**2 Programming**

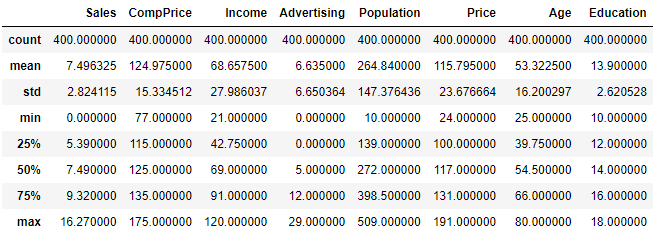
**2.1 Decision Tree**

In this question, we implement Decision Tree, Bagging of Trees and Random Forests to predict the sale of the Carseats dataset. All the algorithms can be implemented by sklearn, the loss will be set as MSE.

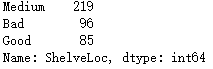
**2.1.1 Data Statistics**

There are 11 columns of the dataset (including sales), 8 of them are numeric type, 3 of them are string type (‘ShelveLoc’, ’Urban’, ‘US’).

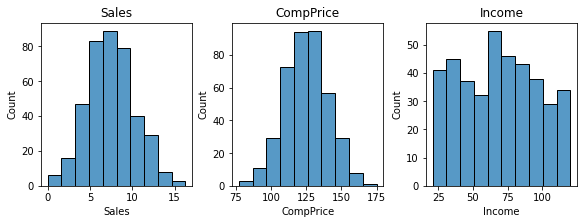
The details of the numerical data are as follows:

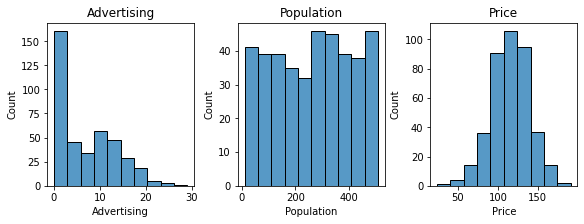


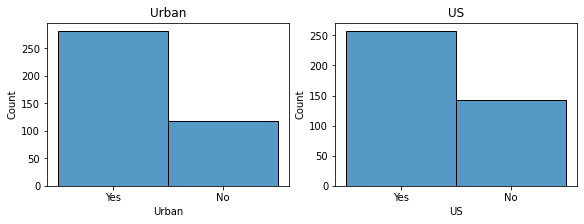
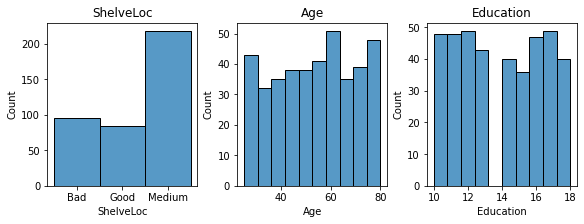
The details of the categorical data are as follows:



Also, I draw the histograms of each column, and use seaborn to visualize them.







Data Preprocessing:

1. Since there’re 3 categorical variables, we cannot use it directly to do regressions. It’s straightforward to think about one-hot encoding in this scenario since each of the three categorical variables has less than or equal to 3 categories. Thus, I use one-hot encoding to change the variable. There’s also no need to do normalization since decision tree is not sensitive to the scale of the numerical data.

2. For the training and testing data, I choose the first option, that is, I use the first 300 rows as the training set, and the remaining 100 rows as the testing set.

**2.1.2 Decision Tree**

First, I use GridSearchCV offered by sklearn to search for the best maximum depth and least node size. The result is as follows:

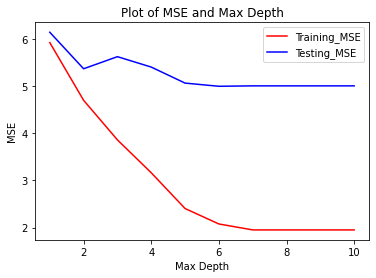


Then, I define a function decision\_tree (maxdepth, minleafsamples), which use DecisionTreeRegressor in sklearn to do the task and return its training and testing mse by metrics.mean\_squared\_error ().

