Project: Gradient Boosting from scratch

Goal

Implement gradient boosting model using approximate function
$f(x) = 0.8\cos(3.2\pi x) + 0.64\cos(10.24\pi x) + 0.51\cos(32.77\pi x)$, with samples between [0,2] interval in increments of 0.005 units (401 samples), to an accuracy (mean absolute error) defined as

$$\frac{1}{401}\sum_{i=1}^{401} |f(x_i) - Model(x_i)| \leq 0.05$$

.

Understanding of Decision Trees and Gradient Boosting

Decision trees in one of the most widely used machine learning approach used for predictions of the values. It is used for both regression and classification problems. The algorithm works by splitting the data based on certain conditions i.e. if-then-else rules to come to a decision that is best for the problem. It is a tree-like graph visually upside down with root at the top and leaf nodes at the bottom. It is a recursive process that is applied to each subtree with the new nodes. It is easy to understand and to explore the data but is prone to overfitting.
To solve the given problem, regression tree is used where the dependent variable is continuous in nature and prediction is performed with mean value. To improve the performance and accuracy, ensemble modelling techniques like bagging and boosting is used.

Gradient Boosting

One of the strongest methods to build predictive models with accuracy is gradient boosting. It is a sequential procedure in which each new model that is produced is added to the past model to minimize the error and improve the accuracy and performance concurrently. i.e. each successive tree is grown using the information from the previous trees. This algorithm starts by creating a base tree model and repeatedly grows many trees in a way where new tree is an improvement of the past one. It is represented as below –
Gradient Boosting = Gradient Descent + Boosting
where gradient decent is used as an optimization technique to fit the best model using weak learner models like decision trees. It fits the new tree by modifying the original data when boosting. In principle, we are updating the predictions so that the sum of the residuals is near to 0 (or minimum) and the expected values are near enough to the real values.
One of the disadvantages of this algorithm is that it can lead to over fitting, but it can be handled by mentioning the stooping criteria so that iterations get stopped when the required result is achieved.

Analysis – Implementation using R

Implementation of gradient boosting using a regression tree as basic component and train as many trees as needed until the desired accuracy is reached. Steps to implement the model and predict the accuracy are as follows:

1. Generating the dataset – Created a data frame, with the x values [0,2] increments of 0.005 units and the corresponding y values calculated from the stated function. Plotted the graph of data distribution and the given function.

```
> #Glimpse of the generated dataset values
> head(df)
  x_data   y_data
1  0.000 1.950000
2  0.005 1.874715
3  0.010 1.666170
4  0.015 1.371861
5  0.020 1.057325
6  0.025 0.789414
> dim(df)    #showing 401 samples
[1] 401   2
```

