2 Programming

1. Decision Tree (required)

Task description Fit (*i.e.*, regression) the real variable *Sales* in the **Carseats** dataset, using decision tree, bagging, and random forests. All these algorithms can be implemented by calling **sklearn** in Python. The loss is set as sum of squared error (SSE).

Dataset **Carseats** contains 400 data points (saved in 400 rows). For each data, the first column is the value of target variable *Sales* that we want to fit; the remaining 9 columns indicate 9 features (or attributes), as shown in Fig. 1. There is no fixed train/test splitting. In this project, you have two options: simply set the first 300 rows as the training set, and the remaining 100 rows as the testing set; randomly split the whole dataset to 300 train + 100 test, and try multiple times. The dataset can be obtained from `https://github.com/selva86/datasets/blob/master/Carseats.csv`

What you should do

- **Data statistics**: analyze the statistics of the target variable and each feature, and try to visualize the statistics (*e.g.*, histogram) (0.5 point)
- **Decision tree**: solve the above problem using decision tree method; report the train/test errors with respect to different maximum depths, different least node sizes; plot the learned tree (2 points)
- **Bagging of trees**: solve the above problem using the bagging method, with decision tree as the base learner; report the train/test errors with respect to different depths, different number of trees (2 points)
- **Random forests**: solve the above problem using the random forest method, with decision tree as the base learner; report the train/test errors with respect to different number of trees, different values of m (the number of candidate attributes to split in every step, see Slides 'W7-Decision Tree', Page 68) (2 points)
- Plot the curve of bias^2 with respect to different number of trees in random forests, *e.g.*, $\#tree = 10, 20, \ldots, 100$. Then, describe the relationship between bias^2 and different number of trees; repeat the procedure for variance. (1.5 points)

Carseats

Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
9.5	138	73	11	276	120	Bad	42	17	Yes	Yes
11.22	111	48	16	260	83	Good	65	10	Yes	Yes
10.06	113	35	10	269	80	Medium	59	12	Yes	Yes
7.4	117	100	4	466	97	Medium	55	14	Yes	Yes
4.15	141	64	3	340	128	Bad	38	13	Yes	No
10.81	124	113	13	501	72	Bad	78	16	No	Yes
6.63	115	105	0	45	108	Medium	71	15	Yes	No
11.85	136	81	15	425	120	Good	67	10	Yes	Yes
6.54	132	110	0	108	124	Medium	76	10	No	No
4.69	132	113	0	131	124	Medium	76	17	No	Yes
9.01	121	78	9	150	100	Bad	26	10	No	Yes
11.96	117	94	4	503	94	Good	50	13	Yes	Yes
3.98	122	35	2	393	136	Medium	62	18	Yes	No
10.96	115	28	11	29	86	Good	53	18	Yes	Yes
11.17	107	117	11	148	118	Good	52	18	Yes	Yes
8.71	149	95	5	400	144	Medium	76	18	No	No
7.58	118	32	0	284	110	Good	63	13	Yes	No
12.29	147	74	13	251	131	Good	52	10	Yes	Yes
13.91	110	110	0	408	68	Good	46	17	No	Yes
8.73	129	76	16	58	121	Medium	69	12	Yes	Yes
6.41	125	90	2	367	131	Medium	35	18	Yes	Yes
12.13	134	29	12	239	109	Good	62	18	No	Yes
5.08	128	46	6	497	138	Medium	42	13	Yes	No
5.87	121	31	0	292	109	Medium	79	10	Yes	No
10.14	145	119	16	294	113	Bad	42	12	Yes	Yes

Figure 1: Some examples of **Carseats**.

2. Handwritten Digit Recognition using sk-learn (4 points)

Task description Use the function "MLPClassifier" of sk-learn to construct a fully connected network to classify the MNIST data. The dataset can be downloaded from `http://yann.lecun.com/exdb/mnist/`. The training-testing spilliting has been given. Show the performance of your neural network with different structures:

- number of hidden layers chosen from $\{1, 2, 3\}$
- number of hidden nodes chosen from $\{50, 200, 784\}$ (let all hidden layers have the same number of nodes)

2. Programming Problems

Problem 2.1: Decision tree, Bagging, and Random forests

2.1 Overview

In this program, the main target is to visualize each feature's statistics from the **Carseats dataset** and fit the target variable **Sales** using decision tree, bagging, and random forests algorithms imported from sklearn and report their performance respectively.

Carseats

Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
9.5	138	73	11	276	120	Bad	42	17	Yes	Yes
11.22	111	48	16	260	83	Good	65	10	Yes	Yes
10.06	113	35	10	269	80	Medium	59	12	Yes	Yes
7.4	117	100	4	466	97	Medium	55	14	Yes	Yes
4.15	141	64	3	340	128	Bad	38	13	Yes	No
10.81	124	113	13	501	72	Bad	78	16	No	Yes
6.63	115	105	0	45	108	Medium	71	15	Yes	No
11.85	136	81	15	425	120	Good	67	10	Yes	Yes
6.54	132	110	0	108	124	Medium	76	10	No	No
4.69	132	113	0	131	124	Medium	76	17	No	Yes
9.01	121	78	9	150	100	Bad	26	10	No	Yes
11.96	117	94	4	503	94	Good	50	13	Yes	Yes
3.98	122	35	2	393	136	Medium	62	18	Yes	No
10.96	115	28	11	29	86	Good	53	18	Yes	Yes
11.17	107	117	11	148	118	Good	52	18	Yes	Yes
8.71	149	95	5	400	144	Medium	76	18	No	No
7.58	118	32	0	284	110	Good	63	13	Yes	No
12.29	147	74	13	251	131	Good	52	10	Yes	Yes
13.91	110	110	0	408	68	Good	46	17	No	Yes
8.73	129	76	16	58	121	Medium	69	12	Yes	Yes
6.41	125	90	2	367	131	Medium	35	18	Yes	Yes
12.13	134	29	12	239	109	Good	62	18	No	Yes
5.08	128	46	6	497	138	Medium	42	13	Yes	No
5.87	121	31	0	292	109	Medium	79	10	Yes	No
10.14	145	119	16	294	113	Bad	42	12	Yes	Yes

Figure 3: Carseats Dataset Preview

2.1.1 Data Visualization

Before training the dataset, I first visualized the distribution of each feature and their relationship with the target variable Sales using **matplotlib.pyplot package.**

Note that for the textual data in the **ShelveLoc**, **Urban**, and **US**, I omitted their relationship plots with Sales and some descriptive statistics since they are not very intuitive and informative.

The visualization output is given below:

	Sales	CompPrice	Income	Advertising	Population	Price	Age	Education
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	7.496325	124.975000	68.657500	6.635000	264.840000	115.795000	53.322500	13.900000
std	2.824115	15.334512	27.986037	6.650364	147.376436	23.676664	16.200297	2.620528
min	0.000000	77.000000	21.000000	0.000000	10.000000	24.000000	25.000000	10.000000
25%	5.390000	115.000000	42.750000	0.000000	139.000000	100.000000	39.750000	12.000000
50%	7.490000	125.000000	69.000000	5.000000	272.000000	117.000000	54.500000	14.000000
75%	9.320000	135.000000	91.000000	12.000000	398.500000	131.000000	66.000000	16.000000
max	16.270000	175.000000	120.000000	29.000000	509.000000	191.000000	80.000000	18.000000

