ASSIGNMENT 3: GRADIENT BOOSTING

ABHISHEK BOSE

Instructor: Marco Montes De Oca

June 14, 2019

Abstract

The aim of this assignment is to implement gradient boosting from scratch. We have been given a approximate function for which we need to perform gradient boosting.

Contents

Introduction

Gradient Boosting is a type of ensemble learning. The main idea behind ensemble learning is rather than relying on a single model to determine the best approximation we take the ouput from multiple model to reach the conclusion. Some of the technique used in ensemble learning:
1. **Bagging**: In order to increase the model variance, this technique trains each model randomly by taking a subset from the training set.
2. **Boosting**: It involves training a new model multiple times incrementally by taking the inputs from previous models to decrease the residual. One of the major drawback of boosting is it sometimes over-fit the training data.

In our assignment we will discuss the *Gradient Boosting Algorithm* to illustrate how we can inrease the performance. Normally we would like to decrease the bias, variance and noise which cause the difference between actual and predict also known as *residuals*. Gradient Boosting help us to reduce bias and variance except noise, which is irreducible error. We can represent

$$\sum_{i=1}^{m}(y-\hat{y})^2 = noise^2 + bias^2 + variance$$

Since each predictors are learning from mistakes committed by previous predictors, it takes less time/iterations to reach close to actual predictions depending on the learning rate. But we must have a clear idea about where to stop or it could lead to overfitting of data.

Algorithm

Input: training set a differentiable loss function for number of iterations M.

$$Let\ the\ training\ set\ be\ :\{(x_i, y_i)\}_{i=1}^{n}\ and\ the\ loss\ function\ be\ L(y, f(x)).$$

1. **Initialize model with a constant value taking any model:** *Here in this assignment we have considered Decision tree*
$$F_0(x) = argmin_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$$
2. **For m = 1 to M:**
    1. **Compute so-called pseudo-residuals:** *This is done by substracting oveserved value and predicted value we obtain from step 1.*
    2. **Fit a base learner (e.g. tree) to the pseudo-residuals we have calculated:** *In this assignment we will be adding a decision tree with low complexity parameter since we want our residual to be low as 10 also we will be altering our minimum split to 2 and 3 that will impact our iteration i.e. low minsplit can bring down the number of iteration with an increase in variance and the depth in tree will grow immensely since variance is not a scope here we can focus on performance keeping our iteration low.*
    3. **Compute multiplier by solving the following one-dimensional optimization problem:** (*Predict the new residual from the model we built in the previous step.*
    4. **Update the model:***Add the predicted value and the new residuals with a learning rate and update the predicted value. This loop continues till our desired residual is reached.*
3. **Output:** *Once we achieved our target residual we can use this model to do further prediction on test data.*

Analysis:

In this assignment we have been give a function that we need to approximate i.e

$$f(x) = sin(x) + 5cos(2x) - 3sin(3x) + (1 - e^{-x/3}) + 25$$

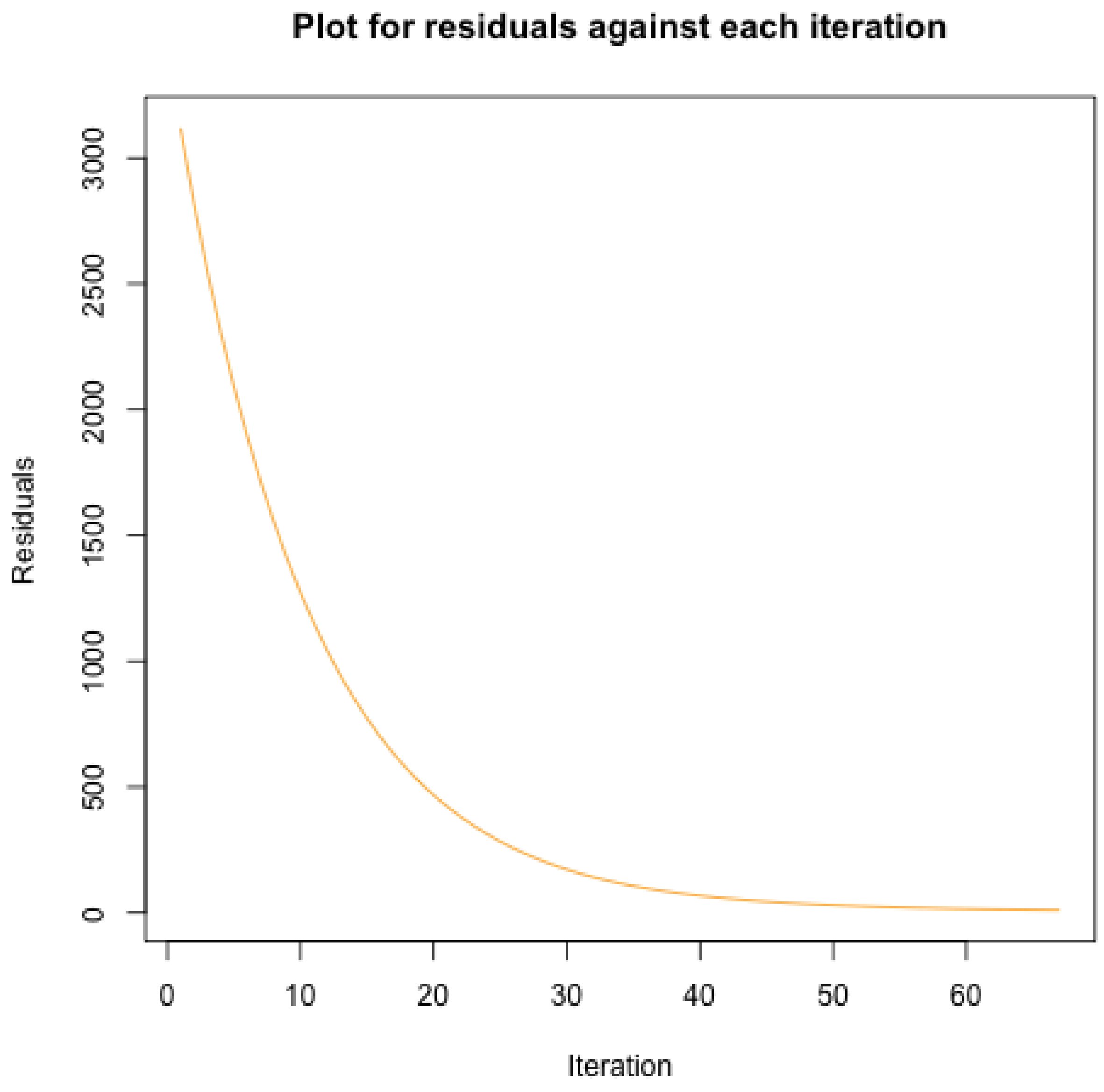
We have approximated the x value as:

```
x<-seq(-10, 10, by=0.01)
```

and its respective y value or f(x) as:

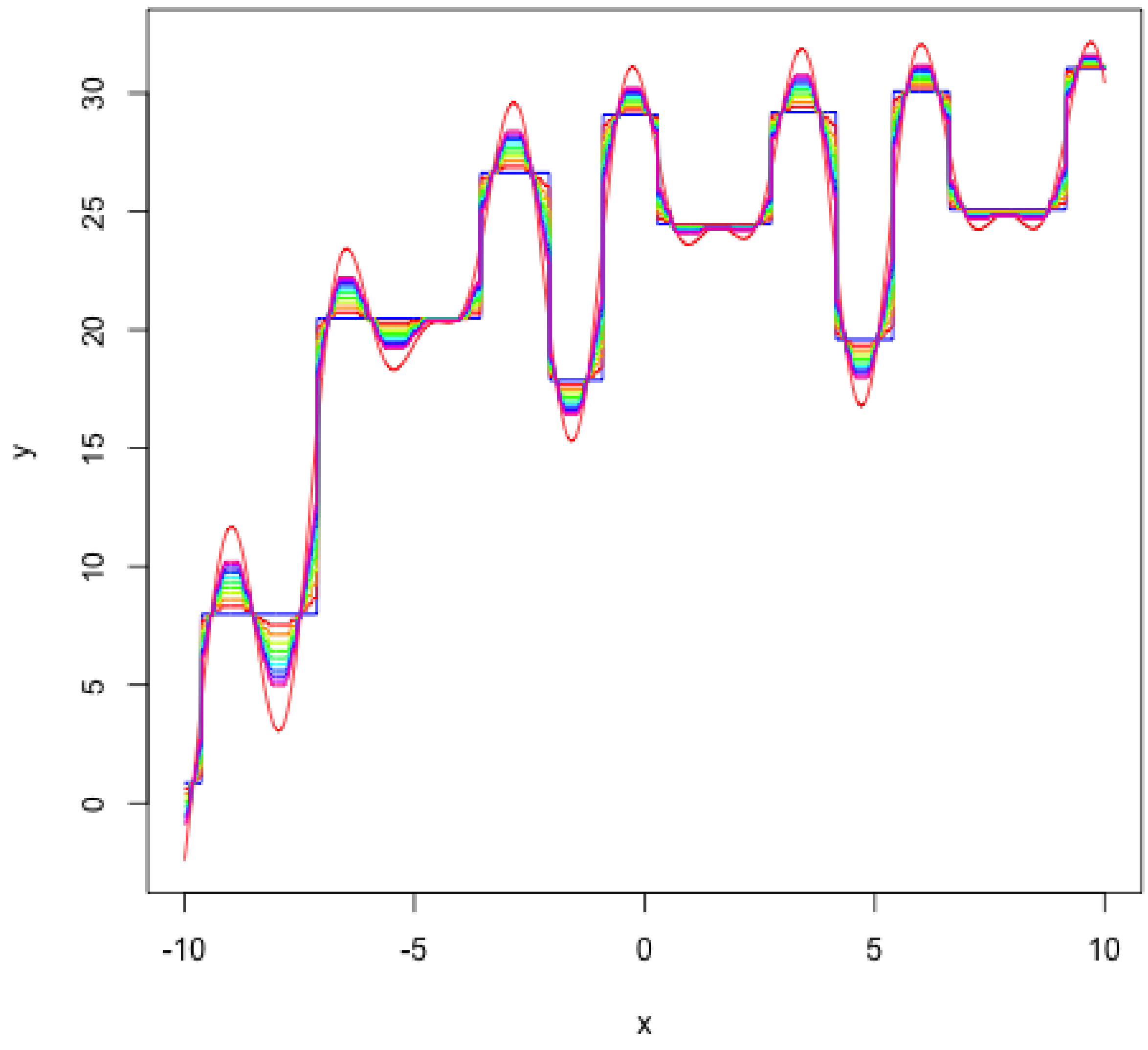
```
y<- sin(x) + 5*cos(2*x) - 3*sin(3*x) + (1-exp(-x/3)) + 25
```

We can observe that the sum of the absolute value of residual start from a larger value close to 3000 and gets reduce to 10 after which we stop the iteration. If we plot the magnitude of such residual against each iteration we will get a plot which look similar to the one mentioned below:



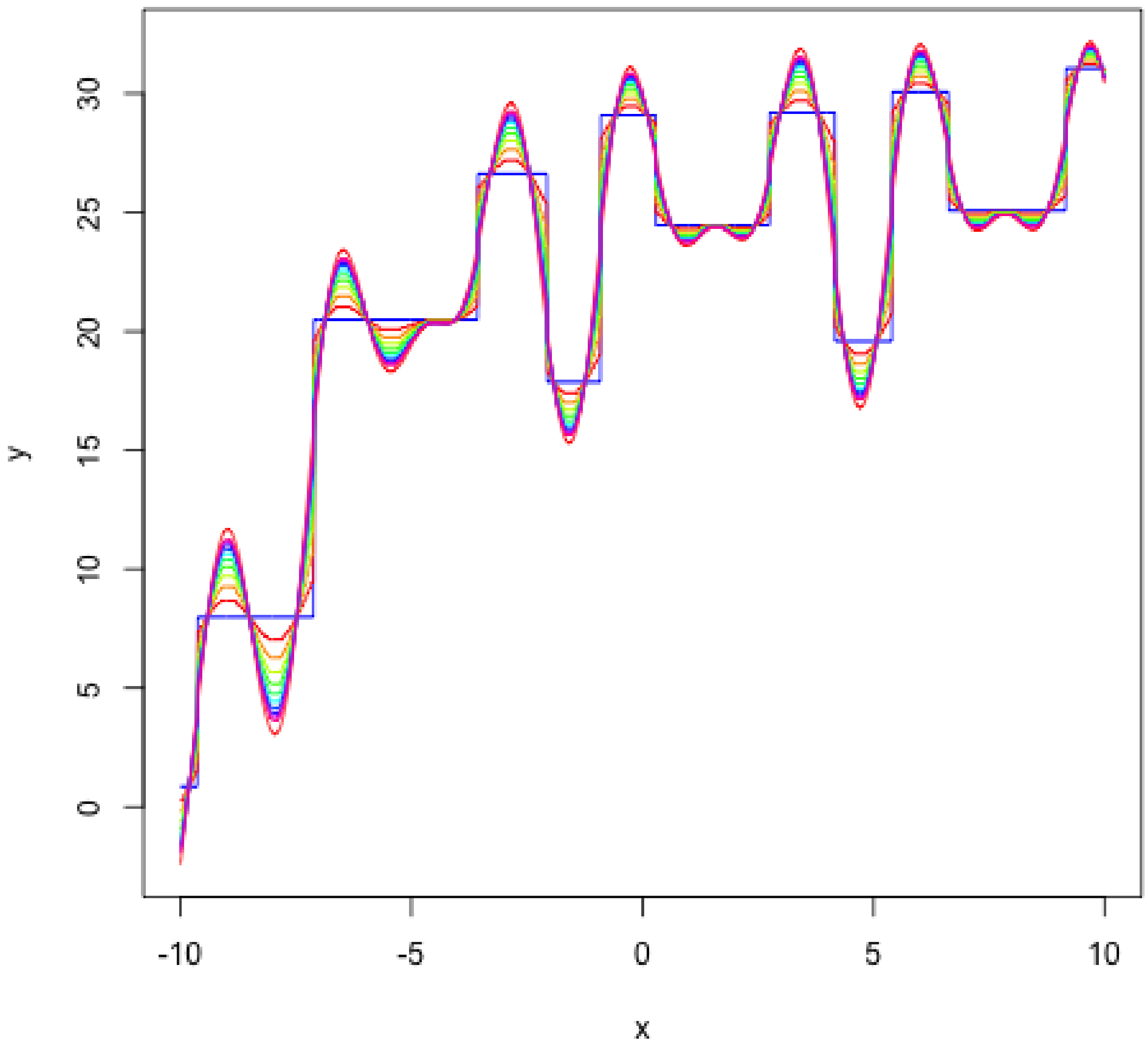
Setting the learning rate at 0.1, minsplit = 2 and complexity parameter = 0.001 we get the following plot:

- In the plot below the red line represent the plot for our function that we approximate.
- The blue line represent our initital prediction or model initialization with a constant value.
- At learning 0.1 we can see that from our initial predicticton model we are progressing at a very slow rate i.e. 0.1 towards actual plot of approximation function.
- It stop at a point where our sum of residual is less than 10 but the end point is not co insiding with our actual plot.



