**By Gabriel Vasconcelos**

We discussed some of the parameters we have to tune to estimate a boosting model using the xgboost package. In this post I will discuss the two parameters that were left out in part I, which are the gamma and the min\_child\_weight. These two parameters are much less obvious to understand but they can significantly change the results. Unfortunately, the best way to set them changes from dataset to dataset and we have to test a few values to select the best model. Note that there are many other parameters in the xgboost package. I am only showing the ones I use more.

**Parameters**

* **min\_child\_weigth:** This parameter puts a lower bound on the number of instances in a terminal node. Bigger values require more instances in terminal nodes, which makes each tree in the boosting smaller and the algorithm more conservative.
* **gamma:** Controls the minimum reduction in the loss function required to grow a new node in a tree. This parameter is sensitive to the scale of the loss function, which will be linked to the scale of your response variable. The xgboost documentation defines general loss functions as L(\theta) = \sum_i \ell(\theta). In the quadratic loss, for example, we have \ell(\theta) = (y_i - \hat{y}_i)^2. If we want to use a gamma that requires a RMSE reduction in a split of at least mour gamma must be in the order of (m*N)^2, where Nis the sample size.

**Examples**

I will use a housing dataset from the Ecdat package, which includes 546 observations of sold houses with their prices and characteristics. The objective is to change the two parameters keeping everything else constant and see what happens.

library(xgboost)

library(ggplot2)

library(reshape2)

library(Ecdat)

data = Housing

N = nrow(data)

# = divided price by 10000 to use smaller values of gamma = #

y = data$price/10000

x = data[,-1]

# = Transform categorical variables into dummies = #

for(i in 1:ncol(x)){

if(is.factor(x[,i])){

x[,i] = ifelse(x[,i] == "yes",1,0)

}

}

x = as.matrix(x)

# = select train and test indexes = #

set.seed(1)

train=sample(1:N,456)

test=setdiff(1:N,train)

# = min\_child\_weights candidates = #

mcw=c(1,10,100,400)

# = gamma candidates = #

gamma=c(0.1,1,10,100)

# = train and test data = #

xtrain = x[train,]

ytrain = y[train]

xtest = x[test,]

ytest = y[test]

**min\_child\_weight**

The code below runs the boosting for the four min\_child\_weights in the mcw vector. You can see the convergence and the test RMSE for each setup below the code. The results show a significant change in the model as we move on the min\_child\_weight. Values of 1 and 10 seems to overfit the data a little while and a value of 400 ignores a lot of information and returns a poor predictive model. The best solution in this case was for min\_child\_weight = 100.

set.seed(1)

conv\_mcw = matrix(NA,500,length(mcw))

pred\_mcw = matrix(NA,length(test), length(mcw))

colnames(conv\_mcw) = colnames(pred\_mcw) = mcw

for(i in 1:length(mcw)){

params = list(eta = 0.1, colsample\_bylevel=2/3,

subsample = 1, max\_depth = 6,

min\_child\_weight = mcw[i], gamma = 0)

xgb = xgboost(xtrain, label = ytrain, nrounds = 500, params = params)

conv\_mcw[,i] = xgb$evaluation\_log$train\_rmse

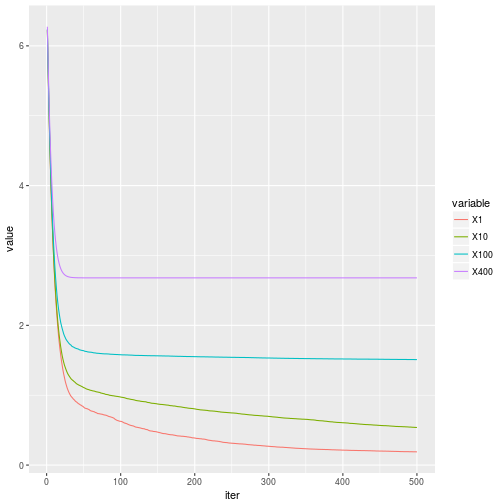
pred\_mcw[,i] = predict(xgb, xtest)

}

conv\_mcw = data.frame(iter=1:500, conv\_mcw)

conv\_mcw = melt(conv\_mcw, id.vars = "iter")

ggplot(data = conv\_mcw) + geom\_line(aes(x = iter, y = value, color = variable))



(RMSE\_mcw = sqrt(colMeans((ytest-pred\_mcw)^2)))

## 1 10 100 400

## 1.744679 1.751487 1.690150 2.607713

**gamma**

The results for gamma are similar to the min\_child\_weight. The models in the middle (gamma = 1 and gamma = 10) are superior in terms of predictive accuracy. Unfortunately, the convergence plot does not give us any clue on which model is the best. We have to test the model in a test sample or in a cross-validation scheme to select the most accurate. Both the min\_child\_weight and the gamma are applying shrinkage to the trees by limiting their size, bu they look at different measures to do so.

set.seed(1)

conv\_gamma = matrix(NA,500,length(gamma))

pred\_gamma = matrix(NA,length(test), length(gamma))

colnames(conv\_gamma) = colnames(pred\_gamma) = gamma

for(i in 1:length(gamma)){

params = list(eta = 0.1, colsample\_bylevel=2/3,

subsample = 1, max\_depth = 6, min\_child\_weight = 1,

gamma = gamma[i])

xgb = xgboost(xtrain, label = ytrain, nrounds = 500, params = params)

conv\_gamma[,i] = xgb$evaluation\_log$train\_rmse

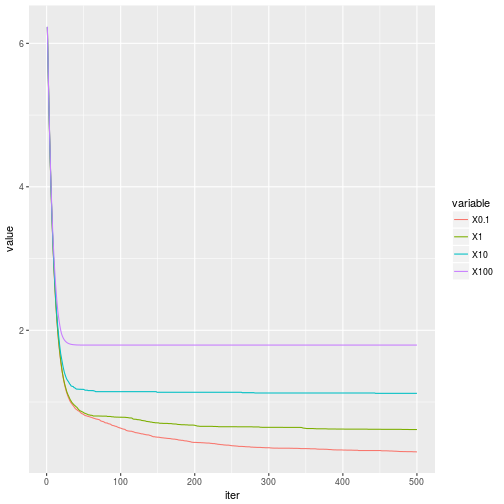
pred\_gamma[,i] = predict(xgb, xtest)

}

conv\_gamma = data.frame(iter=1:500, conv\_gamma)

conv\_gamma = melt(conv\_gamma, id.vars = "iter")

ggplot(data = conv\_gamma) + geom\_line(aes(x = iter, y = value, color = variable))



(RMSE\_gamma = sqrt(colMeans((ytest-pred\_gamma)^2)))

## 0.1 1 10 100

## 1.810786 1.737987 1.739655 1.909785