Figure 4: Statistic Table

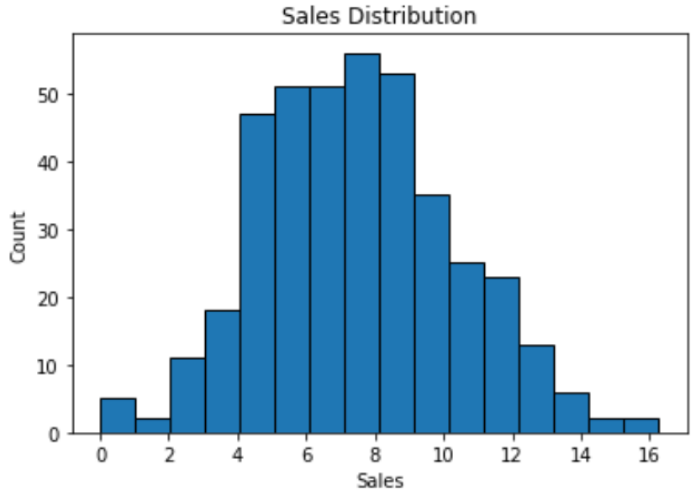


Figure 5

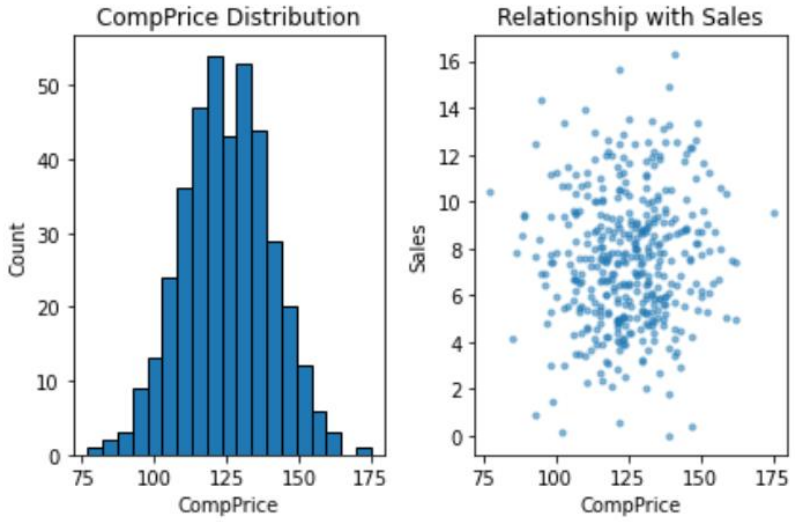


Figure 6

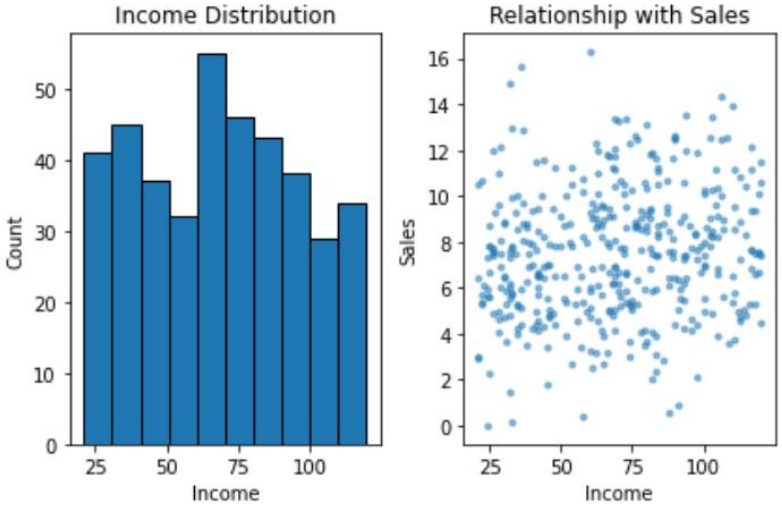


Figure 7

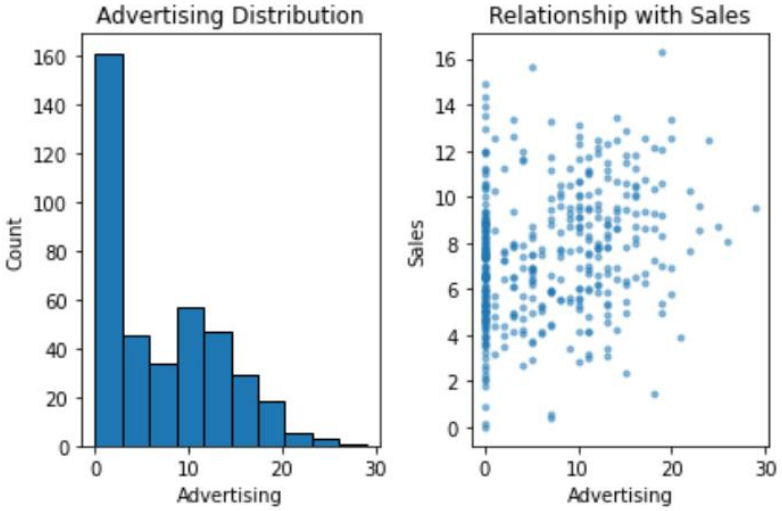


Figure 8

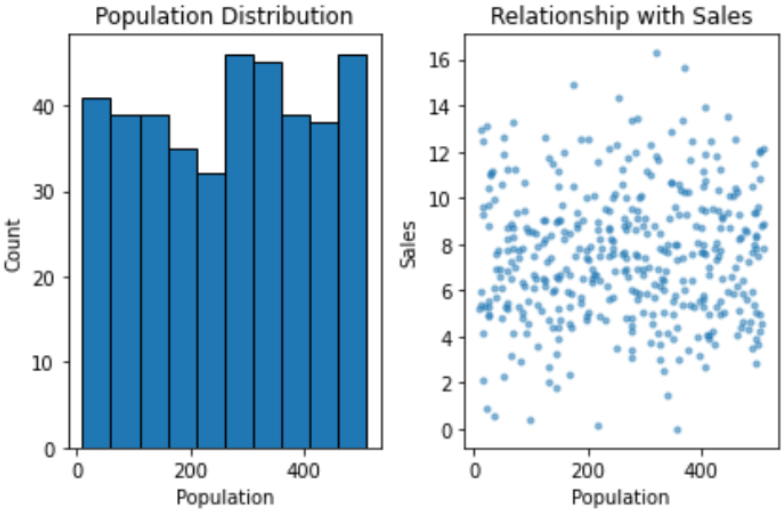


Figure 9

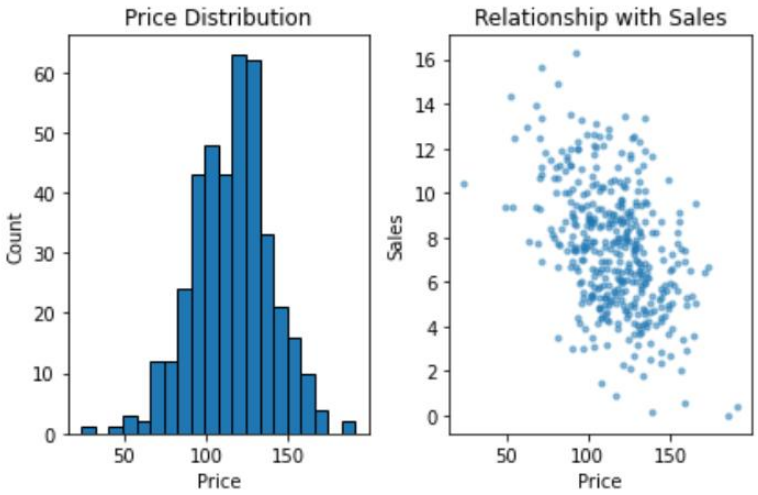


Figure 10

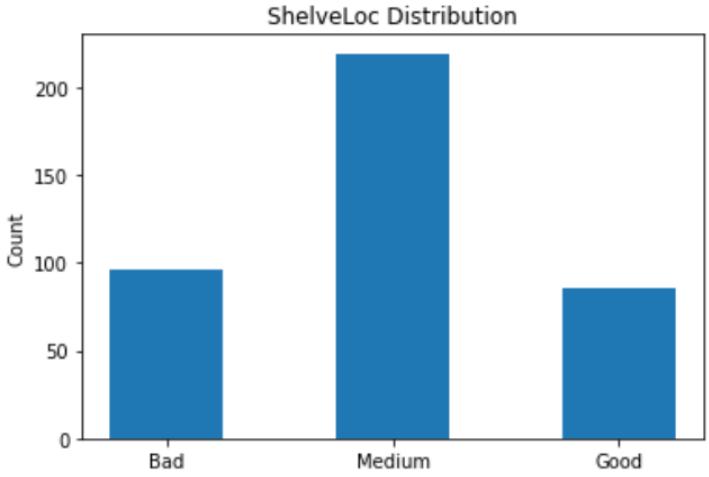


Figure 11

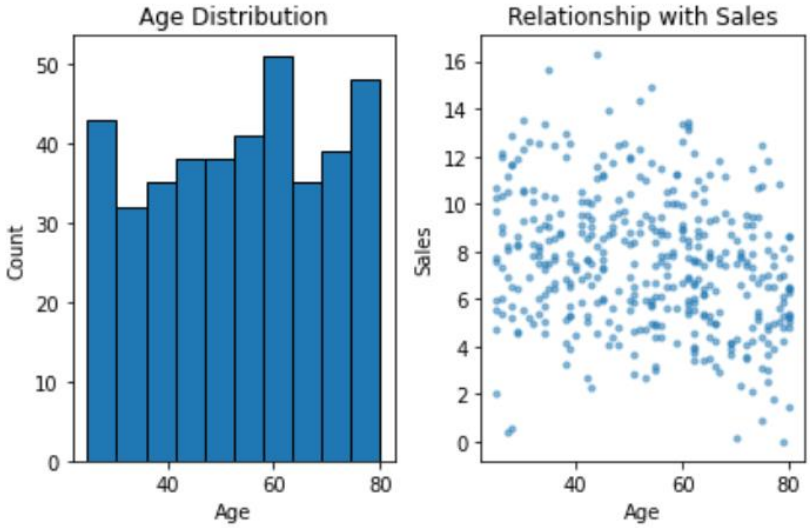


Figure 12

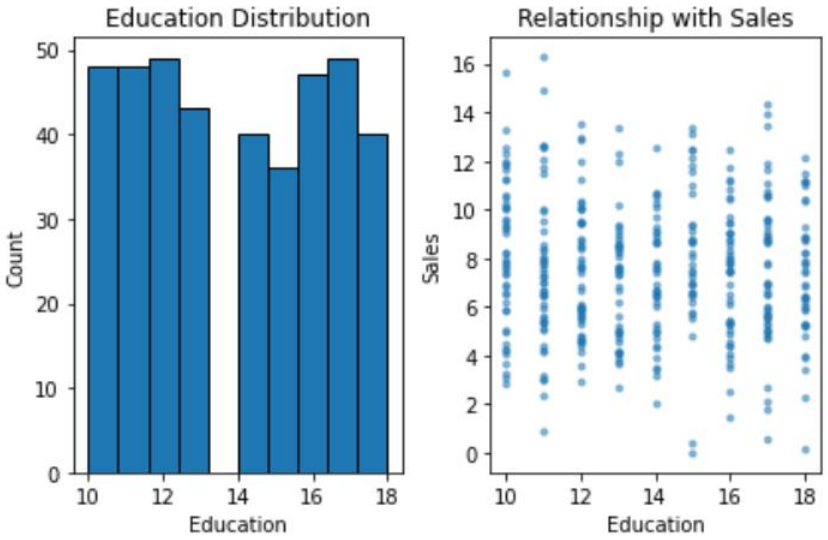


Figure 13

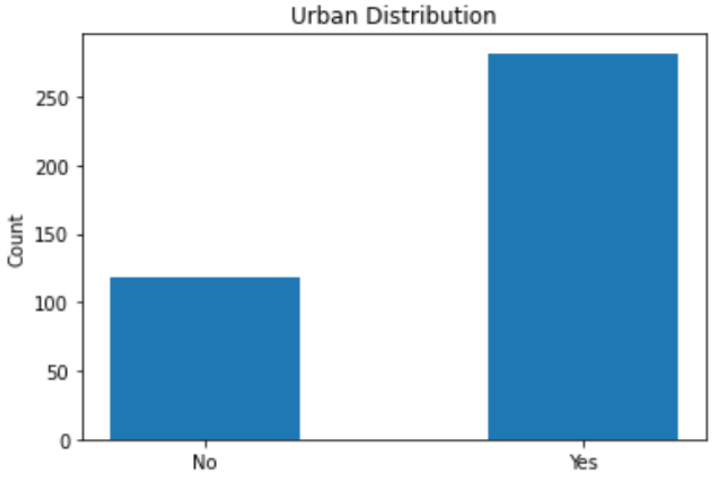


Figure 14

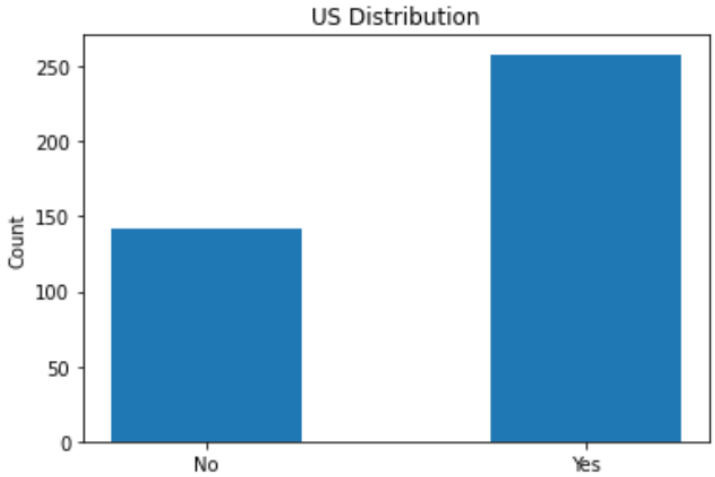


Figure 15

2.1.2 Decision Tree

Mathematical model

Basic Regression-Tree-Growing Algorithm

1. Start with a single node containing all points. Calculate $\bar{y}$ and $\mathcal{S}$.
2. For each node,