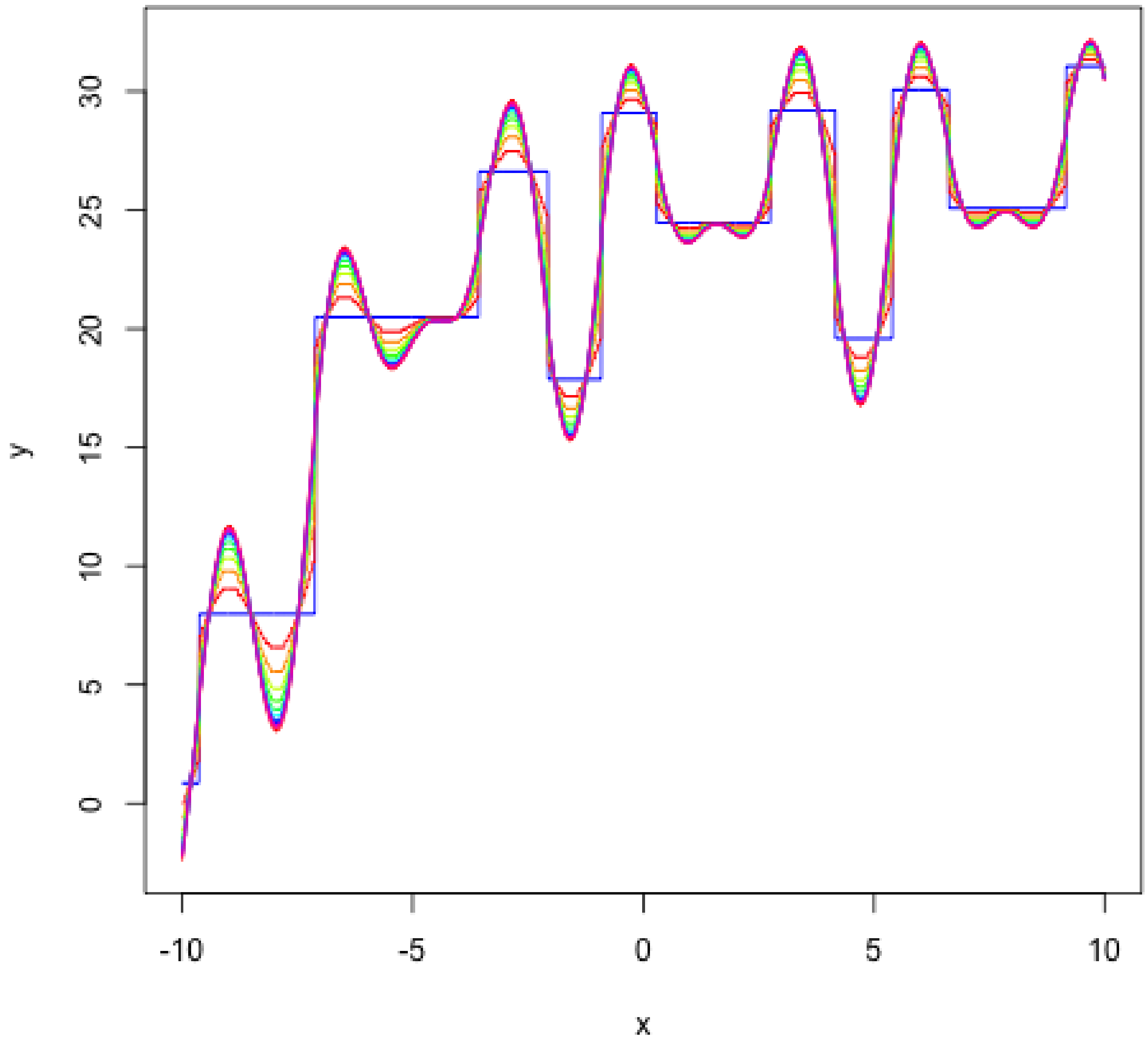
Setting the learning rate at 0.2, minsplit = 3 and complexity parameter = 0.0001 we get the following plot:

- In the plot below the red line represent the plot for our function that we approximate.
- The blue line represent our initital prediction or model initialization with a constant value.
- At learning 0.2 we can see that from our initial predicticton model we are progressing at a rate towards actual plot of approximation function which is faster compare to 0.1 learing rate.
- It stop at a point where our sum of residual is less than 10 but the end point is closer to our actual plot compare to the previous plot.



