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)),
      hour4 = as.character(round(hour(datetime) / 4)),
      hour8 = as.character(round((hour(datetime) - 4) / 8)),
      hour12 = as.character(round((hour(datetime)) / 12))
      )

}
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

```
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

```
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:3.893698
## [500]    train-rmse:0.250318
## [1000]   train-rmse:0.243826
## [1500]   train-rmse:0.242128
```

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

```
## [0]  train-rmse:3.894248+0.005278    test-rmse:3.894712+0.047065
## [500]    train-rmse:0.251562+0.002323    test-rmse:0.306134+0.018008
## [1000]   train-rmse:0.246225+0.002513    test-rmse:0.302811+0.017838
## [1500]   train-rmse:0.243651+0.002150    test-rmse:0.301413+0.017966
```

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

```
##    train.rmse.mean train.rmse.std test.rmse.mean test.rmse.std
## 1:        0.241291        0.00206       0.300260      0.018055
## 2:        0.241291        0.00206       0.300260      0.018056
## 3:        0.241291        0.00206       0.300260      0.018055
## 4:        0.241291        0.00206       0.300259      0.018054
## 5:        0.241291        0.00206       0.300260      0.018054
## 6:        0.241291        0.00206       0.300261      0.018055
```

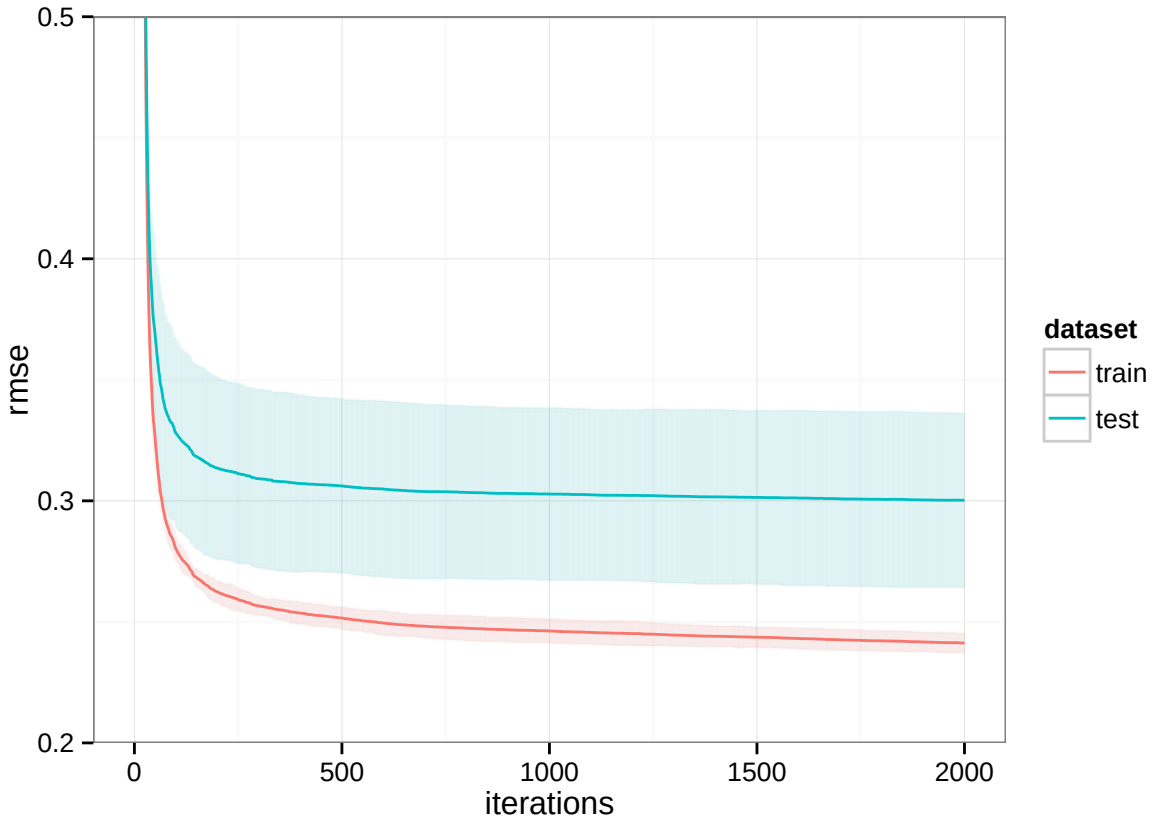
Validate Score

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

