xgboost Package Example (Version 0.3-0)

Tong He

August 29, 2014

1 Introduction

This is an introductory document of using the `xgboost` package in R.

`xgboost` is short for eXtreme Gradient Boosting (Tree). It is an efficient and scalable implementation of Friedman et al. (2001). It supports regression and classification analysis on different types of input datasets.

It has several features:

1. Speed: `xgboost` can automatically do parallel computation on Windows and Linux, with openmp. It is generally over 10 times faster than `gbm`.

2. Input Type: `xgboost` takes several types of input data:

   - Dense Matrix: R's dense matrix, i.e. `matrix`
   - Sparse Matrix: R's sparse matrix `Matrix::dgCMatrix`
   - Data File: Local data files
   - xgb.DMatrix: `xgboost`'s own class. Recommended.

3. Sparsity: `xgboost` accepts sparse input for both tree booster and linear booster.

4. Customization: `xgboost` supports customized objective function and evaluation function

5. Performance: `xgboost` has better performance on several different datasets. Its rising popularity and fame in different Kaggle competitions is the evidence.

2 Example with iris

In this section, we will illustrate some common usage of `xgboost`.

```
> library(xgboost)
> data(iris)
> bst <- xgboost(as.matrix(iris[,1:4]),as.numeric(iris[,5]),
+                nrounds = 5)
```

```
[0]          train-rmse:1.213439
[1]          train-rmse:0.865154
[2]          train-rmse:0.621801
[3]          train-rmse:0.453095
[4]          train-rmse:0.331856
```

```
> xgb.save(bst, 'model.save')
```

```
[1] TRUE
```

```
> bst = xgb.load('model.save')
> pred <- predict(bst, as.matrix(iris[,1:4]))
```

xgboost is the main function to train a Booster, i.e. a model. predict does prediction on the model.

Here we can save the model to a binary local file, and load it when needed. We can't inspect the trees inside. However we have another function to save the model in plain text.

```
> xgb.dump(bst, 'model.dump')
```

```
[1] TRUE
```

The output looks like

```
booster[0]:
0:[f2<2.45] yes=1,no=2,missing=1
    1:leaf=0.147059
    2:[f3<1.65] yes=3,no=4,missing=3
        3:leaf=0.464151
        4:leaf=0.722449
booster[1]:
0:[f2<2.45] yes=1,no=2,missing=1
    1:leaf=0.103806
    2:[f2<4.85] yes=3,no=4,missing=3
        3:leaf=0.316341
        4:leaf=0.510365
```

It is important to know xgboost's own data type: xgb.DMatrix. It speeds up xgboost.

We can use xgb.DMatrix to construct an xgb.DMatrix object:

```
> iris.mat <- as.matrix(iris[,1:4])
> iris.label <- as.numeric(iris[,5])
> diris <- xgb.DMatrix(iris.mat, label = iris.label)
> class(diris)
```

```
[1] "xgb.DMatrix"
```

```
> getinfo(diris,'label')

  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [28] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
 [55] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [82] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3
[109] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[136] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

We can also save the matrix to a binary file. Then load it simply with xgb.DMatrix

```
> xgb.DMatrix.save(diris, 'iris.xgb.DMatrix')

[1] TRUE

> diris = xgb.DMatrix('iris.xgb.DMatrix')
```

3 Advanced Examples

The function xgboost is a simple function with less parameters, in order to be R-friendly. The core training function is wrapped in xgb.train. It is more flexible than xgboost, but it requires users to read the document a bit more carefully.

xgb.train only accept a xgb.DMatrix object as its input, while it supports some additional features as custom objective and evaluation functions.

```
> logregobj <- function(preds, dtrain) {
+    labels <- getinfo(dtrain, "label")
+    preds <- 1/(1 + exp(-preds))
+    grad <- preds - labels
+    hess <- preds * (1 - preds)
+    return(list(grad = grad, hess = hess))
+ }
> evalerror <- function(preds, dtrain) {
+    labels <- getinfo(dtrain, "label")
+    err <- sqrt(mean((preds-labels)^2))
+    return(list(metric = "MSE", value = err))
+ }
> dtest <- slice(diris,1:100)
> watchlist <- list(eval = dtest, train = diris)
> param <- list(max_depth = 2, eta = 1, silent = 1)
> bst <- xgb.train(param, diris, nround = 2, watchlist, logregobj, evalerror)

[1]          eval-MSE:3.264351        train-MSE:4.559875
[2]          eval-MSE:58.79757        train-MSE:66.24299
```

3

The gradient and second order gradient is required for the output of customized objective function.

We also have `slice` for row extraction. It is useful in cross-validation.

For a walkthrough demo, please see `R-package/demo/demo.R` for further details.

4 The Higgs Boson competition

We have made a demo for the Higgs Boson Machine Learning Challenge. Here are the instructions to make a submission

1. Download the datasets and extract them to `data/`.

2. Run scripts under `xgboost/demo/kaggle-higgs/`: higgs-train.R and higgs-pred.R. The computation will take less than a minute on Intel i7.

3. Go to the submission page and submit your result.

We provide a script to compare the time cost on the higgs dataset with `gbm` and `xgboost`. The training set contains 350000 records and 30 features.

`xgboost` can automatically do parallel computation. On a machine with Intel i7-4700MQ and 24GB memories, we found that `xgboost` costs about 35 seconds, which is about 20 times faster than `gbm`. When we limited `xgboost` to use only one thread, it was still about two times faster than `gbm`.

Meanwhile, the result from `xgboost` reaches 3.60@AMS with a single model. This results stands in the top 30% of the competition.

References

Friedman, Jerome H. (2001). Greedy function approximation: a gradient boosting machine. In *Annals of Statistics* (2001): 1189-1232.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics* 28.2 (2000):337-407.