   - If all the points in the node have the same value for all the independent variables, **stop**.
   - Otherwise, search over all binary splits of all variables for the one which will reduce $\mathcal{S}$ as much as possible.
     - If the largest decrease in $\mathcal{S}$ would be less than some threshold δ, or one of the resulting nodes would contain less than q points, **stop**.
     - Otherwise, **take that split**, creating two new nodes.
3. In each new node, go back to step 1.

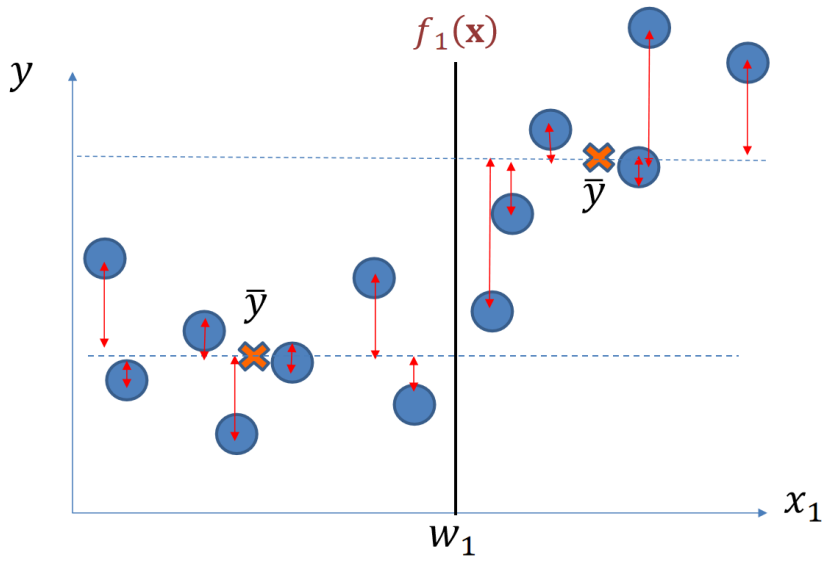


Figure 16: Decision Tree on Regression

The loss is set to Sum of Squared Loss (SSE), given as:

$$\mathrm{L}(\theta) = \sum_{i=1}^{\mathrm{n}} (\hat{y} - y)^2$$

Where $\hat{y}$ is the predicted value and y is the ground-truth value;
And $\theta = \{W_1, W_2 \ldots, W_k\}$