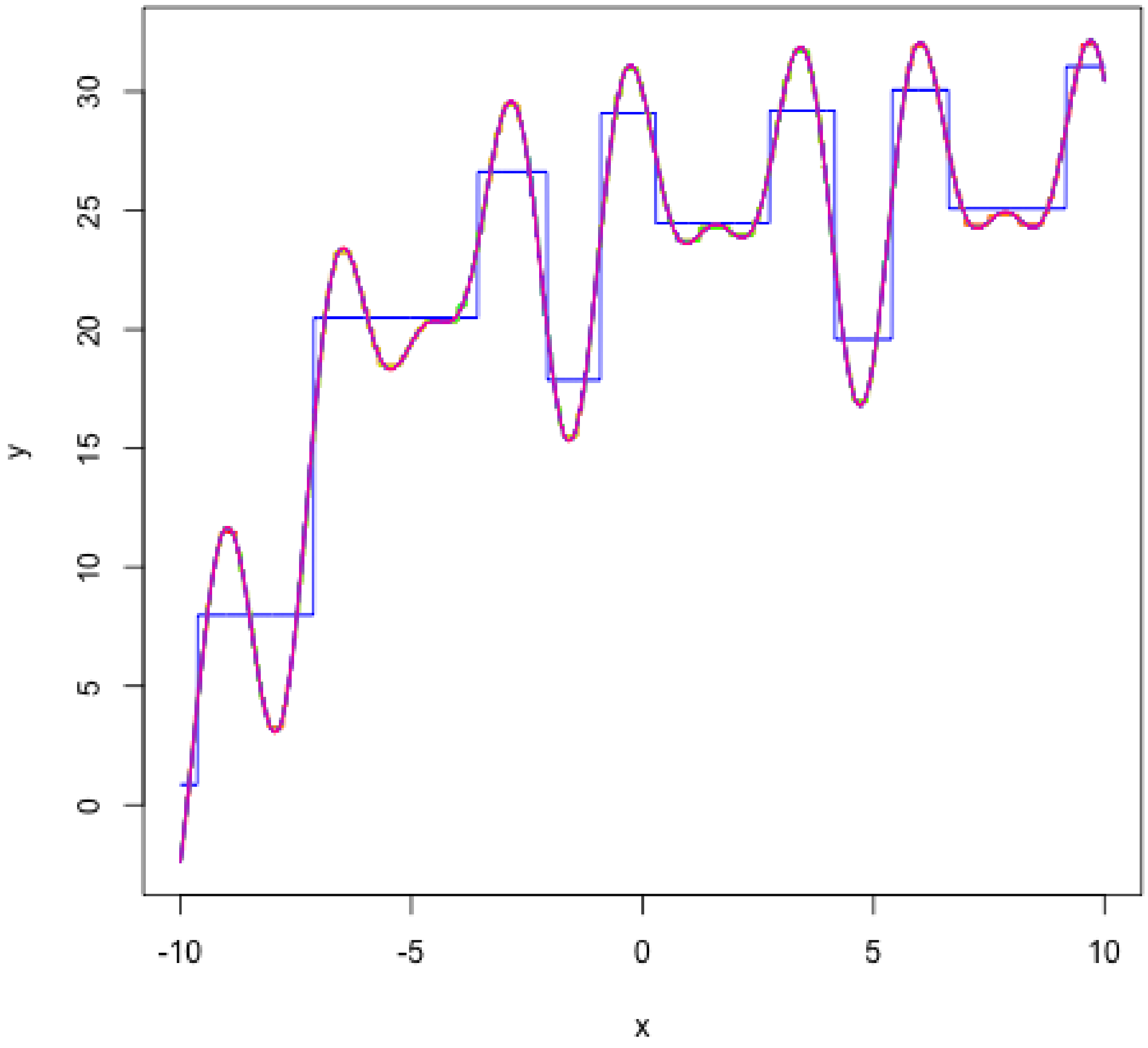
Setting the learning rate at 0.3, minsplit = 3 and complexity parameter = 0.0001 we get the following plot:

- In the plot below the red line represent the plot for our function that we approximate.
- The blue line represent our initital prediction or model initialization with a constant value.
- At learning 0.3 we can see that from our initial predicticton model we are progressing at a rate towards actual plot of approximation function which is faster compare to 0.2 learing rate.
- It stop at a point where our sum of residual is less than 10 but the end point is closer to our actual plot compare to the previous plot.



