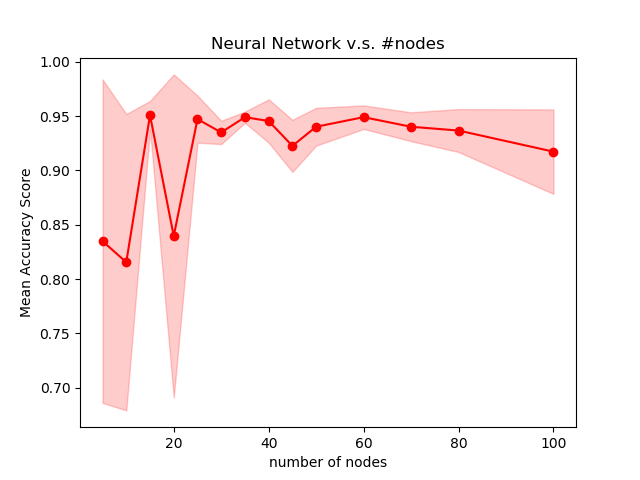
It is recommended that for small dataset, the “lbfgs” optimization methods is better than the default “sgd” or sgd variant “adam”. This is indeed the case in my experimentation using k = 3 cross validation (lbfgs = 0.945, sgd = 0.90, adam = 0.93). Thus, in my final evaluation I used the **model with one layer of 25 nodes and lbfgs to fit the data**. Similar reasoning applies for choosing “**logistic**” activation in my final implementation.

Figure 1. Accuracy v.s. number of nodes

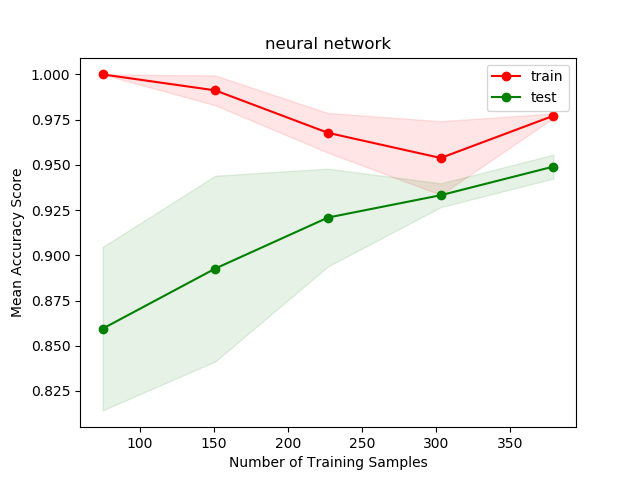
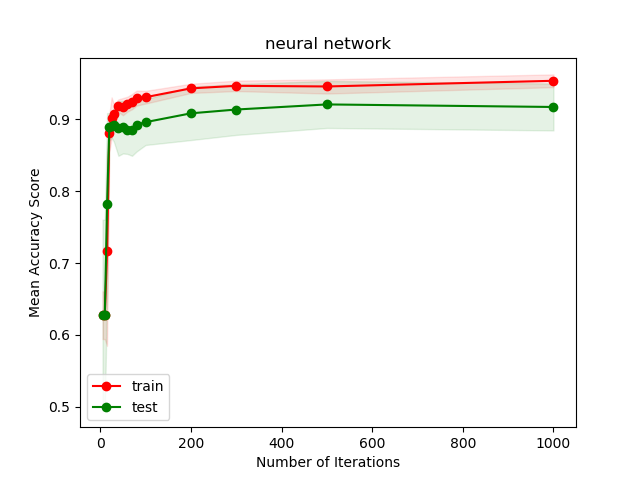
Figure 2 shows the learning curve for the neural network. As we can see from the figure, the accuracy decreases for the training set with larger sample size, while it is the reverse trend for test set. This is caused by overfitting when sample size is small while prediction accuracy increases with more and more data. The classification rate of success is ~92%. The comparison of different models will be presented at the end of this section.