Result Analysis:

1. First, I change maximum depth from 1 to 10, while maintaining least node sizes as 9, according to the GridSearchCV.

The result is shown below:

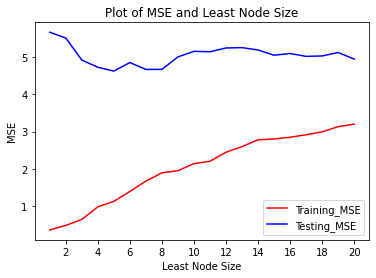


From this figure, we find that with the increase of max depth, which means that model complexity is increasing, the train error always decreases. The test error decrease at

first and start be stable when the depth is around 5. It proves that when the complexity is too small, the model will be underfitted, both training and testing error is high. When the complexity is too large, it tends to be overfitted. The training error is smaller when increasing the max depth, but the testing error is stable.

2. Then I change least node size from 1 to 20, while maintaining maximum depth as 8, according to the GridSearchCV.

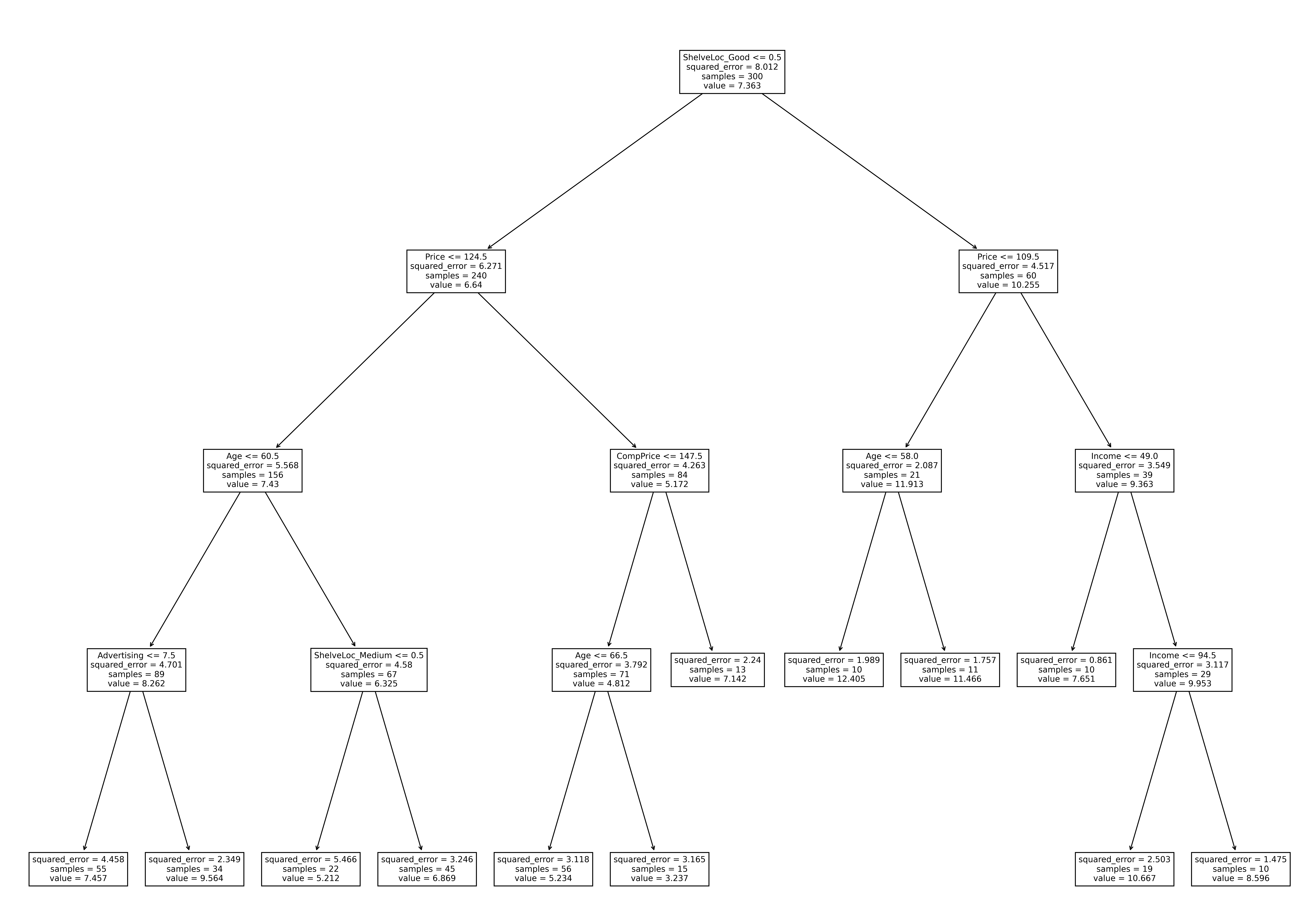
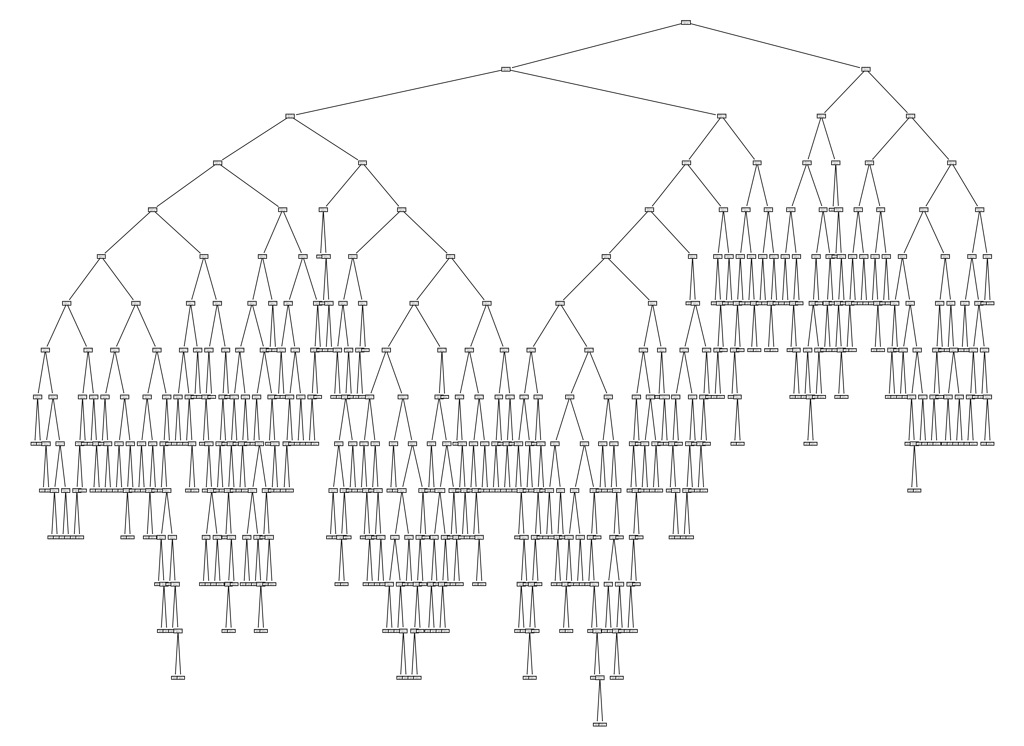
The result is shown below:



