Hypertuning XGBoost parameters

XGBoost have been doing a great job, when it comes to dealing with both categorical and continuous dependant variables. But, how do I select the optimized parameters for an XGBoost problem?

This is how I applied the parameters for a recent Kaggle problem:

```
param <- list(  objective           = "reg:linear",
                booster = "gbtree",
                eta                 = 0.02, # 0.06, #0.01,
                max_depth           = 10, #changed from default of 8
                subsample           = 0.5, # 0.7
                colsample_bytree    = 0.7, # 0.7
                num_parallel_tree   = 5
                # alpha = 0.0001,
                # lambda = 1
)


clf <- xgb.train(   params              = param,
                    data                = dtrain,
                    nrounds             = 3000, #300, #280, #125, #250, # changed from 300
                    verbose             = 0,
                    early.stop.round    = 100,
                    watchlist           = watchlist,
                    maximize            = FALSE,
                    feval=RMPSE
)
```

All I do to experiment is randomly select (with intuition) another set of parameters for improving on the result.

Is there anyway I automate the selection of optimized(best) set of parameters?

(Answers can be in any language. I'm just looking for the technique)

r python xgboost

asked Dec 13 '15 at 14:19

Dawny33
2,387 2 5 41

2 Answers

Whenever I work with xgboost I often make my own homebrew parameter search but you can do it with the caret package as well like KrisP just mentioned.

1. **Caret**

See this answer on Cross Validated for a thorough explanation on how to use the caret package for hyperparameter search on xgboost. How to tune hyperparameters of xgboost trees?

2. **Custom Grid Search**

I often begin with a few assumptions based on Owen Zhang's slides on tips for data science P. 14

GBDT Hyper Parameter Tuning

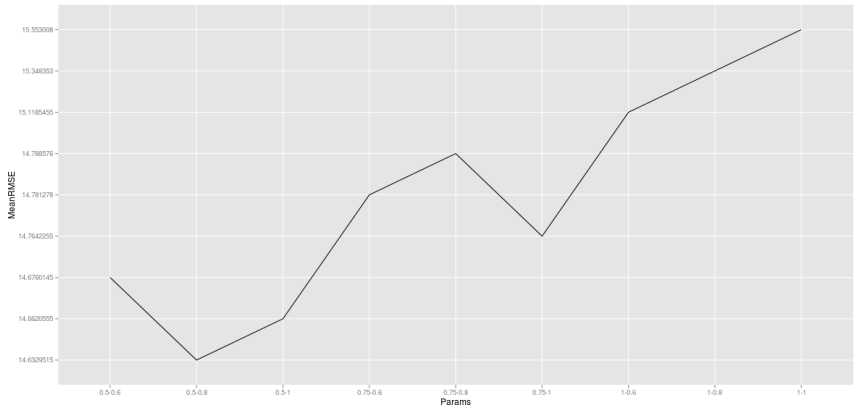
Hyper Parameter	Tuning Approach	Range	Note
# of Trees	Fixed value	100-1000	Depending on datasize
Learning Rate	Fixed => Fine Tune	[2 - 10] / # of Trees	Depending on # trees
Row Sampling	Grid Search	[.5, .75, 1.0]	
Column Sampling	Grid Search	[.4, .6, .8, 1.0]	
Min Leaf Weight	Fixed => Fine Tune	3/(% of rare events)	Rule of thumb
Max Tree Depth	Grid Search	[4, 6, 8, 10]	
Min Split Gain	Fixed	0	Keep it 0

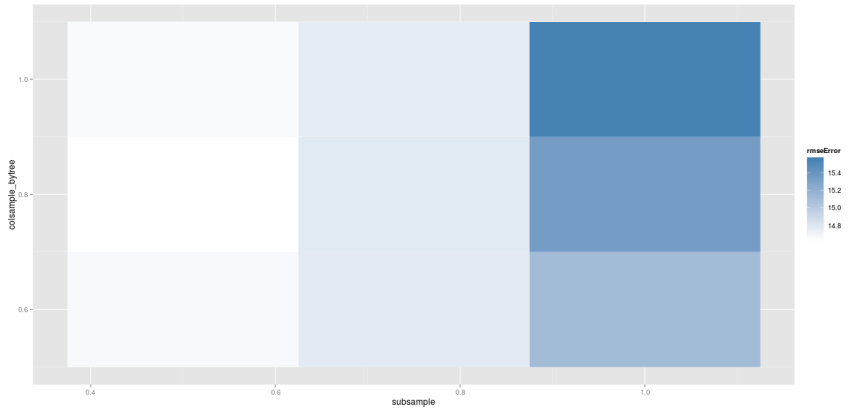
Best GBDT implementation today: https://github.com/tqchen/xgboost
by **Tianqi Chen** (U of Washington)

Here you can see that you'll mostly need to tune row sampling, column sampling and maybe maximum tree depth. This is how I do a custom row sampling and column sampling search for a problem I am working on at the moment:

```
searchGridSubCol <- expand.grid(subsample = c(0.5, 0.75, 1),
                                colsample_bytree = c(0.6, 0.8, 1))
ntrees <- 100

#Build a xgb.DMatrix object
DMMatrixTrain <- xgb.DMatrix(data = yourMatrix, label = yourTarget)

rmseErrorsHyperparameters <- apply(searchGridSubCol, 1, function(parameterList){

    #Extract Parameters to test
    currentSubsampleRate <- parameterList[["subsample"]]
    currentColsampleRate <- parameterList[["colsample_bytree"]]

    xgboostModelCV <- xgb.cv(data =  DMMatrixTrain, nrounds = ntrees, nfold = 5, showsd =
TRUE,
                             metrics = "rmse", verbose = TRUE, "eval_metric" = "rmse",
                             "objective" = "reg:linear", "max.depth" = 15, "eta" = 2/ntrees,
                             "subsample" = currentSubsampleRate, "colsample_bytree" =
currentColsampleRate)

    xvalidationScores <- as.data.frame(xgboostModelCV)
    #Save rmse of the last iteration
    rmse <- tail(xvalidationScores$test.rmse.mean, 1)

    return(c(rmse, currentSubsampleRate, currentColsampleRate))

})
```

And combined with some ggplot2 magic using the results of that apply function you can plot a graphical representation of the search.





In this plot lighter colors represent lower error and each block represents a unique combination of column sampling and row sampling. So if you want to perform an additional search of say eta (or tree depth) you will end up with one of these plots for each eta parameters tested.

I see you have a different evaluation metric (RMPSE), just plug that in the cross validation function and you'll get the desired result. Besides that I wouldn't worry too much about fine tuning the other parameters because doing so won't improve performance too much, at least not so much compared to spending more time engineering features or cleaning the data.

3. **Others**

Random search and Bayesian parameter selection are also possible but I haven't made/found an implementation of them yet.

edited Dec 14 '15 at 14:42

answered Dec 14 '15 at 2:02

wacax
553 1 20

Wow, that's excellently explained. Maybe, I'll wait for a day before accepting :) – Dawny33 Dec 14 '15 at 5:46

You could use the caret package to do hyperparameter space search, either through a grid search , or through random search.

answered Dec 13 '15 at 17:06

KrisP
131 2