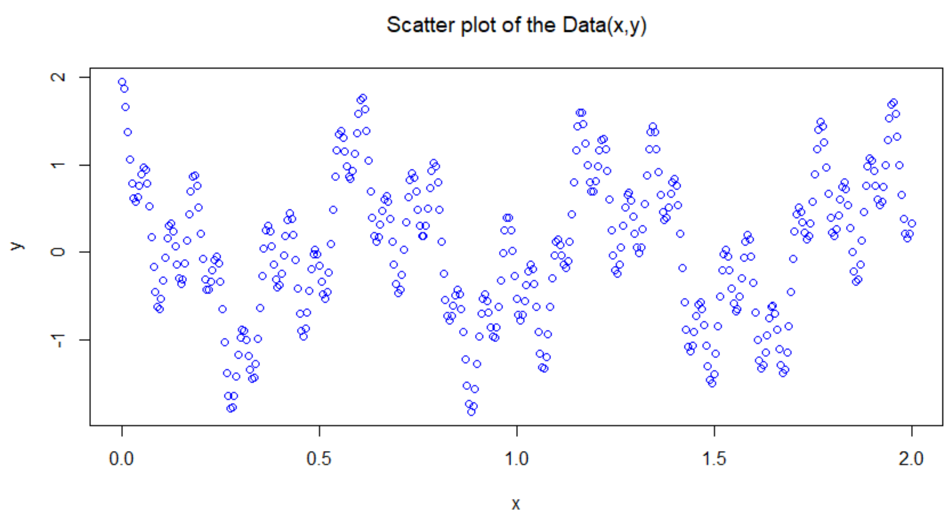

Fig.1 Data Distribution for x and y values

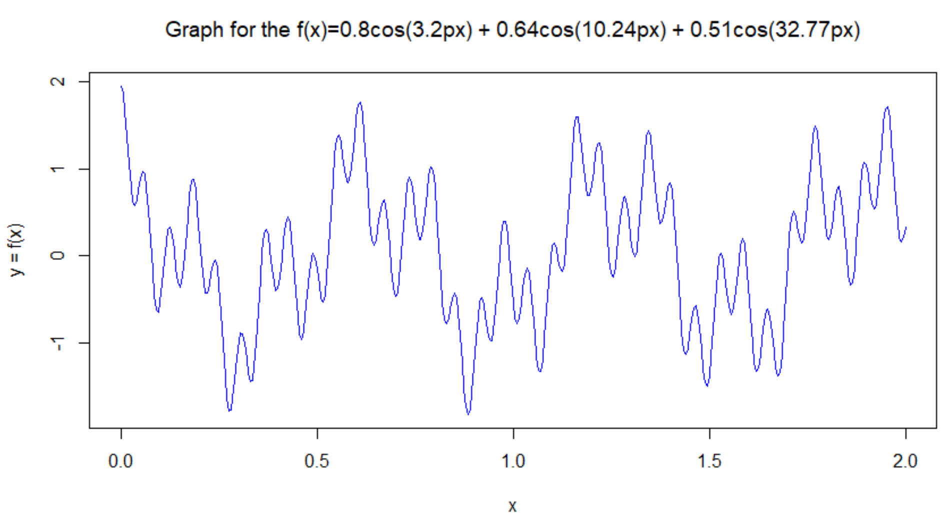

Fig.2 Original function f(x) representation

2. After generating the data, we will create a decision tree model using the y values (dependent variable) with rpart() function and plot the tree graph using rpart.plot().

