Random forest with gradient boosting

Deepesh Kataria

June 6th 2019

we've learned about the regression trees using the samples of random forest. There are multiple advantages and disadvantages associated with the use of regression trees as like other models in the machine learning. The laws or models are not universal hence we choose the suitable technique depending upon the output and the objective of the project and resources provided.

Trees are robust and give better accuracy as compare to other model on the expense of computation power and transparency – [1].

Trees nodes split on the conditions like if statement 1 then this node otherwise other node. That is split function for the tree. More the depth of the tree more will be the accuracy, but we've kept check of overfitting of the model.

We've studied in week 1 about the gradient boosting that it minimizes the cost function. We'll assemble the trees and optimize the results by reducing the loss function using the gradient boosting. Boosting is based on the idea that weak learners can learn better. The weakness is the overfitting, so we need to define a stopping point for the model.

We've used rpart library of the R language to implement the model. We'll

start by defining the function and generate the values of x

```
x1 = sort(seq(-10,10,by=.01)) y1 =sin(x1) + 5*cos(2*x1) -
3*sin(3*x1) + (1-exp(-x1/3)) +25 df1=data.frame(x1,y1)
```

The above code will give us the values of x in the range of -10 to 10 by the step of .01 giving us around 2001 observations. We've our Y function which is objective variable and we've created a data frame by combining both.

```
alpha1=0.1
fit1=rpart(y1~x1,data=df1,maxdepth = 4)
yp1=predict(fit1)
df1$yr1=df1$y1 - alpha1*yp1
YP1=alpha1*yp1
```

In the above code we've set our model with the max depth of the tree as 4 and learning rate which is alpha as 0.1. we'll predict the values of Y1 using the values of x. yr1 is the loss that we've to minimize using the gradient descent. We'll store the product values of learning rate and current point in YP1.

```
#loop for 200 iterations
for(t1 in 1:200){
  fit1=rpart(yr1~x1,data=df1,maxdepth = 4)
yp1=predict(fit1,newdata=df1)   df1$yr1=df1$yr1
- alpha1*yp1
  YP1=cbind(YP1,alpha1*yp1)}
```

nd1=data.frame(x1=seq(-10,10,by=.01))

After that we'll use for loop to find the next step in the model. Here we've defined another model which will predict the loss of the y function using the x values and we'll move toward the local minima of the loss function using gradient descent. we'll create a data frame of the product of learning rate and the current location using the cbind function. *#function for iteration alphaiz=function(M){ y=apply(YP[,1:M],1,sum) y1=apply(YP1[,1:M],1,sum) plot(dfx,dfy,type='l',ylab="",xlab="",lwd=1) lines(df$x,y,type="s",col="red",lwd=1) lines(df1$x1,y1,type="s",col="green",lwd=1) fit1=rpart(y1~x1,data=df1,maxdepth = 4) yp1=predict(fit1,newdata=nd1) lines(nd1$x1,yp1,type="s",col="purple",lwd=1)*

fit=rpart(y~x,data=df)

yp=predict(fit,newdata=nd)

lines(nd$x,yp,type="s",col="blue",lwd=1)

lines(nd$x,sin(nd$x),lty=2)

y2=apply(YP2[,1:M],1,sum)

lines(df2$x2,y2,type="s",col="orange",lwd=1)

fit2=rpart(y2~x2,data=df2)

yp2=predict(fit2,newdata=nd2)

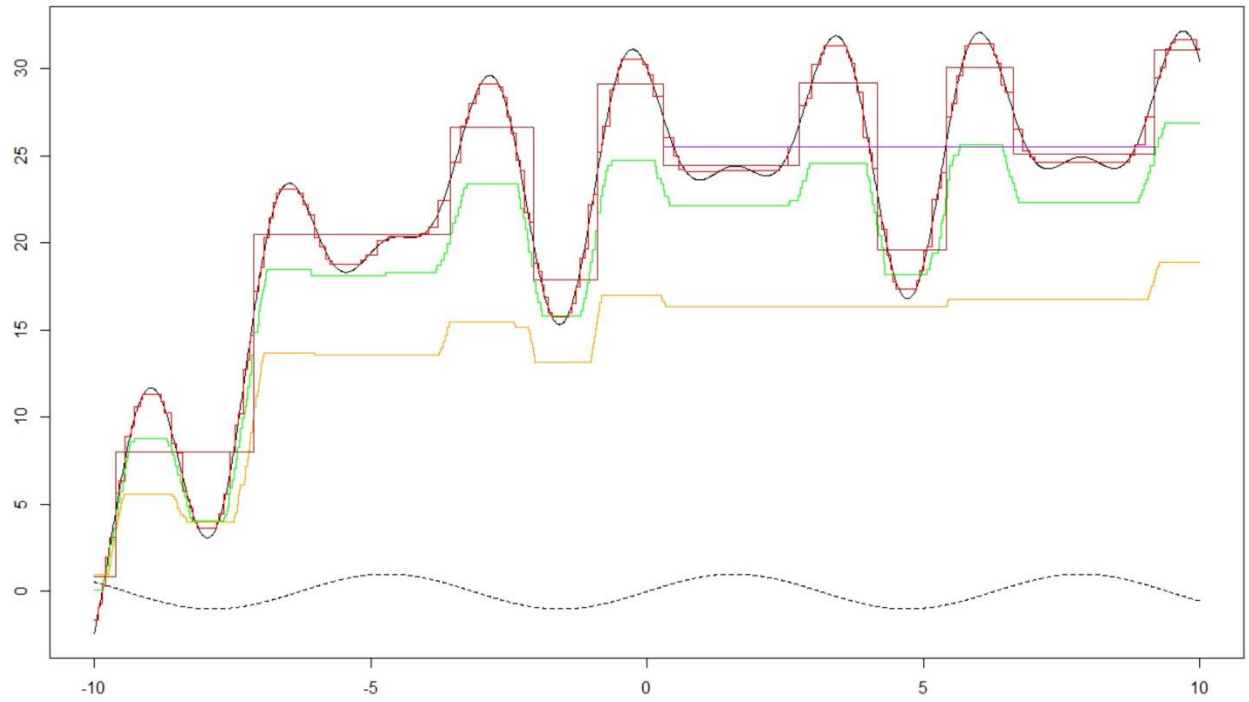
lines(nd2$x2,yp2,type="s",col="brown",lwd=1)} *# ----------[2]*