From this figure, we find that with the increase of node size, which means that model complexity is decreasing, the training error always increases. The testing error decreases first and then increases. This proves that when the node size is small, the model tends to be overfitted, when the node size is large, the model tends to be underfitted.

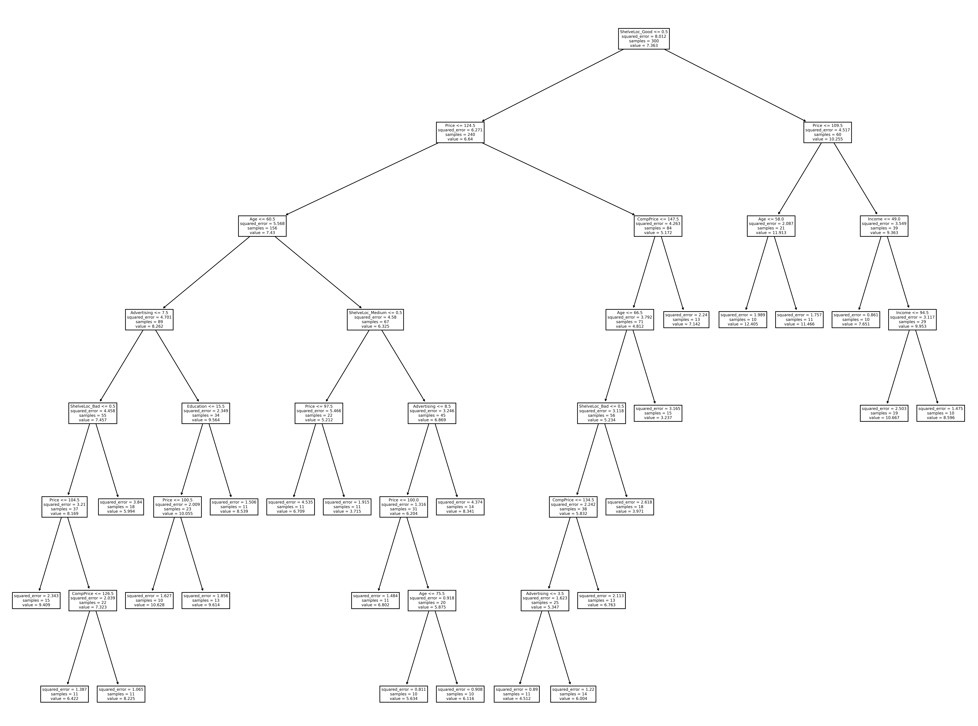
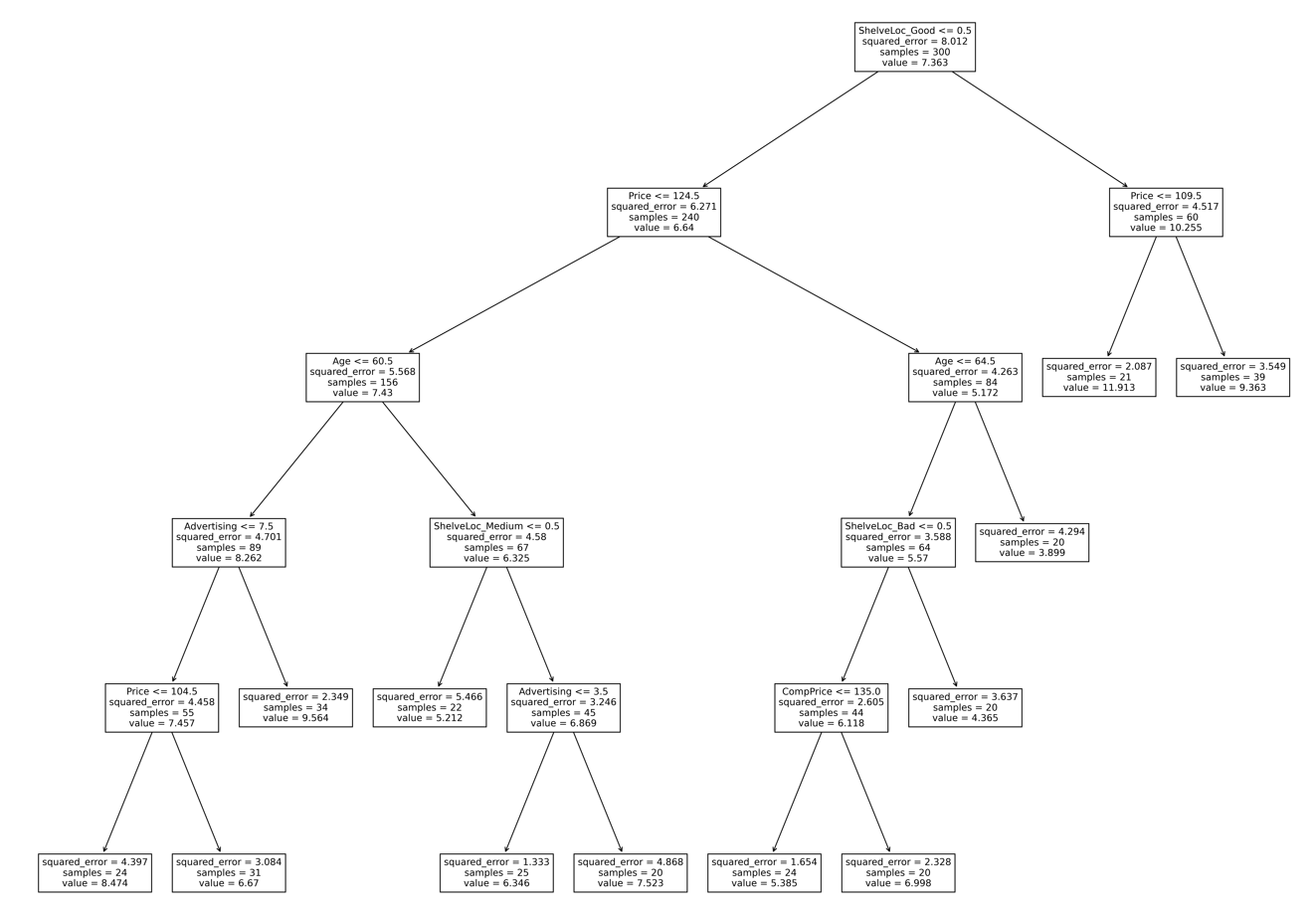
Below are some of the Decision Tree plots, each of which has different set of parameters.