Figure 2. Learning curve (size) for neural network

Figure 3 shows the learning curve over iteration numbers for the neural network. The model converges at ~500 iterations, with minor improve on prediction accuracy after 600 iterations. This is especially true for prediction accuracy on test dataset. Again, the result suggests a ~92

% prediction accuracy after 1000 iterations.

It might be possible to fine tune the model further by changing other parameters, such as L2 penalty rate and learning rate scheduling. Here I only explore a few key parameters. The result suggests that optimization method, model complexity (number of nodes) and activation function are important for the performance of neural network.

Figure 3. Learning curve (#iter) for neural network

***Section 3.1.2. K Nearest Neighbor***

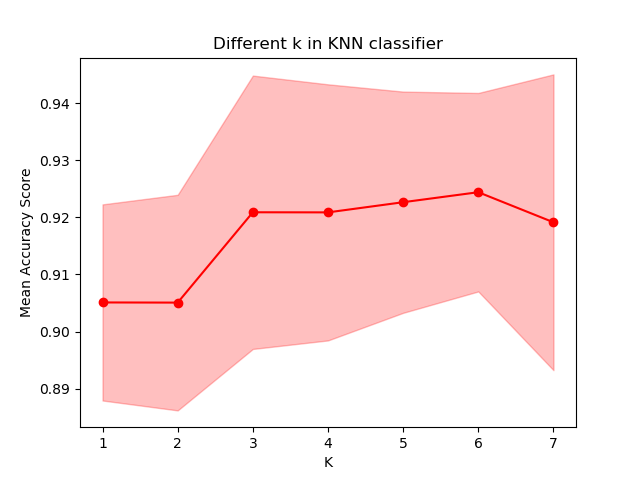
To fit K Nearest Neighbor (KNN) model to the breast cancer data, it is important to use the best K value. In Figure 4, the cross-validated prediction accuracy scores (k = 3) are plotted against different k values to find the best k. From the figure, we can see that for this dataset, models with k = 3, 4, 5, 6 all perform similarly in the range of randomness of the fitting process. So, **I picked K = 6 for my final model (with all the other parameters set to default)** and fitted the model against the data to generate learning curve.

Figure 4. Accuracy score using different k values

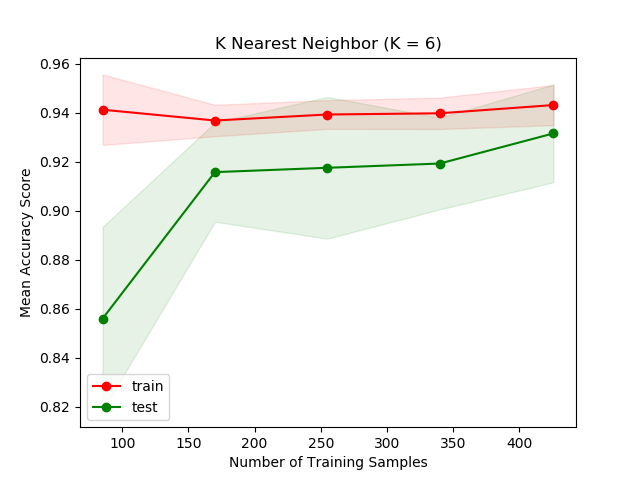
Figure 5 shows the learning curve over size for the KNN method. Again, the learning curve is generated using cross validation to get standard errors (k = 4). We can see that prediction accuracy does not have a strong change for training data, as it fluctuates around 94%. However, the prediction accuracy on the test model in cross validation largely increases with more data. This suggests that there is less concern on overfitting or data under-representation issues.

