Bike sharing

Load default libraries

```
library(data.table)
library(dplyr)
library(lubridate)
library(ggplot2)
library(caret)
library(stringi)
library(xgboost)
library(reshape2)
theme_set(theme_bw())
set.seed(42)
```

Read the data

```
full_train = fread("./train.csv", header = T, sep = ",", integer64 = "numeric")
test = fread("./test.csv", header = T, sep = ",", integer64 = "numeric")
```

Change type to categorical variables

```
fix_types = function(data) {
  data %>%
    mutate(
      datetime = ymd_hms(datetime),
      workingday = as.character(workingday),
      weather = as.character(weather)) %>%
    mutate(
      year = as.character(year(datetime)),
      month = as.character(month(datetime)),
      day = as.character(mday(datetime)),
      wday = as.character(wday(datetime)),
      hour = as.character(hour(datetime)))

}
full_train = fix_types(full_train)
test = fix_types(test)

train = full_train[as.integer(full_train$day) < 13]
validate = full_train[as.integer(full_train$day) >= 13]
```

Prepare features

Prepare target

We going to predict log(Y+1) to optimize the target cost function

```r
preparedTrainTarget = log(train$count + 1)
preparedValidateTarget = log(validate$count + 1)
preparedFullTarget =log(full_train$count + 1)
```

Xgboost train and cross-validate

Interesting link: how to tune hyperparameters

```r
dtrain <- xgb.DMatrix(train_matrix, label = preparedTrainTarget)

xgbControl = list(
   subsample=0.8, colsample_bytree = 0.8, metrics=list("rmse"), gamma = 0.9,
                  max.depth = 6, eta = 0.1, alpha = 1, lambda = 1, objective = "reg:linear"
)
model = xgboost(params = xgbControl, data = dtrain,
                nround=2000, nthread = 4, print.every.n = 500)
```

```
## [0]  train-rmse:3.901571
## [500]    train-rmse:0.327442
## [1000]   train-rmse:0.318005
## [1500]   train-rmse:0.313631
```

```r
history <- xgb.cv(params = xgbControl, data = dtrain,
                  nround=2000, nthread = 4, nfold = 10, print.every.n = 500)
```

```
## [0]  train-rmse:3.900603+0.005327    test-rmse:3.900368+0.053800
## [500]    train-rmse:0.326971+0.001496    test-rmse:0.379543+0.023749
## [1000]   train-rmse:0.318975+0.001247    test-rmse:0.374132+0.022885
## [1500]   train-rmse:0.315297+0.000965    test-rmse:0.372025+0.022676
```

```r
print(tail(history))
```

```
##    train.rmse.mean train.rmse.std test.rmse.mean test.rmse.std
## 1:        0.313641       0.001088       0.371084      0.022632
## 2:        0.313632       0.001105       0.371092      0.022631
## 3:        0.313617       0.001109       0.371057      0.022629
## 4:        0.313617       0.001109       0.371054      0.022626
## 5:        0.313617       0.001109       0.371058      0.022628
## 6:        0.313617       0.001109       0.371056      0.022628
```

