dmlc XGBoost

Introduction

- Uses more regularized model formation, which gives better performance and reduces overfitting.
- Boosted trees – add trees that tries to complement the other trees
- Faster, parallelized the prior gains (as we can’t completely parallelize the tree building)

Advantages:

- Memory optimized
    - Mostly done in first pass
    - No dynamic memory is required
- Cache friendly
- More regularized

Features

- Handles missing values
- Parallelized tree constructions
- Can continue training

Installation

```
• R: install.packages( ‘xgboost’ )
• Python: pip install xgboost
```

Parameters:

```
• booster
    o gbtree
    o dart
    o gblinear
• silent
    o 0 - prints messages
    o 1 - doesn’ t print messages
• nthread
    o max (default - uses all threads to run)
• eta
    o learning rate in range of (0, 1)
    o 0.3 (default)
• gamma
    o Also, called as min_split_loss
```

- minimum loss required to make further split
- Helps in avoiding overfitting
- range: [0, infinity)

- nrounds
  - number of iterations for boosting
- max_depth
  - Helps in avoiding overfitting
- maximum depth of the tree
  - range: [1, infinity)
  - 6 (default)
- min_child_weight
  - Helps in avoiding overfitting
  - 1 (default)
  - Range: [0, infinity)
- max_delta_step
  - max delta step we allow each trees weight estimation to be
  - Helps in imbalanced classification
  - 0 (default)
- subsample
  - range: (0, 1]
  - if subsample = 0.5, xgboost takes 50% of data to grow trees
  - Helps i n avoiding overfitting
  - 1 (default)
- colsample_bytree
  - range: (0, 1]
  - Percentage of number of columns to be considered for each tree
  - Helps in avoiding overfitting
  - 1 (default)
- colsample_by_level
  - Percentage of number of columns to be considered at each split
  - Helps in avoiding overfitting
- alpha
  - 0 (default)
  - L1 regularization
  - Helps in avoiding overfitting

- lambda
    - 1 (default)
    - L2 regularization
    - Helps in avoiding overfitting
- scale_pos_weight
    - For unbalanced classification
    - Useful value = (no. of -ves)/(no. of +ves)
- seed
    - for reproduction of random values
- objective
    - reg:linear - linear regression
    - reg:logistic - logistic regression
    - binary:logistic - logistic regression with probability scores
    - binary:logitraw - logistic regression before applying logit function
- eval_metric
    - rmse - root mean squared error
    - mae - mean absolute error
    - error - error in the classification (1 - accuracy)
    - auc - area under the curve
    - logloss - logistic loss

R Code

```
features = c( 'feature-1' , 'feature-2' ….., 'feature-n' )

target = 'Y'

train_x = subset(train, select = features)

train_y = train$Y

test_x = subset(test, select = features)

test_y = test$Y


dtrain = xgb.DMatrix(data = train_x, label = train_y)

dtest = xgb.DMatrix(data = test_x, label = test_y)
```

```
# use xgboost for simple model
# use xgb.train for advanced modelling
watchlist = list(train = dtrain, test = dtest) # watchlist is used to calculate measures and print
fit = xgb.train(data = dtrain, max_depth = 2, eta = 0.3, nround = 2,
      watchlist = watchlist, objective = 'binary:logistic', eval_metric = 'logloss')
# use booster = 'gblinear' to find a linear link between parameters

# get info out of DMatrix
label = getinfo(dtrain, 'label')

# importance
imp = xgb.importance(model = fit)
xgb.plot.importance(importance_matrix = imp)

# save models
xgb.save(fit, 'xgboost.model')
```

Complete R – Code with hyper parameter tuning

```
# # Say o_train, o_test, o_valid are training, testing and validation datasets
# # Y be the target variable
library(Matrix)
library(xgboost)
# # one hot encoding - as xgboost doesn't allow categorical variables
```

```
train_matrix = sparse.model.matrix(Y ~. -1, data = o_train)
test_matrix = sparse.model.matrix(Y ~. -1, data = o_test)
valid_matrix = sparse.model.matrix(Y ~. -1, data = o_valid)
# # form DMatrix to send as input to xgboost
dtrain = xgb.DMatrix(data = as.matrix(train_matrix), label = as.numeric(as.character(train$Y)))
dtest = xgb.DMatrix(data = as.matrix(test_matrix), label = as.numeric(as.character(test$Y)))
dvalid = xgb.DMatrix(data = as.matrix(valid_matrix), label = as.numeric(as.character(valid$Y)))
# # store output
original = as.numeric(as.character(as.vector(train$Y)))
print("# # basic xgboost # #")
hyper_params = list(booster = "gbtree", # default
      objective = "binary:logistic",
      eta = 0.01,
      gamma = 1,
      scale_pos_weight = 85,
      max_depth = 3,
      min_child_weight = 1, # default
      subsample = 0.5,
      colsample_bytree = 0.5
)
watchlist = list(eval = dvalid, train = dtrain)
fit = xgb.train(param = hyper_params, data = dtrain, nrounds = 100, print_every_n = 10, watchlist = watchlist)
predicted = predict(fit, dtest)
```

```
cutoff = 0.5

original = as.integer(as.character(test$Y))

print(performance(predicted = as.numeric(predicted >= cutoff), original = original))


## tuning parameters ##

searchGridSubCol = expand.grid(subsample = c(0.5, 0.75, 1), colsample_bytree = c(0.6, 0.8, 1))

ntrees = 100

errors = apply(searchGridSubCol, 1, function(parameterList){

      # Extract Parameters to test

      currentSubsampleRate = parameterList[["subsample"]]

      currentColsampleRate = parameterList[["colsample_bytree"]]


      fit = xgb.cv(data =  dtrain, nrounds = ntrees, nfold = 5, showsd = TRUE, verbose = TRUE,

            "eval_metric" = "auc", "objective" = "binary:logistic", "max_depth" = 15, "eta" = 2/ntrees,

            "subsample" = currentSubsampleRate, "colsample_bytree" = currentColsampleRate,

            watchlist = watchlist, print_every_n = 10)

      predicted = predict(fit, dtest)

      # cutoff = getCutoff(probabilities, original, plotROC = FALSE, all = FALSE)

      cutoff = 0.5

      original = as.integer(as.character(test$Y))

      perf = performance(predicted = as.numeric(predicted >= cutoff), original = original)
```

```
    auc_scores = as.data.frame(fit$evaluation_log)

    # Save rmse of the last iteration

    auc = cbind(tail(auc_scores, 1), subsample = currentSubsampleRate,
colsample_bytree = currentColsampleRate)


    return(perf = cbind(auc, perf))

})
```