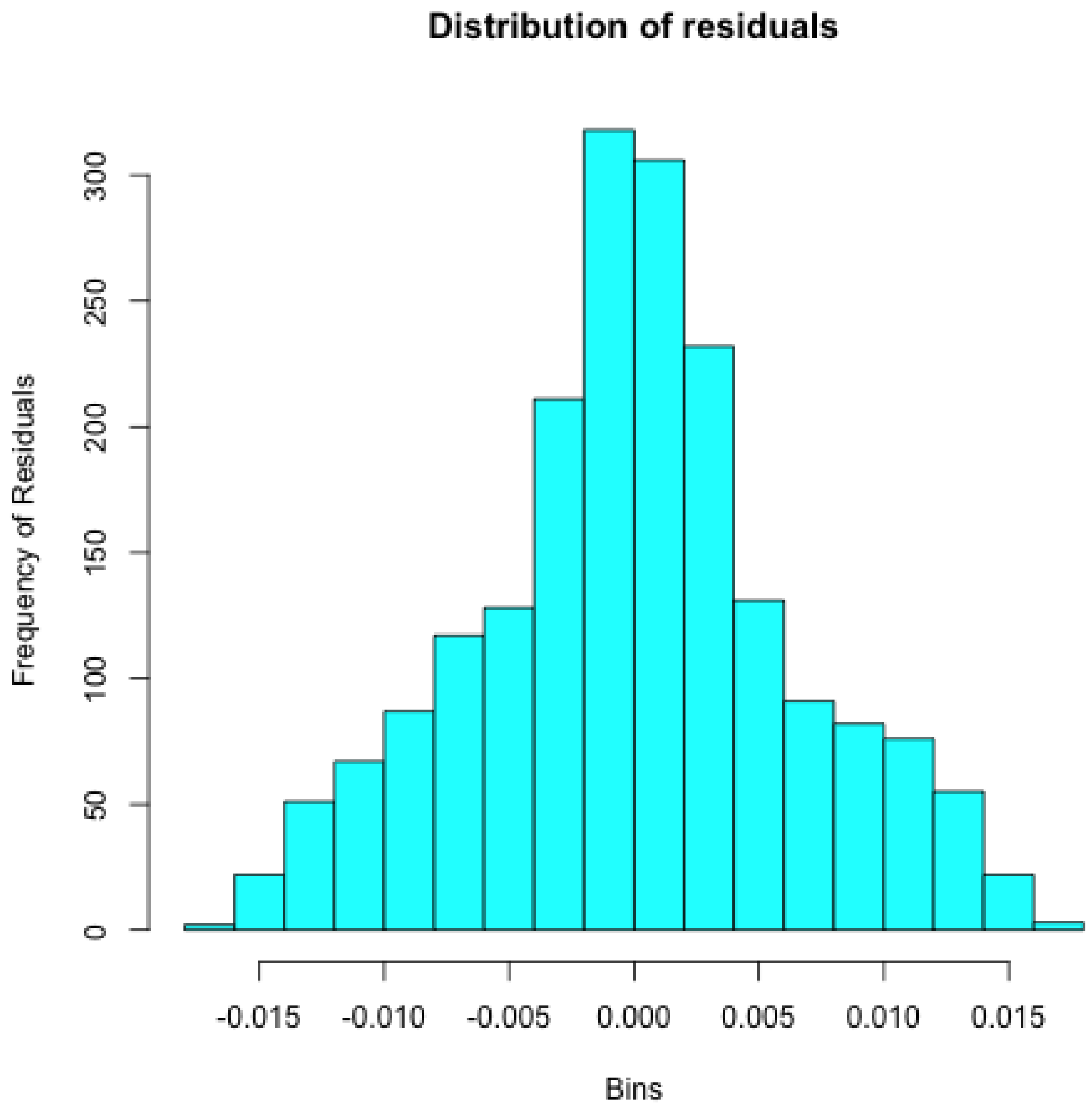
Setting the learning rate at 1, minsplit = 2 and complexity parameter = 0.0001 we get the following plot:

- In the plot below the red line represent the plot for our function that we approximate.
- The blue line represent our initital prediction or model initialization with a constant value.
- At no learning rate we can see that it start from a point which is more or less closer to the approximation function.
- This means without setting a learning rate it is getting overfitted and take the non-reducecable error along with prediction.



Conclusion:

Let's the distribution of our residuals at learning rate at 0.1, minsplit = 2 and cp = 0.001.



The most common problem for a gradient boosting algorithm is the problem of overfitting *(Although we are not discussing the problem of overfitting in our assignment since we have ignored the high variance in our residuals)* we need to be careful about the algorithm so that it doesnot get over fitted. We can apply the following rules to make sure that it doesnot get overfitted.

- As we have observed that using of a shrinkage factor or learning rate, which cab be applied to the contribution of each base learner.
- We have observed that (when working with decision trees), the maximum tree split also plays a huge role in determining the fit. As the tree depth increases it results in increase in variance and there are chances that it might results in overfitting.