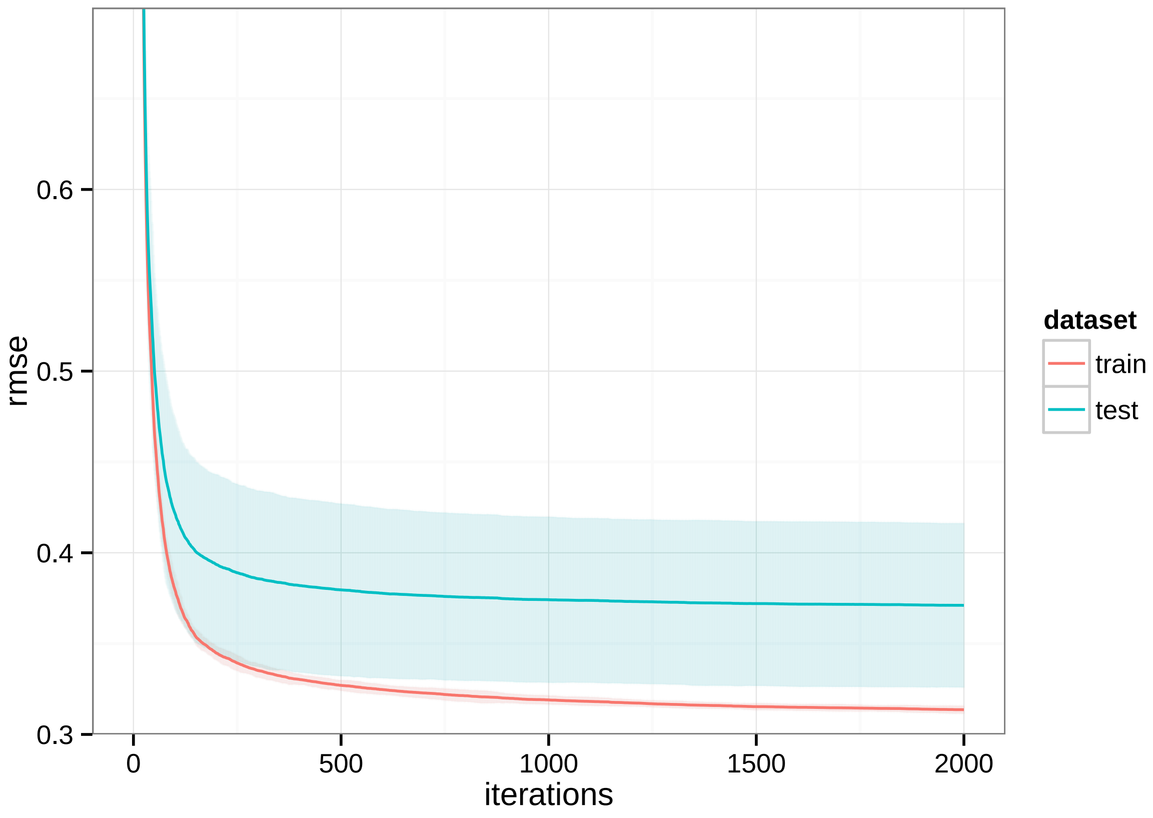
Validate Score

```r
validate_predictions = predict(model, validate_matrix)
RMSE(validate_predictions, preparedValidateTarget)
```

```
## [1] 0.3752475
```