(If the plot shown below is hard to recognize, you can directly find them in the assignment folder, whose names are p1.png, p2.png, etc. They will be much clearer.)



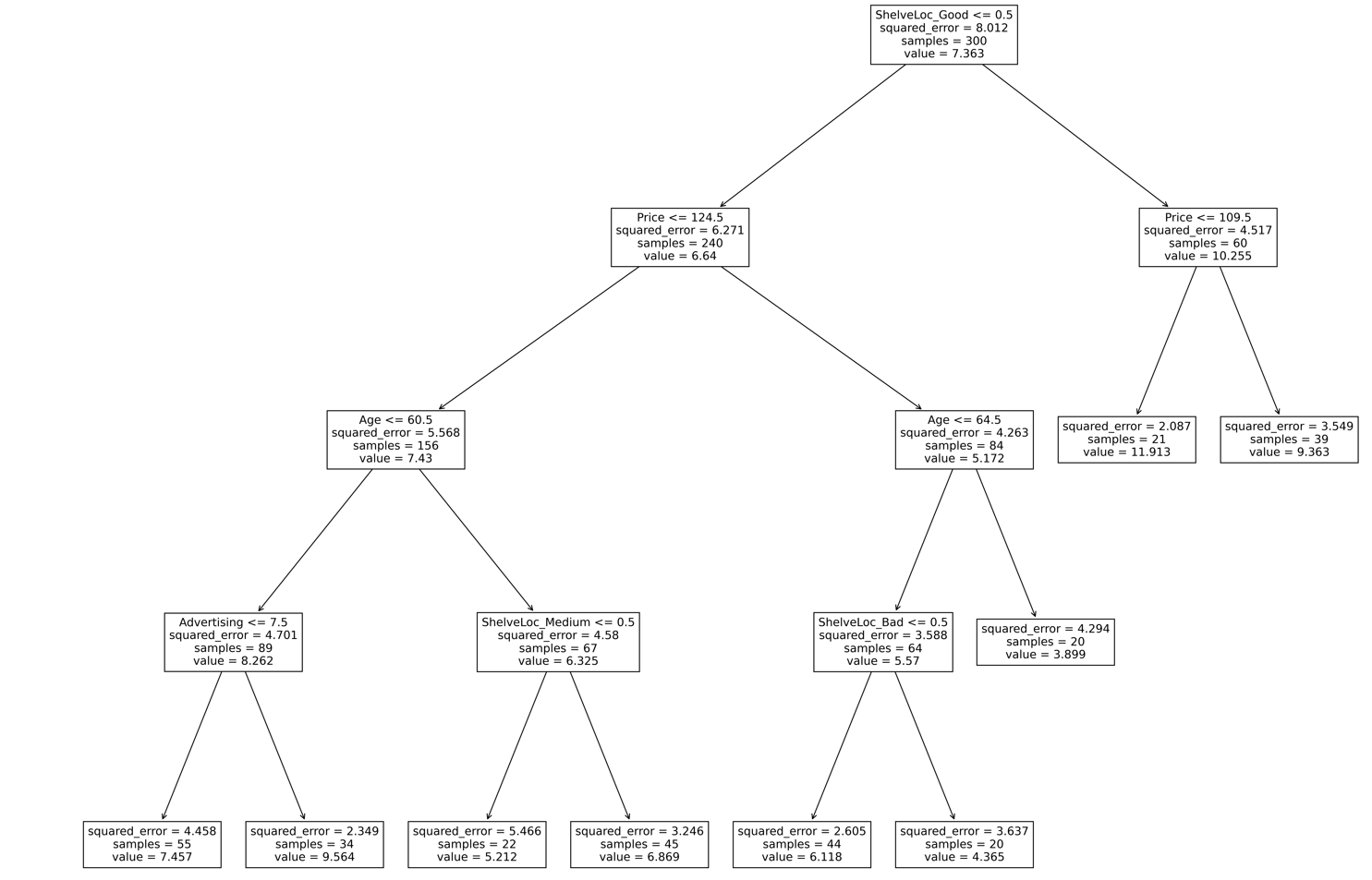
Default tree

Depth = 4, Leaf = 10



Depth = 8, Leaf = 20

Depth = 8, Leaf = 10



Depth = 4, Leaf = 20

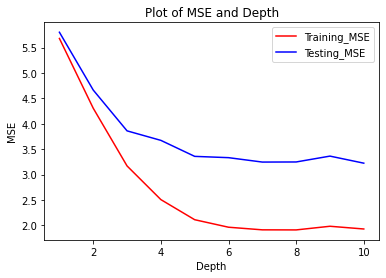
**2.1.3 Bagging of Trees**

First, I define a function bagging (depth, numtrees), which use BaggingRegressor in the outer space and DecisionTreeRegressor in the inner space from sklearn to do the task and return its training and testing mse by metrics.mean\_squared\_error ().

Result Analysis:

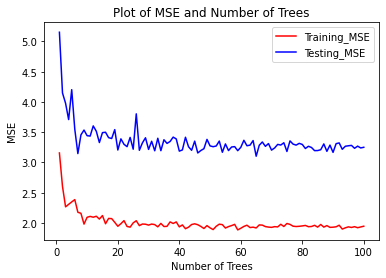
1. First, I change maximum depth from 1 to 10, while maintaining the number of trees as 50.

The result is shown below:



From this figure, we find that with the increase of depth, which means that model complexity is increasing, the training error always decreases. The testing error decreases at first and start to be stable when the depth is larger than 6. It proves that when the depth is too small, the model will be underfitted, when the depth is too large, the model might be overfitted. (E.g. Depth = 100)

2. Then, I change the number of trees from 1 to 100, while maintaining the maximum depth as 6. The result is shown below:



From this figure, we find that when the number of trees increase, the training error and testing error start to decrease at first and then be stable. The number of trees means data-level randomness, has no relationship with complexity since each tree may have roughly the same depth.

**2.1.4 Random Forests**

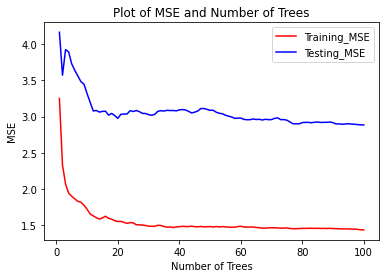
First, I use GridSearchCV offered by sklearn to search for the best number of trees, value of m, maximum depth and least node size. The result is as follows:



Then, I define a function random\_forests (numtrees, m), which use RandomForestRegressor in sklearn to do the task and return its training and testing mse by metrics.mean\_squared\_error ().

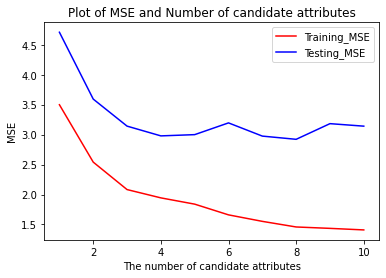
Result Analysis:

1. First, I change the number of trees from 1 to 100, while setting max\_depth as 7, m as 7, min\_samples\_leaf as 5, according to the GridSearchCV. The result is shown below:



From this figure, we find that when the number of trees increase, the training error and testing error start to decrease at first and then be stable. The number of trees means data-level randomness, has no relationship with complexity since each tree may have roughly the same depth in the whole random forest, for example, they may have the depth of only 2 or 3.