The Mean Squared Error (MSE), given as:

$$Total\ MSE = \frac{\sum_{i=1}^{k_1}(y_i - \bar{y}_{k1})^2}{k_1} + \frac{\sum_{i=1}^{k_2}(y_i - \bar{y}_{k2})^2}{k_2} + \cdots$$

$$where\ k_1 + k_2 + \cdots k_n = n$$

Implementation Details

In this program, the hyper-parameters that need fine-tuning are: **max depth** and **least node size,** which are specified as **max_depth** and **min_samples_leaf** in the sklearn **DecisionTreeRegressor**.
The range for searching of these parameters is set to **[2, 14]** respectively. **The program fixes one parameter while iterating over another.**
To plot the decision tree, the function first utilizes **tree.plot_tree()** from sklearn to plot the tree structure, then use plt.savefig() to save the figure to a png file, and finally display image in the jupyter notebook.
Specifically, the output file names are regularized to "**depthN_sizeM.png**", where N is the max depth input and M is the least node size. **(Specified as 3 and 3 as default)**
Below are some illustrations for the above statements.

```
def decision_tree(depth, nodesize, flag=0):
    clf = DecisionTreeRegressor(max_depth=depth, min_samples_leaf=nodesize)
    clf.fit(X_train, y_train)

    y_tr_predict = clf.predict(X_train)
    train_mse = mean_squared_error(y_train, y_tr_predict)
    train_err = 1- clf.score(X_train, y_train)

    y_ts_predict = clf.predict(X_test)
    test_mse = mean_squared_error(y_test, y_ts_predict)
    test_err = 1- clf.score(X_test, y_test)
```

Figure 17: Function Logic

```
#   plot tree
    elif (flag == 1):
        name = "depth"+str(depth)+"size"+str(nodesize)+'.png'
        plot_tree(clf, feature_names=headers, filled=True, impurity=True)
        plt.savefig(fname=name, dpi=600)
```

Figure 18: Plot Decision Tree

```
# Plot Sample Decision Tree

decision_tree(depth = 3, nodesize = 3, flag=1)
```

Figure 19

```
# Decision Tree

for i in range(2,15):
    decision_tree(depth = 8, nodesize = i)
for i in range(2,15):
    decision_tree(depth = i, nodesize = 10)
```

Figure 20

Outcome Analysis

Given the specific range, the performance evaluations are given below:

```
Max_depth: 8  Least node size: 2
Training MSE:  0.5122717084295334
Testing MSE:  4.759858693745025
Training Error:  0.06419555460655224
Testing Error:  0.6145395199878354

Max_depth: 8  Least node size: 10
Training MSE:  2.0872732398864295
Testing MSE:  4.187501673369895
Training Error:  0.2615675647220644
Testing Error:  0.540643206841981

Max_depth: 5  Least node size: 10
Training MSE:  2.4383359171147565
Testing MSE:  4.374016626981242
Training Error:  0.3055611386311603
Testing Error:  0.5647239238207212

Max_depth: 8  Least node size: 3
Training MSE:  0.7029722129629629
Testing MSE:  4.726847718888889
Training Error:  0.08809327226463526
Testing Error:  0.6102775134982144

Max_depth: 8  Least node size: 11
Training MSE:  2.2755111900865215
Testing MSE:  4.742454019140737
Training Error:  0.2851566863000282
Testing Error:  0.6122924237891775

Max_depth: 6  Least node size: 10
Training MSE:  2.1606720001317394
Testing MSE:  4.207416767190757
Training Error:  0.27076556266699525
Testing Error:  0.5432144201877191

Max_depth: 8  Least node size: 4
Training MSE:  0.9087987042328041
Testing MSE:  4.768562909824263
Training Error:  0.11388650960794977
Testing Error:  0.6156633106537714

Max_depth: 8  Least node size: 12
Training MSE:  2.375550042456374
Testing MSE:  4.532030751004424
Training Error:  0.2976930991145747
Testing Error:  0.585124933635595

Max_depth: 7  Least node size: 10
Training MSE:  2.0872732398864295
Testing MSE:  4.187501673369895
Training Error:  0.26156756472206444
Testing Error:  0.540643206841981

Max_depth: 8  Least node size: 5
Training MSE:  1.0797185793650794
Testing MSE:  4.526661655620432
Training Error:  0.13530540898663446
Testing Error:  0.5844317363135015

Max_depth: 8  Least node size: 13
Training MSE:  2.5598291361551166
Testing MSE:  4.515102611839985
Training Error:  0.3207861148476716
Testing Error:  0.5829393623433059

Max_depth: 8  Least node size: 10
Training MSE:  2.0872732398864295
Testing MSE:  4.187501673369896
Training Error:  0.26156756472206444
Testing Error:  0.5406432068419811

Max_depth: 8  Least node size: 6
Training MSE:  1.3453993395382395
Testing MSE:  4.296434088120781
Training Error:  0.16859931038105924
Testing Error:  0.5547073373507493

Max_depth: 8  Least node size: 14
Training MSE:  2.6724899905749218
Testing MSE:  4.6318354411287315
Training Error:  0.3349042594043943
Testing Error:  0.5980105948091349

Max_depth: 9  Least node size: 10
Training MSE:  2.0872732398864295
Testing MSE:  4.187501673369896
Training Error:  0.26156756472206444
Testing Error:  0.5406432068419811

Max_depth: 8  Least node size: 7
Training MSE:  1.565306288103563
Testing MSE:  3.97536194008092
Training Error:  0.19615704642754905
Testing Error:  0.5132541059770599

Max_depth: 2  Least node size: 10
Training MSE:  4.678050440790932
Testing MSE:  5.227524173325446
Training Error:  0.58623194992489
Testing Error:  0.6749192366617581

Max_depth: 10  Least node size: 10
Training MSE:  2.0872732398864295
Testing MSE:  4.187501673369895
Training Error:  0.26156756472206444
Testing Error:  0.540643206841981

Max_depth: 8  Least node size: 8
Training MSE:  1.773739157010582
Testing MSE:  4.419644944839421
Training Error:  0.2222769031316041
Testing Error:  0.57061493999548

Max_depth: 3  Least node size: 10
Training MSE:  3.670448188301054
Testing MSE:  4.885482983525846
Training Error:  0.4599638301809774
Testing Error:  0.6307587180161653

Max_depth: 11  Least node size: 10
Training MSE:  2.087273239886429
Testing MSE:  4.187501673369895
Training Error:  0.2615675647220643
Testing Error:  0.540643206841981

Max_depth: 8  Least node size: 9
Training MSE:  1.9817439793280793
Testing MSE:  4.211401810102114
Training Error:  0.24834311898889838
Testing Error:  0.543728924192028

Max_depth: 4  Least node size: 10
Training MSE:  2.9309720185530157
Testing MSE:  4.631415260763235
Training Error:  0.3672960485054366
Testing Error:  0.5979563458373198

Max_depth: 12  Least node size: 10
Training MSE:  2.0872732398864295
Testing MSE:  4.187501673369895
Training Error:  0.26156756472206444
Testing Error:  0.540643206841981

Max_depth: 13  Least node size: 10
Training MSE:  2.0872732398864295
Testing MSE:  4.187501673369895
Training Error:  0.26156756472206444
Testing Error:  0.540643206841981

Max_depth: 14  Least node size: 10
Training MSE:  2.0872732398864295
Testing MSE:  4.187501673369895
Training Error:  0.26156756472206444
Testing Error:  0.540643206841981
```

We can conclude that given range **[2,14]**, the best parameters for the **max depth** and **least node size** are approximately **8** and **9** respectively.