Figure 5. Learning curve (size) for KNN

***Section 3.1.3. Decision Tree***

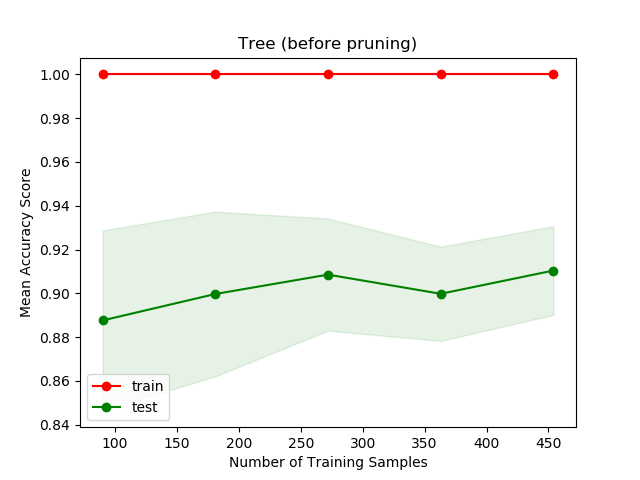
To fit a decision tree model to the data, I first used the default parameters provided by scikit-learn function. Figure 6 shows the learning curve (generated with cross validation, k = 5). As we can see from the figure, the model has a strong overfitting issue: it can fit the training data with 100% accuracy but the accuracy for test data doesn’t improve much after the model has seen ~50% of the total data. This means that some type of tree pruning is needed to account for the overfitting. To do that, I used both pre-pruning and post-pruning to adjust the tree. First, I tuned the parameters of the fitting function to force the tree-building algorithm to stop early before it can correctly classify every training sample. This is a common approach used for pre-pruning. There are three parameters that one can use: **max\_depth**, the maximum depth of the tree; **min\_samples\_split**, the minimum number of samples required to perform a split; **min\_samples\_leaf**, the minimum samples required for the leaf node. I did a grid search over some parameter space (boundary defined by the default tree) to find the best combination based on cross validation (k = 4) and found that the most important factor is **min\_samples\_leaf.** There are a few parameter combinations that have similar performances. **I picked the simplest one as my final model (4, 8, 8, respectively, same order as above)**. The accuracy increases from 91% to 94%. Figure 7 shows the learning curve for this model.

Figure 6. Learning curve (size) for decision tree

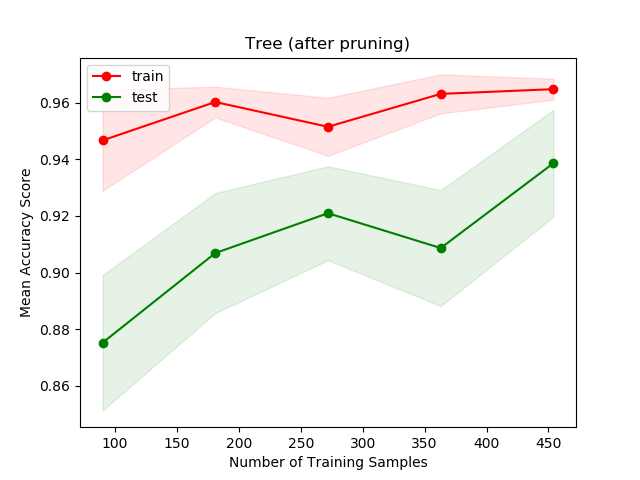
We can also do post-pruning. Here I implement the simplest post-pruning method, the reduced error pruning. I used a greedy algorithm so that from all the leaf nodes, any subtree that does not decrease prediction accuracy when changed into a leaf node is pruned into a leaf, without backtrack. The pruned model gives higher accuracy of 95% over pre-pruned tree. However, due to the restricted tree structure used in scikit-learn, I didn’t have time to implement a new learning curve function. However, the improve upon the overall prediction accuracy is obvious from the analysis.