2. Then, I change value of m from 1 to 100, while setting max\_depth as 7, number of trees as 20, min\_samples\_leaf as 5, according to the GridSearchCV. The result is shown below:

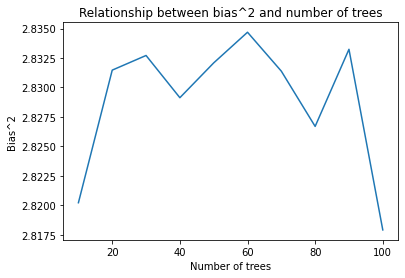


From this figure, we find that with the increase of candidate attributes, the training error is always decreasing. The testing error decreases at first and start to be stable when m = 3 or 4, which is same as we discussed in lecture slides that m = N/3 is always a good choice.

**2.1.5 and Variance**

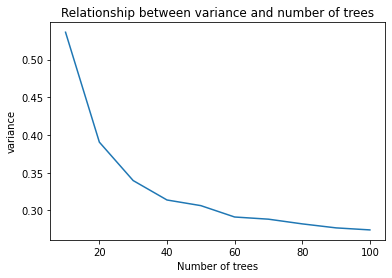
In this question, I use bias\_variance\_decomp () provided by mlxtend to get the average bias and average variance throughout the whole dataset.

Below is the curve of bias w.r.t the number of trees:



From the figure, we can obtain that there’s no obvious relationship between bias and the number of trees, which is consistent with the statement provided in lecture slides that different trees in the random forests are independent and the overall model complexity is not increased, so we can’t expect increasing the number of trees can reduce bias.

However, for the curve of variance and the number of trees, it’s different:



From the figure, we can see that increasing the number of trees can indeed decrease the variance. Since random forests introduce both data-level randomness and model-level randomness (From random feature selection), so there is a guarantee that variance will be reduced.

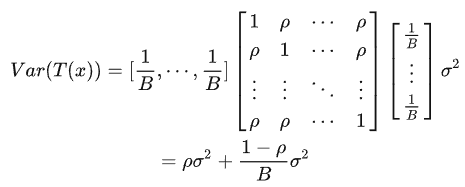
Mathematical Proof:

We can write a random forest as the formula below:

where B is the number of trees, is the i-th decision tree, is the training sample.

We can easily obtain that for a certain x, follows the same distribution, so increasing the number of trees has no contribution to reduce the model bias.

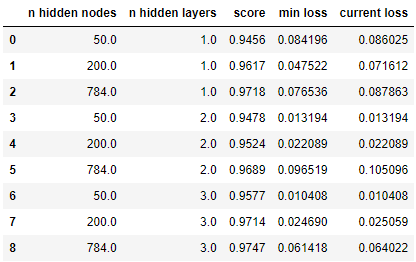
However, is dependent to each other, since the samples are drawn from the same dataset, and they use the same algorithm to get each decision tree. If we set the correlation coefficient to be , and variance of each tree to be , we can get the variance of the whole random forest:



From the formula derived, we can see that if we increase the number of trees B, we can significantly reduce the variance of the random forest. In conclusion, our conclusions from processing the dataset and mathematical formulation are the same.

**2.2 Handwritten Digit Recognition**

In this question, I use load\_mnist.py to load the train/test data, and use MLPClassifier from sklearn to construct a Multi-Layer Perceptron neural network, with hyperparameters: (solver = 'adam', activation = 'relu', alpha = 0.05, random\_state = 3020, max\_iter = 100, verbose = False, learning\_rate\_init = 3e-4, batch\_size = 512), which are figured out by GridSearchCV. The result is shown below:



From the chart, we can see that if we fix the number of hidden nodes, with the increase of the number of hidden layers, we may not have a higher prediction accuracy. In contrast, if we fix the number of hidden layers, with the increase of the number of hidden nodes, we would have a higher prediction accuracy.