Some sample decision tree figures are illustrated below.

- **max depth = 3, least node size = 3 (better view)**

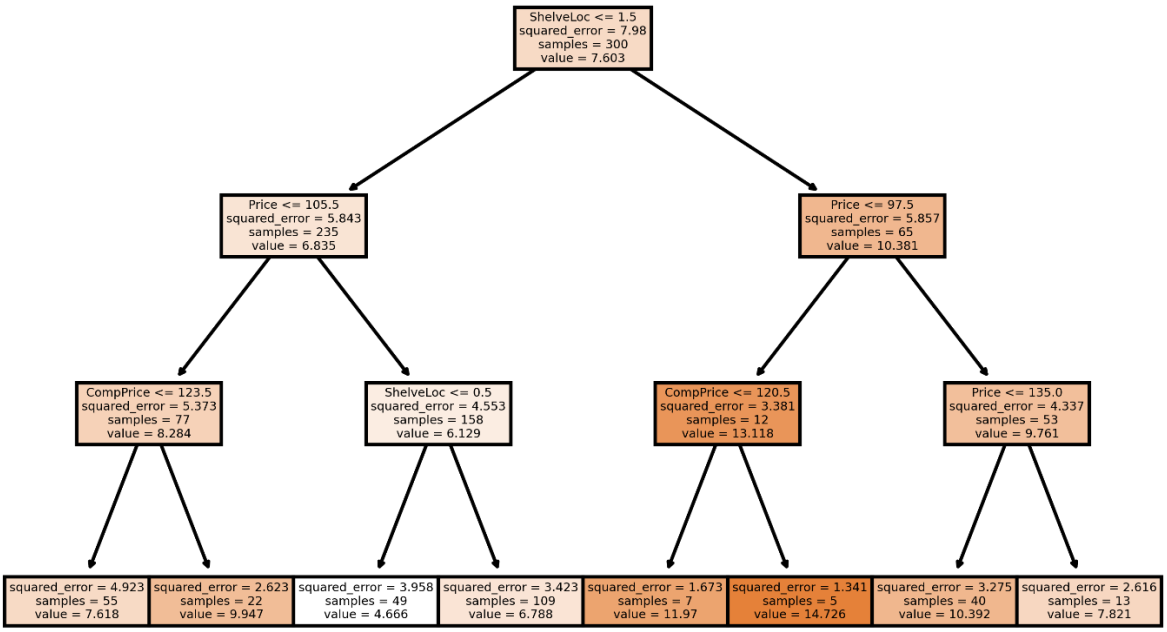


Figure 21: depth3_size3

- **max depth = 8, least node size = 9 (poor view)**

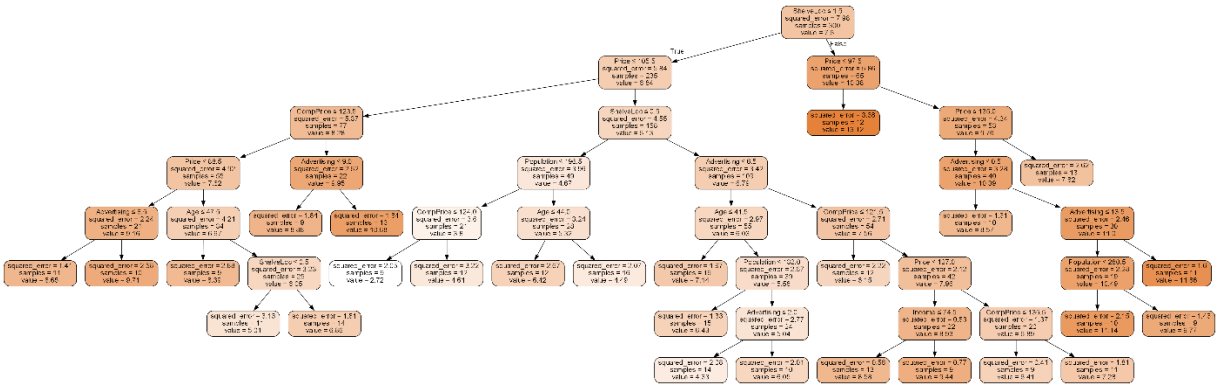


Figure 22: depth8_size9 (partial)

2.1.3 Bagging Trees

Mathematical model

Similar to single decision trees, bagging only adds bootstrap aggregation strategy to increase the data-level randomness and tries to avoid overfitting.
The basic model is illustrated below:

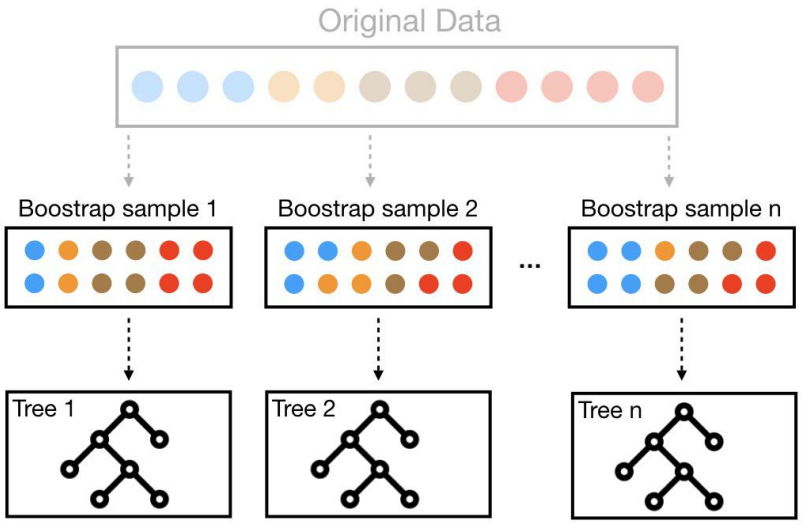


Figure 23

Implementation Details

In this program, the hyper-parameters that need fine-tuning are: **depth** and **number of trees,** which are specified as **max_depth** and **n_estimators** in the sklearn **BaggingRegressor**.

The range for searching of **max_depth** is set to **[2, 14]**, and the range for **n_estimators** is set to **[10, 20, 30, …, 100]**. **The program fixes one parameter while iterating over another.**

```
def bagging_tree(depth, n_trees):
    clf = BaggingRegressor(base_estimator=DecisionTreeRegressor(max_depth=depth), n_estimators=n_trees)
    clf.fit(X_train, y_train.ravel())

    y_tr_predict = clf.predict(X_train)
    train_mse = mean_squared_error(y_train, y_tr_predict)
    train_err = 1- clf.score(X_train, y_train)

    y_ts_predict = clf.predict(X_test)
    test_mse = mean_squared_error(y_test, y_ts_predict)
    test_err = 1- clf.score(X_test, y_test)
```

Figure 24: Function Logic

```
# Bagging Trees

for i in range(10, 101, 10):
    bagging_tree(depth=10, n_trees=i)
for i in range(2, 15):
    bagging_tree(depth=i, n_trees=80)
```