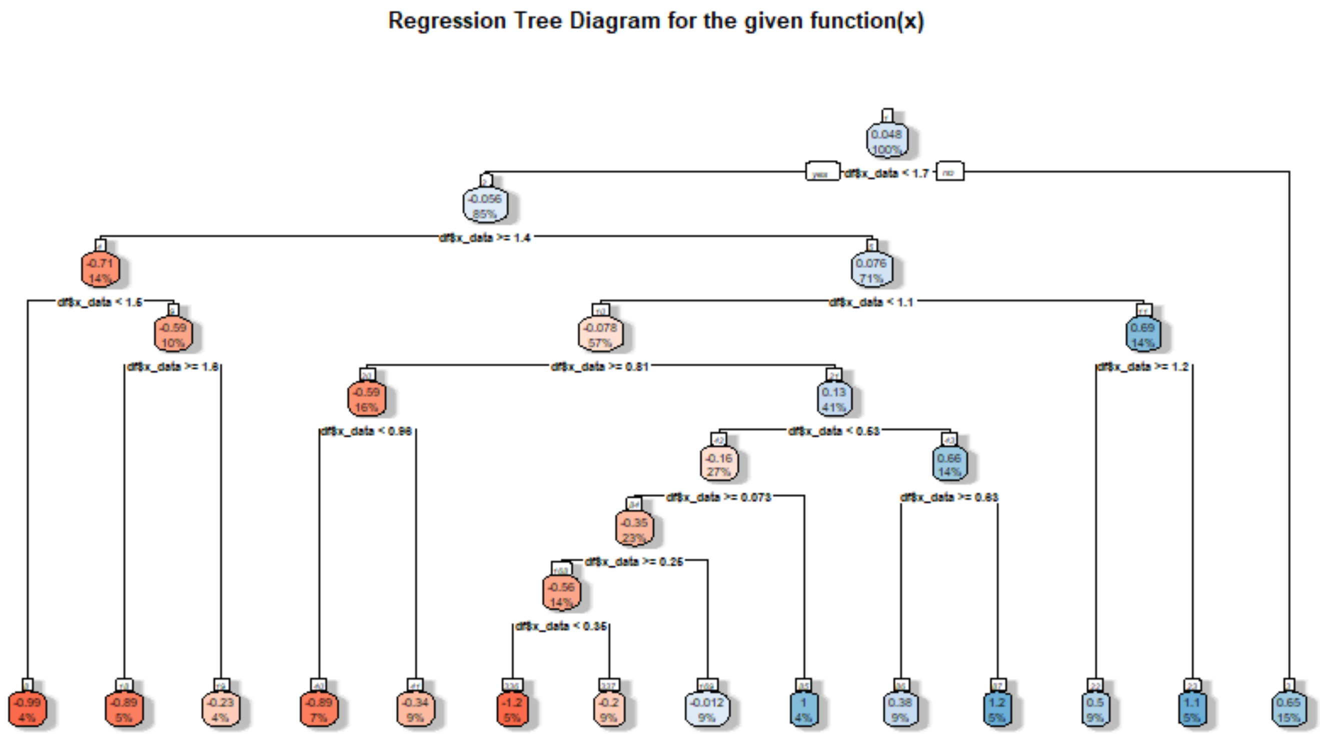

Fig.3 Decision Tree Model for the given y values

Here, we observe that the root node level is at 401. The tree splits based on the given conditions. For ex: Here 1st division is seen when x_data < 1.7. Similarly, the tree structure grows with the levels.

3. Now, we use the predict () function to predict the new values of y_data (actual) and calculate the error residuals using the formula (y_actual – y_predicted).

```
> head(result)
      x        y   y_pred       error
1 0.000 1.950000 1.026015  0.92398530
2 0.005 1.874715 1.026015  0.84870045
3 0.010 1.666170 1.026015  0.64015524
4 0.015 1.371861 1.026015  0.34584643
5 0.020 1.057325 1.026015  0.03131024
6 0.025 0.789414 1.026015 -0.23660071
```

4. Now, we calculate the Mean absolute error as defined in the problem:

```
> mae_acc <- sum(abs(df$y_data - df$y_pred)) # formula defining the accuracy
> message("Mean absolute Error of the Model is : ", mae_acc)
Mean absolute Error of the Model is : 134.474732272317
```

This error (MAE) should be < = 20.05. We can visualize from Fig.4 that after 1st iteration the outcome is not that accurate and needs improvement to fit the given data.