We've defined a function which will give us the plot of the accuracy for different parameters like learning rate and depth of the tree and we've used apply function to get the sum of YP data frame and using x values to predict. Then plot the predicted values and check the accuracy of the predicted values. All work done just need to pass the number of iterations in the function. The value in the function cannot extend the value of the for loop.

```
alphaiz(200)
YP_ = data.frame(lapply(YP,abs))
YP_ = sum(YP_[,201])
YP_A1 = data.frame(lapply(YP1,abs))
YP_A1 = sum(YP_A1[,201])
YP_A2 = data.frame(lapply(YP2,abs))
YP_A2 = sum(YP_A2[,201])
YP_
YP_A1
YP_A2
```

This will provide us the plot and the accuracies and then we can compare the values.

For Iterations = 20.

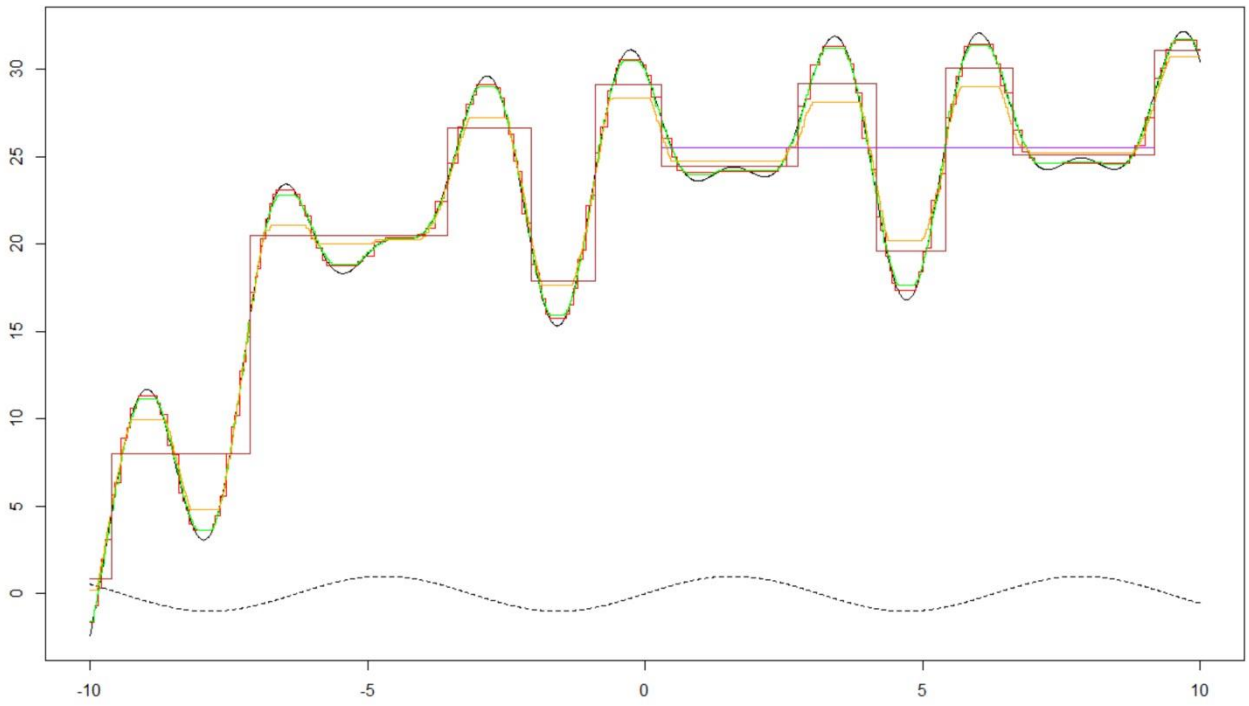


1. Red and blue(hardly visible) are for maxdepth=maxdepth and learning rate = 0.5. Red is the summation YP dataframe and blue is for newly predicted values using the optimized model
2. Green and purple are for maxdepth=4 and learning rate=0.01. we can clearly see that the depth is not enough and learning rate is slow.
3. Orange and brown are for learning rate =0.05 and maxdepth=3.