Figure 25: Function invoke

Outcome Analysis

```
Max_depth: 10 Number of trees: 10
Training MSE:  0.6062929090102176
Testing MSE:  2.8591728026560124
Training Error:  0.07597786273860707
Testing Error:  0.3691442949799094

Max_depth: 10 Number of trees: 20
Training MSE:  0.43748198737708266
Testing MSE:  2.852810152970641
Training Error:  0.054823247795874996
Testing Error:  0.3683228210801406

Max_depth: 10 Number of trees: 30
Training MSE:  0.3895574410743809
Testing MSE:  2.8057188431356326
Training Error:  0.04881756218305622
Testing Error:  0.3622429198049969

Max_depth: 10 Number of trees: 40
Training MSE:  0.4116757203881418
Testing MSE:  2.6719134961926945
Training Error:  0.051589324090116184
Testing Error:  0.3449674755170864

Max_depth: 10 Number of trees: 50
Training MSE:  0.4176169569232895
Testing MSE:  2.911183239665395
Training Error:  0.05233385276141789
Testing Error:  0.3758592987332994

Max_depth: 10 Number of trees: 60
Training MSE:  0.37787592583047747
Testing MSE:  2.630656097484267
Training Error:  0.04735368795891404
Testing Error:  0.3396407833546611

Max_depth: 10 Number of trees: 70
Training MSE:  0.3767197501133749
Testing MSE:  2.6962039270574816
Training Error:  0.04720880129006089
Testing Error:  0.348103583264058

Max_depth: 10 Number of trees: 80
Training MSE:  0.37934461774276396
Testing MSE:  2.6392834629082778
Training Error:  0.0475377377323134
Testing Error:  0.3407546519267257

Max_depth: 10 Number of trees: 90
Training MSE:  0.4055603312285529
Testing MSE:  2.736384663576107
Training Error:  0.05082297140603331
Testing Error:  0.35329126889123064

Max_depth: 10 Number of trees: 100
Training MSE:  0.37820954436366
Testing MSE:  2.617851650052265
Training Error:  0.04739549551223432
Testing Error:  0.3379876168459761

Max_depth: 2 Number of trees: 80
Training MSE:  4.1295392837993825
Testing MSE:  4.677169135425668
Training Error:  0.5174950328718223
Testing Error:  0.6038635724971333

Max_depth: 3 Number of trees: 80
Training MSE:  3.044799658287346
Testing MSE:  4.199686527041123
Training Error:  0.381560409277383
Testing Error:  0.5422163783598831

Max_depth: 4 Number of trees: 80
Training MSE:  2.095746877106401
Testing MSE:  3.535464559506045
Training Error:  0.26262944229976326
Testing Error:  0.456459494519867

Max_depth: 5 Number of trees: 80
Training MSE:  1.394413587683952
Testing MSE:  3.0974426801076205
Training Error:  0.17474155245994227
Testing Error:  0.3999070267200536

Max_depth: 6 Number of trees: 80
Training MSE:  0.9523770706995437
Testing MSE:  3.0653033644683
Training Error:  0.11934755178175305
Testing Error:  0.3957575590831929

Max_depth: 7 Number of trees: 80
Training MSE:  0.6513730321712031
Testing MSE:  2.7729794527659872
Training Error:  0.08162709821351366
Testing Error:  0.35801597725544276

Max_depth: 8 Number of trees: 80
Training MSE:  0.4962953562177529
Testing MSE:  2.6650066735694313
Training Error:  0.06219347099750605
Testing Error:  0.3440757441165052

Max_depth: 9 Number of trees: 80
Training MSE:  0.4003698907517013
Testing MSE:  2.7361761676723715
Training Error:  0.05017252907322267
Testing Error:  0.3532643502407318

Max_depth: 10 Number of trees: 80
Training MSE:  0.3695116024013724
Testing MSE:  2.651187024285699
Training Error:  0.04630550908702957
Testing Error:  0.34229150614146087

Max_depth: 11 Number of trees: 80
Training MSE:  0.3542078860761136
Testing MSE:  2.5935829180747034
Training Error:  0.04438771714014822
Testing Error:  0.33485431061572957

Max_depth: 12 Number of trees: 80
Training MSE:  0.3422623082968131
Testing MSE:  2.791086556851564
Training Error:  0.0428907518031616
Testing Error:  0.3603537632631961

Max_depth: 13 Number of trees: 80
Training MSE:  0.383353318819218
Testing MSE:  2.5725427348299963
Training Error:  0.04804008987204755
Testing Error:  0.33213783835392696

Max_depth: 14 Number of trees: 80
Training MSE:  0.3711518135346089
Testing MSE:  2.6564556253006284
Training Error:  0.04651105286709267
Testing Error:  0.34297172875877857
```

Given the searching range above, we can conclude that the **best parameters value** for **depth** and **number of trees** are around **11** and **80** respectively.

2.1.4 Random Forest

Mathematical model

Compared to bagging, in random forest algorithm, each time a split is to be performed, the search for the split attribute is limited to a random subset of **m** of the **N** attributes, which will reduce the correlation among the trees produced by Bagging and increase randomness into both the data-level and attribute level.

The basic model is illustrated below:

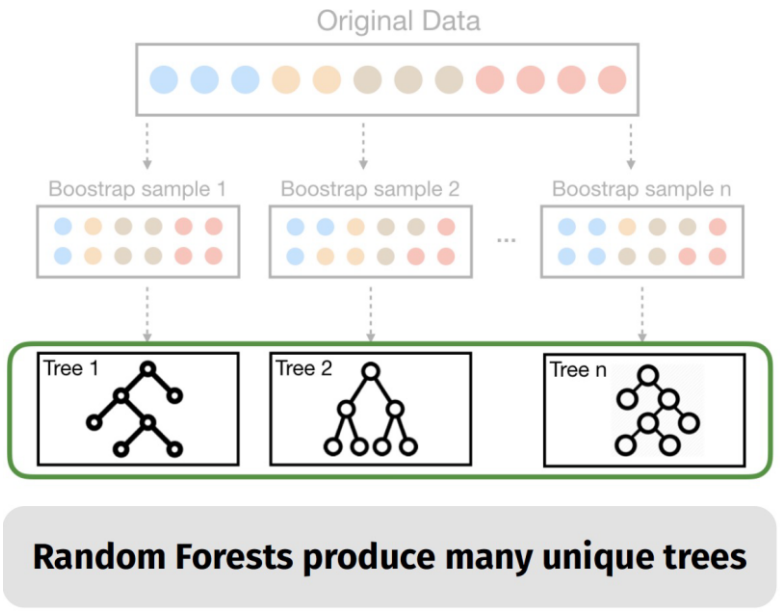


Figure 26

Implementation Details

In this program, the hyper-parameters that need fine-tuning are: **number of trees** and **m (the number of candidate attributes to split in every step),** which are specified as **max_depth** and **max_features** in the sklearn **RandomForestRegressor**.

The range for searching of **m** is set to **[2, 14]**, and the range for **n_estimators** is set to **[10, 20, 30, …, 100]**. **The program fixes one parameter while iterating over another.**

```
def random_forest(n_trees, max_feats="sqrt", flag=0):
    clf = RandomForestRegressor(n_estimators=n_trees, max_features=max_feats)
#     clf.fit(X_train, y_train.ravel())

    if flag == 1:
        loss, bias, var = bias_variance_decomp(clf, X_train, y_train.ravel(), X_test, y_test.ravel(), loss='mse')
        return [bias**2, var]

    clf.fit(X_train, y_train.ravel())
    y_tr_predict = clf.predict(X_train)
    train_mse = mean_squared_error(y_train, y_tr_predict)
    train_err = 1- clf.score(X_train, y_train)

    y_ts_predict = clf.predict(X_test)
    test_mse = mean_squared_error(y_test, y_ts_predict)
    test_err = 1- clf.score(X_test, y_test)
```