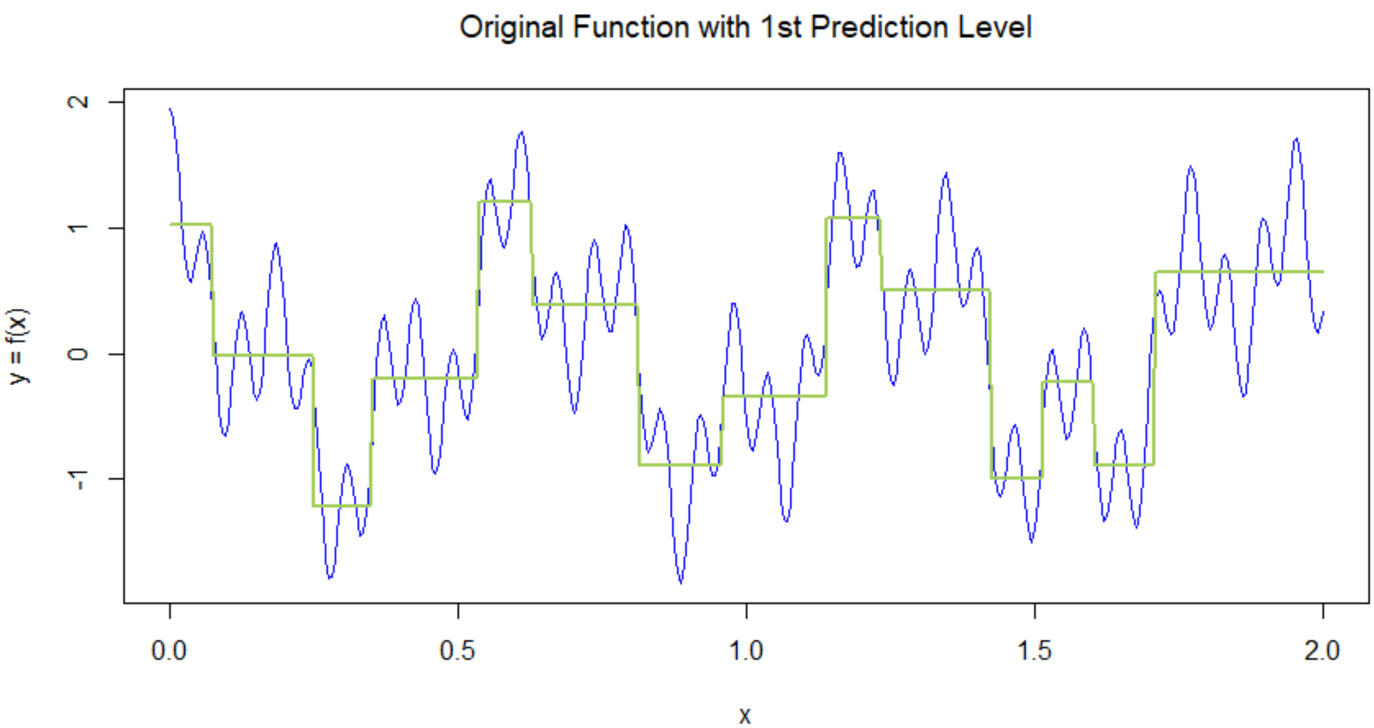

Fig.4 Graphical representation of Original and 1st Prediction value

5. Now, we focus on fit the error data points to get the best fit of the model using the gradient boosting algorithm. So, we will adjust the predicted value of y to reduce the error value and use this error residual to train the next tree model. The adjustment of y_predicted is by adding the product of previous predicted y with the learning rate(alpha) to the new y_predicted. This repeats until the error residual reaches the minima (<=20.05). We use the learning rate alpha (in our case alpha = 0.05) to reduce the error values to some iterations and used cp (complexity_parameter = 0.0000000000000000001) to reach the minimum value of error faster. The lower the value of cp, the faster the minimum error is reached.

6. Running the Step 5 in a loop, gives the error list with the last iteration showing the error value approx. to 20.04. Total tree length is 361.

```
> error_list
  [1] 134.47473 131.09676 128.07067 125.03706 122.44648 119.73577 117.39393 115.11654 113.03779 111.24830 109.52454
 [12] 107.24331 105.26287 103.38536 101.68303 100.23700  98.64901  97.13551  95.79860  94.64177  93.27158  92.10025
 [23]  91.06305  89.83527  88.90843  87.70803  86.77309  85.67342  85.06488  84.23195  83.33310  82.44809  81.58260
 [34]  80.86129  80.01380  79.29546  78.64155  77.76883  77.10639  76.38186  75.51641  74.74526  73.97267  73.24521
 [45]  72.55049  71.79238  71.05044  70.40383  69.77347  69.16992  68.58293  68.02999  67.46029  66.88478  66.37529
 [56]  65.72499  65.18567  64.65276  64.14520  63.70642  63.20974  62.63515  62.16505  61.58295  61.19773  60.75102
 [67]  60.29214  59.87981  59.43085  59.06067  58.67799  58.17651  57.67433  57.29273  56.85964  56.52705  56.07688
 [78]  55.68538  55.31421  54.95183  54.60003  54.13310  53.86462  53.54086  53.17091  52.84583  52.59361  52.18521
 [89]  51.81630  51.43321  51.07647  50.71449  50.48881  50.22895  49.84359  49.54310  49.24815  48.96856  48.63222
[100]  48.32173  48.02497  47.78261  47.49734  47.30470  47.00963  46.75927  46.47631  46.24067  45.89479  45.55840
[111]  45.28069  45.01239  44.79682  44.54766  44.24323  44.03243  43.76630  43.51339  43.22563  42.96738  42.68510
[122]  42.47495  42.26816  42.05043  41.86332  41.61439  41.34829  41.09280  40.87504  40.63811  40.39700  40.18033