Accuracy –

Model	summation(|f(xi) - Model(xi)|)
1	0.42
2	0.085
3	0.042

We've achieved the accuracy level in all the model. We can see that the model 1 has highest learning rate in all models if we are aiming for 100% accuracy. Model 2 has slow learning as compare to model 2 **For iterations = 100**

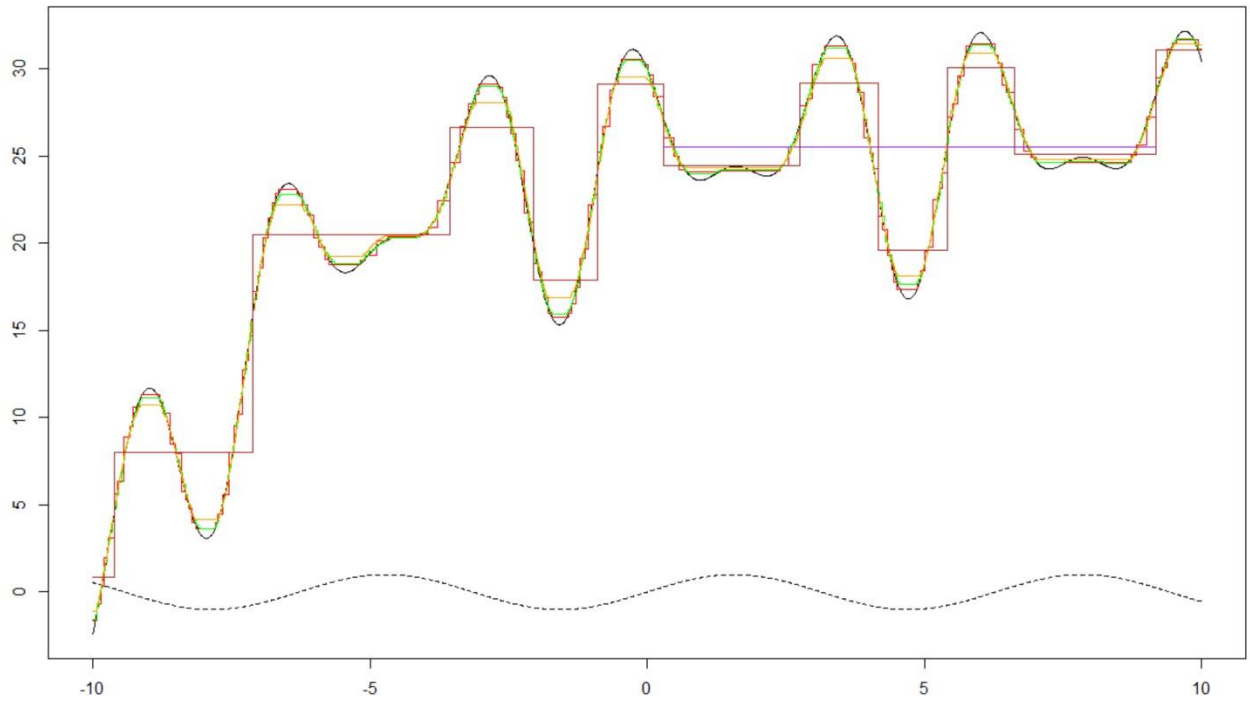


Depth should be greater than 4 as we can see the output of model 2 and 3.

Model	summation(|f(xi) - Model(xi)|)
1	3.99
2	0.79
3	0.399

Model 2 has the slowest learning rate out of all the 3 models. We can find the optimal number of iterations for model 2 for our desired accuracy by decreasing the iterations number.

For iterations = 200



Almost the same plot as the iterations =100 but there can be minute changes which are not transparent.

Model	summation(|f(xi) - Model(xi)|)
1	3.99
2	0.79
3	0.399

Same accuracy as with the iterations =100. That means if we want to achieve lower than this either we've to tune the model's accuracy limit or tune the learning rate.

References

[1] - https://www.youtube.com/watch?v=9wn1f-30_ZY

[2] – Arthur Charpentier - https://www.r-bloggers.com/classification-from-scratch-boosting-11-8/ Code

Help –

[

1- https://stackoverflow.com/questions/22306175/change-negative-values-in-dataframe-columnto-absolute-value] – Stackoverflow.com
2- http://benalexkeen.com/gradient-boosting-in-python-using-scikit-learn/

]