Figure 27: Function Logic

Outcome Analysis

```
Number of trees: 10 Max features to split: 8
Training MSE:  0.5611320200000001
Testing MSE:  2.8564846299999993
Training Error:  0.07031850605575674
Testing Error:  0.3687972283042029

Number of trees: 20 Max features to split: 8
Training MSE:  0.3813968133333333
Testing MSE:  2.7564086200000006
Training Error:  0.04779490952597265
Testing Error:  0.3558765373541721 3

Number of trees: 30 Max features to split: 8
Training MSE:  0.35916628000000017
Testing MSE:  3.1172963900000012
Training Error:  0.04500908045704399
Testing Error:  0.4024703148620472

Number of trees: 40 Max features to split: 8
Training MSE:  0.40527686625000003
Testing MSE:  2.6735515699999994
Training Error:  0.05078744886748532
Testing Error:  0.34517896521794

Number of trees: 50 Max features to split: 8
Training MSE:  0.4000415804000001
Testing MSE:  2.6682507520000014
Training Error:  0.0501313867170009
Testing Error:  0.34449458310518055

Number of trees: 60 Max features to split: 8
Training MSE:  0.36760275916666674
Testing MSE:  2.802576594722222
Training Error:  0.046066301556938694
Testing Error:  0.3618372279649855

Number of trees: 70 Max features to split: 8
Training MSE:  0.34596895850340126
Testing MSE:  2.670628718163266
Training Error:  0.04335525230519777
Testing Error:  0.3448015994009459

Number of trees: 80 Max features to split: 8
Training MSE:  0.36480071104166645
Testing MSE:  2.583050605000001
Training Error:  0.045715161662896686
Testing Error:  0.33349449658809993

Number of trees: 90 Max features to split: 8
Training MSE:  0.3651283490123452
Testing MSE:  2.5828742314814814
Training Error:  0.0457562197593947
Testing Error:  0.33347172521937174

Number of trees: 100 Max features to split: 8
Training MSE:  0.3367774862666666
Testing MSE:  2.669678894499999
Training Error:  0.04220341891643453
Testing Error:  0.34467896883234017

Number of trees: 80 Max features to split: 2
Training MSE:  0.4981267446354164
Testing MSE:  3.3822239523437507
Training Error:  0.06242297224310933
Testing Error:  0.436674787614195

Number of trees: 80 Max features to split: 3
Training MSE:  0.3902556831249995
Testing MSE:  3.075340710468750 7
Training Error:  0.04890506269294481
Testing Error:  0.39705346851873435

Number of trees: 80 Max features to split: 4
Training MSE:  0.40301391890624966
Testing MSE:  2.7974701098437493
Training Error:  0.05050386662511852
Testing Error:  0.3611779359632785

Number of trees: 80 Max features to split: 5
Training MSE:  0.35268405703124983
Testing MSE:  2.6799899617187504
Training Error:  0.044196757832712485
Testing Error:  0.3460102180787 7275

Number of trees: 80 Max features to split: 6
Training MSE:  0.34751486557291644
Testing MSE:  2.7778283981250014
Training Error:  0.043548978330009 7
Testing Error:  0.3586420186455561

Number of trees: 80 Max features to split: 7
Training MSE:  0.34376688260416643
Testing MSE:  2.6626381890625015
Training Error:  0.043079298194691695
Testing Error:  0.3437699519857641

Number of trees: 80 Max features to split: 8
Training MSE:  0.3683008572916665
Testing MSE:  2.7549398557812514
Training Error:  0.046169974785589374
Testing Error:  0.3556869070067099

Number of trees: 80 Max features to split: 9
Training MSE:  0.36560594843749983
Testing MSE:  2.657402549687502
Training Error:  0.04581607034156243
Testing Error:  0.34309398500536403

Number of trees: 80 Max features to split: 10
Training MSE:  0.34233023067708324
Testing MSE:  2.692562747187 5004
Training Error:  0.0428992635261396
Testing Error:  0.34763347499541797

Number of trees: 80 Max features to split: 11
Training MSE:  0.3582370507291665
Testing MSE:  2.589792415781251
Training Error:  0.04489263368199037
Testing Error:  0.33436492351207625

Number of trees: 80 Max features to split: 12
Training MSE:  0.34479619333333333
Testing MSE:  2.6711877685937493
Training Error:  0.043208286721744815
Testing Error:  0.3448737777165858

Number of trees: 80 Max features to split: 13
Training MSE:  0.3593845545312497
Testing MSE:  2.68700008140625
Training Error:  0.045036433625996075
Testing Error:  0.346915286036667

Number of trees: 80 Max features to split: 14
Training MSE:  0.35900378036458314
Testing MSE:  2.61211828078125
Training Error:  0.04498871674371119
Testing Error:  0.33724738856895753
```

Given the searching range above, we can conclude that the **best parameters value** for **number of trees** and **m** are around **80** and **8** respectively.

2.1.5 Bias and Variance

Mathematical model

- Ground truth function: $y = f(x)$
- Predicted target value: $\hat{y} = \hat{f}(x) = h(x)$
- Loss function: $S = E[(h(x) - y)^2]$

-	**Squared Loss**
Single loss	$(y - \hat{y})^2$
Expected loss	$E[(y - \hat{y})^2]$
Main prediction $E[\hat{y}]$	mean (average)
Bias^2	$(y - E[\hat{y}])^2$
Variance	$E[(E[\hat{y}] - \hat{y})^2]$

Figure 28

Then the expected test error can be decomposed as:

$$\begin{aligned} &E_{(\mathbf{x},y),D}\left[(h_D(\mathbf{x}) - y)^2\right] = E_{(\mathbf{x},y),D}\left[[h_D(\mathbf{x}) - \bar{h}(\mathbf{x})) + (\bar{h}(\mathbf{x}) - y)]^2\right] \\ =&E_{(\mathbf{x},y),D}\left[(h_D(\mathbf{x}) - \bar{h}(\mathbf{x}))^2\right] + \underbrace{2E_{(\mathbf{x},y),D}\left[(h_D(\mathbf{x}) - \bar{h}(\mathbf{x}))(\bar{h}(\mathbf{x}) - y)\right]}_{=\ 0} \\ &+ E_{(\mathbf{x},y),D}\left[(\bar{h}(\mathbf{x}) - y)^2\right] \end{aligned}$$

Conducting mathematical operation on the highlighting term:

$$\begin{aligned} E[2(y - E[\hat{y}])(E[\hat{y}] - \hat{y})] &= 2E[(y - E[\hat{y}])(E[\hat{y}] - \hat{y})] \\ &= 2(y - E[\hat{y}])E[(E[\hat{y}] - \hat{y})] \\ &= 2(y - E[\hat{y}])(E[E[\hat{y}]] - E[\hat{y}]) \\ &= 2(y - E[\hat{y}])(E[\hat{y}] - E[\hat{y}]) \\ &= 0. \end{aligned}$$

We find that it equals to 0, so finally we have:

$$\begin{aligned} &E_{(\mathbf{x},y),D}\left[(h_D(\mathbf{x}) - y)^2\right] \\ =&E_{(\mathbf{x},y),D}\left[(h_D(\mathbf{x}) - \bar{h}(\mathbf{x}))^2\right] + E_{(\mathbf{x},y)}\left[(\bar{h}(\mathbf{x}) - t(\mathbf{x}))^2\right] + E_{(\mathbf{x},y)}\left[(t(\mathbf{x}) - y)^2\right] \end{aligned}$$

The final expression is the sum of variance, bias^2, and noise.

During the implementation, we often neglect the noise terms, so it becomes:

$$
\begin{aligned}
E[S] &= E[(y - \hat{y})^2] \\
E[(y - \hat{y})^2] &= (y - E[\hat{y}])^2 + E[(E[\hat{y}] - \hat{y})^2] \\
&= [\text{Bias}]^2 + \text{Variance}.
\end{aligned}
$$

Implementation Details

In the program, I implement the decomposition of the bias and variance with the help of mlxtend, with the mathematical theories provided above.

```
def random_forest(n_trees, max_feats, flag=0):
    clf = RandomForestRegressor(n_estimators=n_trees, max_features=max_feats)
    clf.fit(X_train, y_train.ravel())

    if flag == 1:
        loss, bias, var = bias_variance_decomp(clf, X_train, y_train.ravel(), X_test, y_test.ravel(), loss='mse')
        print(var)
        return [bias**2, var]

    y_tr_predict = clf.predict(X_train)
    train_sse = np.sum((y_train - y_tr_predict)**2)
    train_err = 1- clf.score(X_train, y_train)

    y_ts_predict = clf.predict(X_test)
    test_sse = np.sum((y_test - y_ts_predict)**2)
    test_err = 1- clf.score(X_test, y_test)
```

Figure 29: function logic

Note that if we re-scale the bias^2 plot then it will only fluctuate a little.