[133]  40.01615  39.78861  39.58923  39.37105  39.17649  38.98032  38.79921  38.58965  38.38471  38.22563  38.04872
[144]  37.83150  37.63473  37.46253  37.28990  37.09581  36.95861  36.77715  36.61258  36.44226  36.27200  36.10039
[155]  35.93779  35.75490  35.63556  35.44522  35.26727  35.09944  34.97788  34.82387  34.66895  34.49060  34.34311
[166]  34.18968  34.05158  33.92616  33.74764  33.59690  33.42284  33.26966  33.15890  33.00207  32.89293  32.75822
[177]  32.62643  32.48245  32.33000  32.18030  32.05790  31.93115  31.76858  31.63376  31.50646  31.35374  31.24869
[188]  31.11247  31.01497  30.91268  30.77492  30.67711  30.55969  30.46098  30.36085  30.24218  30.13224  30.02904
[199]  29.94644  29.85455  29.76163  29.67228  29.52271  29.42489  29.34815  29.23127  29.13233  29.02419  28.93570
[210]  28.86670  28.76084  28.66999  28.54753  28.47030  28.37416  28.28432  28.21460  28.14585  28.06288  27.94861
[221]  27.87921  27.77379  27.68224  27.61361  27.52278  27.42852  27.36209  27.25311  27.16556  27.09263  26.98464
[232]  26.90008  26.82206  26.74444  26.65161  26.57410  26.49204  26.42298  26.36276  26.29029  26.21862  26.10503
[243]  26.03403  25.96584  25.88464  25.82640  25.76533  25.69526  25.62574  25.53537  25.46639  25.43115  25.36569
[254]  25.27786  25.20342  25.12640  25.03722  24.97634  24.90736  24.85884  24.76799  24.70798  24.63826  24.56658
[265]  24.50175  24.44338  24.40021  24.33964  24.26799  24.18942  24.15081  24.10726  24.02358  23.98067  23.91801
[276]  23.86569  23.79483  23.71303  23.66435  23.59132  23.56189  23.52856  23.47019  23.43178  23.35422  23.29303
[287]  23.23844  23.17959  23.13353  23.06559  23.00823  22.97517  22.92204  22.85765  22.81865  22.74429  22.71632
[298]  22.66485  22.58603  22.54932  22.50797  22.47745  22.41442  22.38859  22.34342  22.32858  22.25535  22.21208
[309]  22.14611  22.07853  22.05075  21.99857  21.93691  21.91994  21.85301  21.82047  21.77055  21.73102  21.70687
[320]  21.67319  21.64471  21.59288  21.54777  21.48707  21.44900  21.39428  21.36022  21.31438  21.25879  21.23154
[331]  21.20442  21.14253  21.09447  21.05807  21.03292  20.98713  20.92570  20.89812  20.87498  20.82478  20.80917
[342]  20.75192  20.69044  20.67222  20.64074  20.61006  20.55416  20.54414  20.47831  20.42731  20.38652  20.34179
[353]  20.32127  20.26750  20.23637  20.22075  20.16713  20.14027  20.08795  20.07311  20.04041
> length(error_list)
[1] 361
```