Plot learning curve

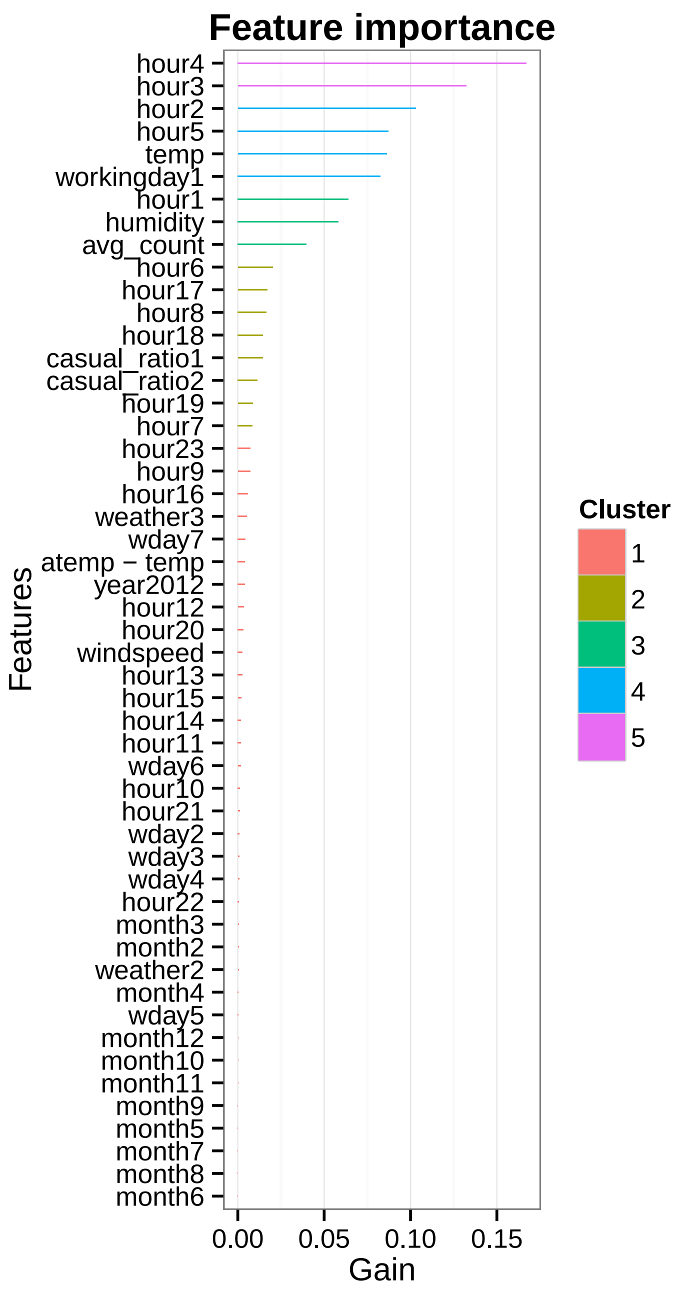
```
plot_learning_curve = function(learning_curves) {
  learning_curves_train = NULL
  learning_curves_train$dataset = 'train'
  learning_curves_train$rmse = learning_curves$train.rmse.mean
  learning_curves_train$rmse.se = learning_curves$train.rmse.std
  learning_curves_train$iterations = 1:nrow(learning_curves)
  learning_curves_test = NULL
  learning_curves_test$dataset = 'test'
  learning_curves_test$rmse = learning_curves$test.rmse.mean
  learning_curves_test$rmse.se = learning_curves$test.rmse.std
  learning_curves_test$iterations = 1:nrow(learning_curves)
  learning_curves_prepared = rbind(as.data.frame(learning_curves_train), as.data.frame(learning_curves_
  ggplot(data = learning_curves_prepared,
         mapping = aes(x=iterations, y=rmse, group = dataset, colour=dataset)) +
    geom_errorbar(aes(ymin=rmse-2*rmse.se, ymax=rmse+2*rmse.se), width=.01, alpha=0.02) +
    geom_line() + coord_cartesian(ylim = c(0.3, 0.7))

}

plot_learning_curve(history)
```



Feature importance

```
imp = xgb.importance(colnames(train_matrix), model = model)
```

```
xgb.plot.importance(imp)
```



Train on full dataset

```
dtrain_final <- xgb.DMatrix(full_train_matrix, label = preparedFullTarget)

model_final = xgboost(params = xgbControl, data = dtrain_final,
                      nround=2000, nthread = 4, print.every.n = 500)
```

```
## [0]  train-rmse:3.914071
## [500]    train-rmse:0.317340
## [1000]   train-rmse:0.309931
## [1500]   train-rmse:0.307571
```

```
history <- xgb.cv(params = xgbControl, data = dtrain_final,
                  nround=2000, nthread = 4, print.every.n = 500, nfold = 10)
```

```
## [0]  train-rmse:3.916993+0.005832    test-rmse:3.916532+0.032254
## [500]    train-rmse:0.319811+0.001173    test-rmse:0.360382+0.017339
## [1000]   train-rmse:0.313787+0.001171    test-rmse:0.356288+0.017049
## [1500]   train-rmse:0.310697+0.001542    test-rmse:0.354289+0.016903
```

Predict and un-log predictions

```
predictions = predict(model_final, test_matrix)
fixed_predictions = exp(predictions) - 1
```

Write the result

```
result = cbind(as.character(test$datetime), fixed_predictions) %>% as.data.frame()
names(result) = c('datetime', 'count')
write.csv(result, 'submission.csv', quote = F, row.names = F)
```