Figure 7. Learning curve (size) for decision tree after pre-pruning

***Section 3.1.4.*** ***Ada-boosting tree***

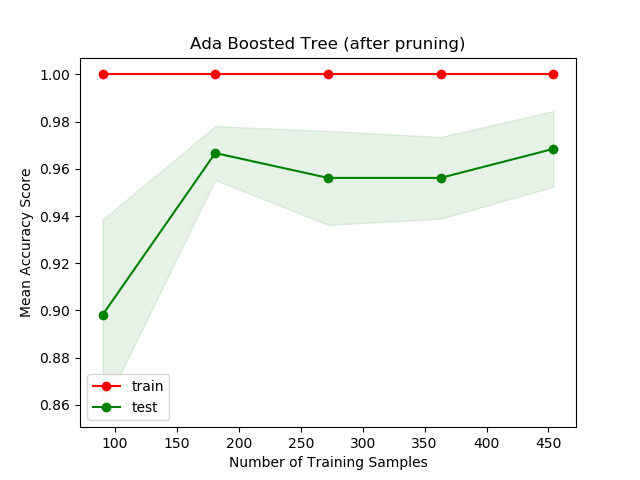
I used Ada-boosting to fit an ensemble model over the decision tree fitted in the last section. Again, I used the same pre-pruning method to search for the best combination of early-stop parameters for the boosting tree. Due to the space limitations I will only show the learning curve for the pruned tree. Prediction accuracy improves drastically over regular decision tree model (95% to 97.5%). Since the adaboost method is an ensemble method, it is not surprising that it can perform better than simple tree method.

Figure 8. Learning curve (size) for adaboost tree after pruning

***Section 3.1.5. Support Vector Machine***

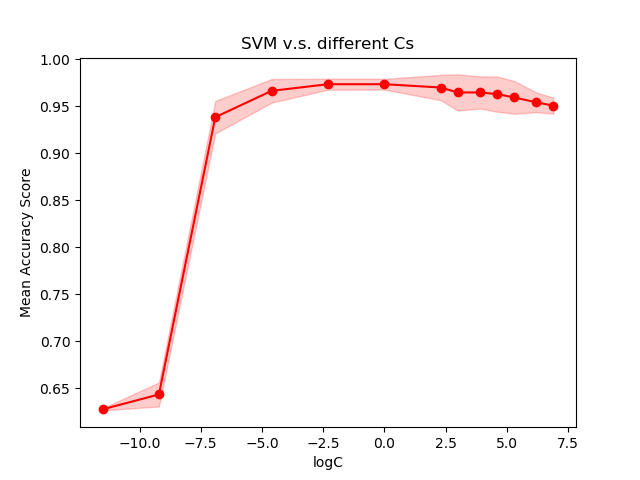
I used two different kernels for fitting SVM models: “linear” and “polynomial”. Before performing model fitting, I scaled the features so that each column in the data has zero mean and unity variance. For SVM, one of the important parameters to be determined is the penalty term over error terms. I used cross validation (k = 4) to determine the best penalty value. Figure 9 shows the result for linear kernel. The best C is 0.1. Similar analysis was done on polynomial kernel and the best is determined in a similar fashion (C = 100).

Figure 10 and Figure 11 show the learning curve for linear and polynomial kernels, respectively. We can see that in this context, linear kernel does a much better job.

Figure 9. Accuracy Score v.s. C’s

|  |  |
| --- | --- |
|  |  |
| Figure 10. Learning curve (linear kernel) | Figure 11. Learning curve (polynomial kernel) |

***Section 3.1.6. Model comparison***

Here is a summary of all the models, their final accuracy and their running time.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model | Neural Network | KNN | Tree | Boosting Tree | SVM linear | SVM polynomial |
| Specification | one layer,  #nodes=25 | K=6 | Reduced error pruning | Pre-pruning with min\_samples\_leaf=8 | C=0.1 | C=100 |
| Accuracy | 92% | 93% | 94% | 96% | 97% | 87% |
| Time | ~1s | ~1s | ~0.01s | ~0.2s | ~0.001s | ~0.01s |

Using proportion of correct classification as the metric, the boosted decision tree and linear-kernel SVM have the best performance within all the available methods. SVM with linear kernel also has a very fast optimization time. This result is not surprising. Considering that patients with breast cancer have well-separated structural features in their specimen images, SVM can efficiently find the separating plane in the feature space. On the other hand, ensemble methods have better performance than other single estimator methods if the underlying base estimator has a not-too-bad performance on the dataset.

**Section 4. Digits recognition dataset**

The exact