Fig.5 Error Value list for the model

7. Now, we validate the implementation visually in Fig.6. The blue line shows the original(actual) function, green line shows the 1st iteration of prediction with MAE of 134.47. the red line shows how the marking of predictions is getting closer to the actual value (blue line) as the error is getting minimized closed to 20.05.

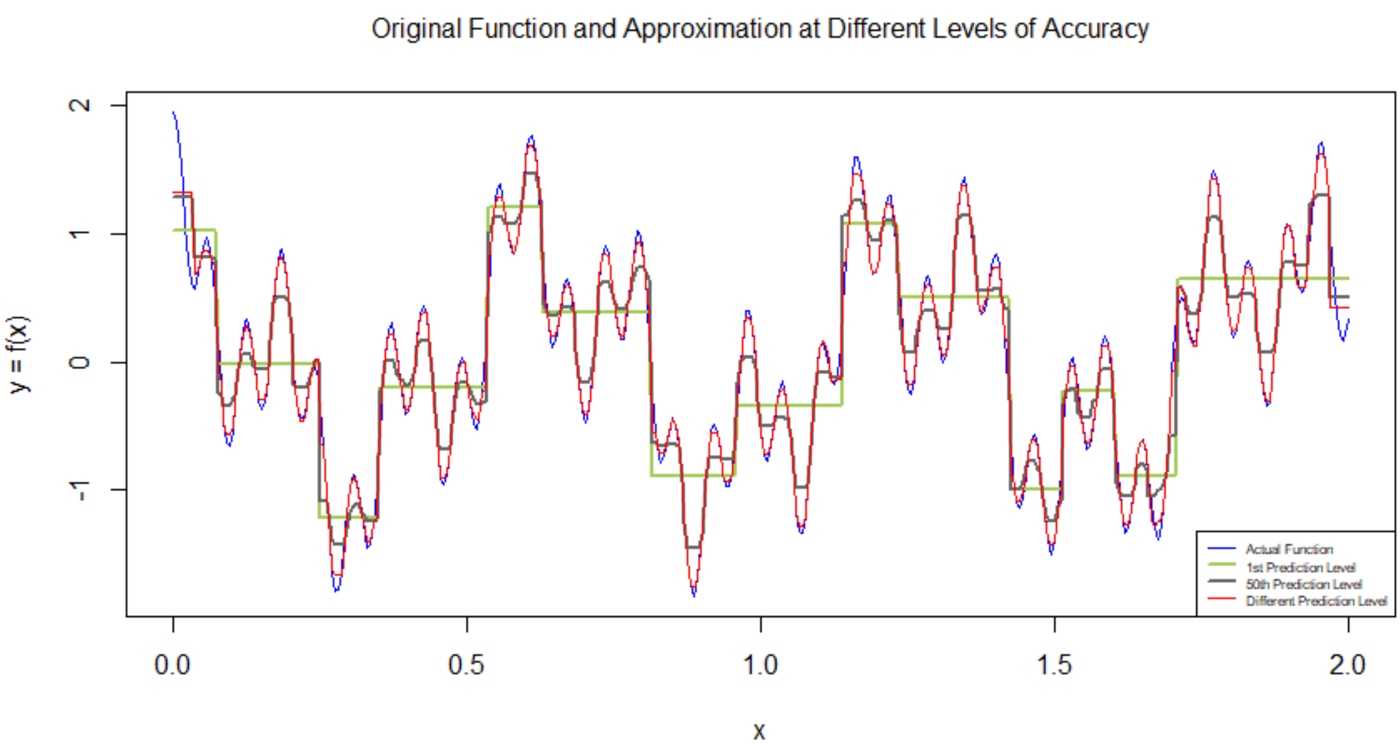

Fig.6 Gradient Boosting for the predicted values

8. The error progression after each tree trained is shown in the below report. Total 360 trees are created.

Data_Samples	Error after 1 tree	Error after 2 tree	Error after 3 tree	Error after 4 tree	Error after 5 tree	Error after 6 tree	Error after 7 tree	Error after 8 tree	Error after 9 tree	Error after 10 tree	Error after 11 tree
1	0.923985301	0.908651695	0.894084769	0.880246189	0.8670995388	0.854610221	0.8427453687	0.831473759	0.820765730	0.810593103	0.800929107
2	0.848700447	0.833366841	0.818799915	0.804961335	0.7918146850	0.779325367	0.7674605149	0.756188905	0.745480876	0.735308249	0.725644253
3	0.640155240	0.624821634	0.610254708	0.596416128	0.5832694779	0.570780160	0.5589153079	0.547643698	0.536935669	0.526763042	0.517099046
4	0.345846427	0.330512821	0.315945895	0.302107315	0.2889606648	0.276471347	0.2646064947	0.253334885	0.242626856	0.232454229	0.222790233
5	0.031310236	0.015976630	0.001409704	-0.012428876	-0.0255755262	-0.038064844	-0.0499296962	-0.061201306	-0.071909335	-0.082081962	-0.09174595
6	-0.236600709	-0.251934315	-0.266501241	-0.280339820	-0.2934864708	-0.305975789	-0.3178406408	-0.329112250	-0.339820279	-0.349992907	-0.35965690
7	-0.406692091	-0.422025697	-0.436592623	-0.450431202	-0.4635778527	-0.476067171	-0.4879320228	-0.499203632	-0.509911661	-0.520084289	-0.52974828
8	-0.456267794	-0.440718529	-0.425946729	-0.411913518	-0.3985819675	-0.385916995	-0.3738852705	-0.362794986	-0.352259215	-0.341910380	-0.33207898
9	-0.396521150	-0.380971886	-0.366200085	-0.352166874	-0.3388353236	-0.326170351	-0.3141386267	-0.303048342	-0.292512571	-0.282163736	-0.27233234
10	-0.269094038	-0.253544774	-0.238772973	-0.224739762	-0.2114082117	-0.198743239	-0.1867115148	-0.175621230	-0.165085460	-0.154736624	-0.14490523

Fig.7 Error Progression Report for the given model Snapshot

error_progression_report_0.05.csv

Conclusion

Gradient boosting with regression trees enhanced the model's precision by decreasing the error value by close to 20.05. The training model was initially weak, but the function showed improvement in the marking on recursive iterations (trees) and adjustments of the predicted y values using learning rates and stopping criteria as shown by red line in Fig. 6 which moves closer and becomes relatively strong to the actual function (blue line) demonstrating high accuracy and low error rate. This overlap on the real data function indicates more precise predictions are being made for the model. This way predicted values can be trained using regression tree via gradient boosting.

References

Hastie, Tibshirani & Friedman. (2008). The Elements of Statistical learning. Stanford: Springer.

Brownlee, J. (2016). A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/

Prince G. (2017). Gradient Boosting from scratch. Medium. Retrieved from https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d