Outcome Analysis

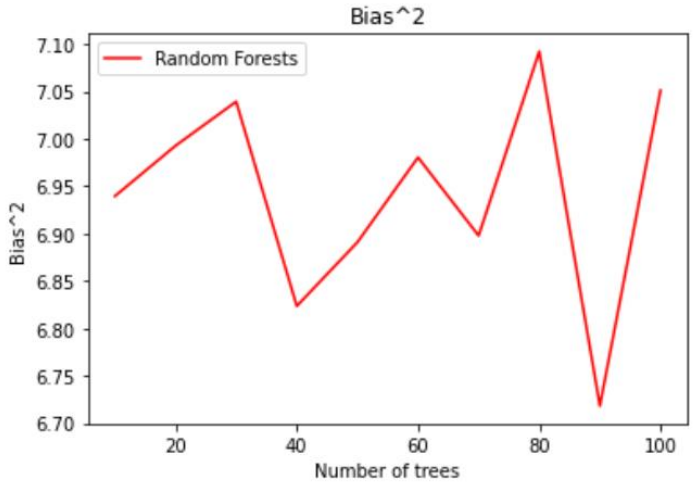


Figure 30

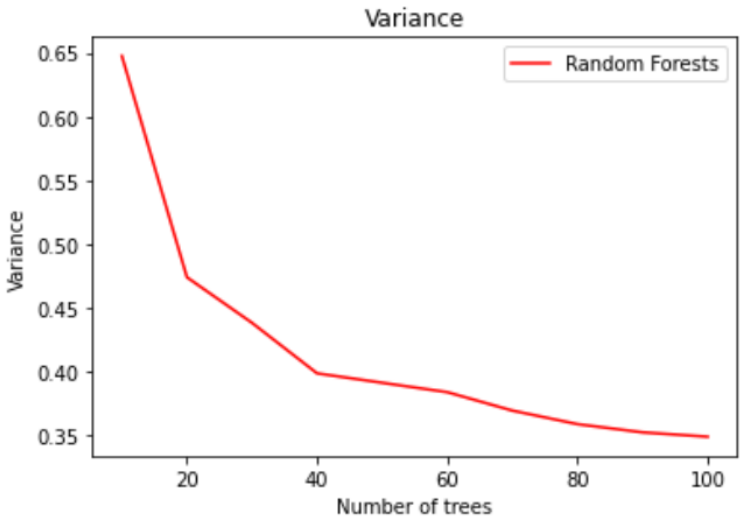


Figure 31

Bias reflects the inherent error that you obtain from your classifier even with infinite training data, it won't have too much change if we fix the model to be random forest regressor.

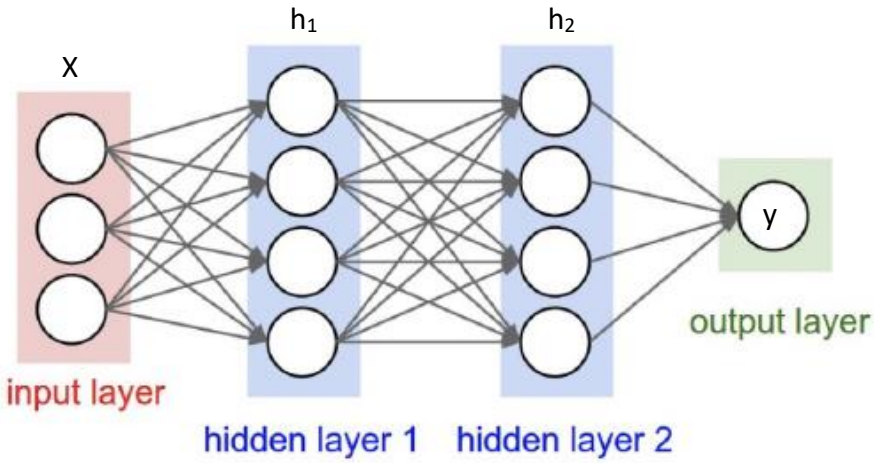
On the other hand, variance captures how much your classifier changes if you train on a different training set, so it is subject to the training set. In this problem, if we increase the number of trees in the random forest, then the average distribution will tend to converge to the expect distribution (with infinite trees), and therefore the variance will decrease.

Problem 2.2: Neural Networks

2.2. Overview

In this program, the main target is to use the function "MLPClassier" of sklearn to construct a **fully connected network** to classify the **MNIST data** provided, and then show the performance of neural network with different structures (i.e., different parameters chosen)

2.1.1 Mathematical Model



$$y = g_1(w^T h_2 + b)$$
$$\mathrm{h}_2 = g_2(\boldsymbol{W_2} h_1 + c)$$
$$\mathrm{h}_1 = g_3(\boldsymbol{W_2} x + d)$$

Where $g(\cdot)$ are the non-linear activation functions

One can use gradient descent with back propagation methods to minimize the loss function and update the parameters.

2.1.2 Implementation Details

In this program, the parameters that need fine-tuning are: **number of hidden layers** and **number of nodes (neurons) within each layer,** which are specified as **hidden_layer_sizes = (# of nodes,) * # of layers** in the sklearn **MLPClassifier**.
The range are given as **[1, 2, 3]** and **[50, 200, 785]** respectively.

Note that I utilized the python script load_mnist.py to load the raw data into np.array
The MNIST raw data are placed under ./MNIST folder

```
X_train, X_test, y_train, y_test = load_mnist(path = './MNIST/', flatten = True, binary_data = False)
```

Figure 32: Load Data

```
def neural_network(layers=1, nodes=100):
    tup = (nodes,) * layers
    clf = MLPClassifier(hidden_layer_sizes=tup).fit(X_train, y_train)
    tr_error = 1 - clf.score(X_train, y_train)
    ts_error = 1 - clf.score(X_test, y_test)
    print("Number of hidden layers:", layers)
    print("Number of hidden neurons per layer:", nodes)
    print("Trainning Error:", tr_error)
    print("Testing Error:", ts_error)
```

Figure 33: Function Logic

Outcome Analysis

```
Number of hidden layers: 1
Number of hidden neurons per layer: 50
Trainning Error: 0.013883333333333359
Testing Error: 0.04900000000000044
Number of hidden layers: 1
Number of hidden neurons per layer: 200
Trainning Error: 0.005083333333333329
Testing Error: 0.02739999999999998
Number of hidden layers: 1
Number of hidden neurons per layer: 785
Trainning Error: 0.0038000000000000256
Testing Error: 0.02400000000000002
Number of hidden layers: 2
Number of hidden neurons per layer: 50
Trainning Error: 0.0038833333333333497
Testing Error: 0.0373
Number of hidden layers: 2
Number of hidden neurons per layer: 200
Trainning Error: 0.0035166666666666124
Testing Error: 0.02529999999999999
Number of hidden layers: 2
Number of hidden neurons per layer: 785
Trainning Error: 0.0030333333333333323
Testing Error: 0.023399999999999976
Number of hidden layers: 3
Number of hidden neurons per layer: 50
Trainning Error: 0.006983333333333341
Testing Error: 0.03700000000000003
Number of hidden layers: 3
Number of hidden neurons per layer: 200
Trainning Error: 0.0027166666666667005
Testing Error: 0.02300000000000002
Number of hidden layers: 3
Number of hidden neurons per layer: 785
Trainning Error: 0.004966666666666675
Testing Error: 0.023900000000000032
```

Figure 34

From the output given above, we can conclude that the NN has the best performance with **number of hidden layers = 3** and **number of hidden neurons in each layer = 200**.