```
## [1] 0.3234346
```

Plot learning curve

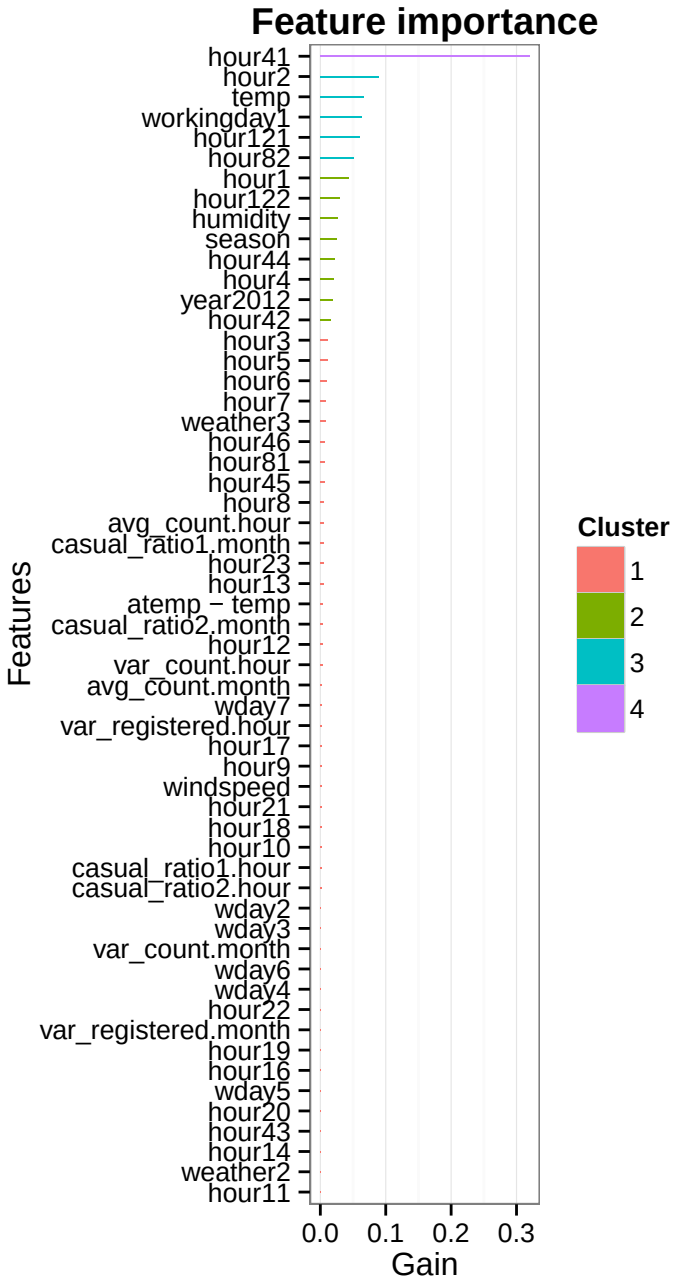
```
plot_learning_curve = function(learning_curves) {
  learning_curves_train = NULL
  learning_curves_train$dataset = 'train'
  learning_curves_train$rmse = learning_curves$train.rmse.mean
  learning_curves_train$rmse.se = learning_curves$train.rmse.std
  learning_curves_train$iterations = 1:nrow(learning_curves)
  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(0.2, 0.5))

}

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:3.910534
## [500]    train-rmse:0.250888
## [1000]   train-rmse:0.243603
## [1500]   train-rmse:0.240877
```

```
history <- xgb.cv(params = xgbControl, data = dtrain_final,
                  nround=2000, nthread = 4, print.every.n = 500, nfold = 3)
```

```
## [0]  train-rmse:3.910445+0.010189    test-rmse:3.911079+0.018799
## [500]    train-rmse:0.251721+0.001122    test-rmse:0.306531+0.009294
## [1000]   train-rmse:0.246767+0.001383    test-rmse:0.304217+0.008636
## [1500]   train-rmse:0.243530+0.002359    test-rmse:0.302450+0.008157
```

Predict and un-log predictions

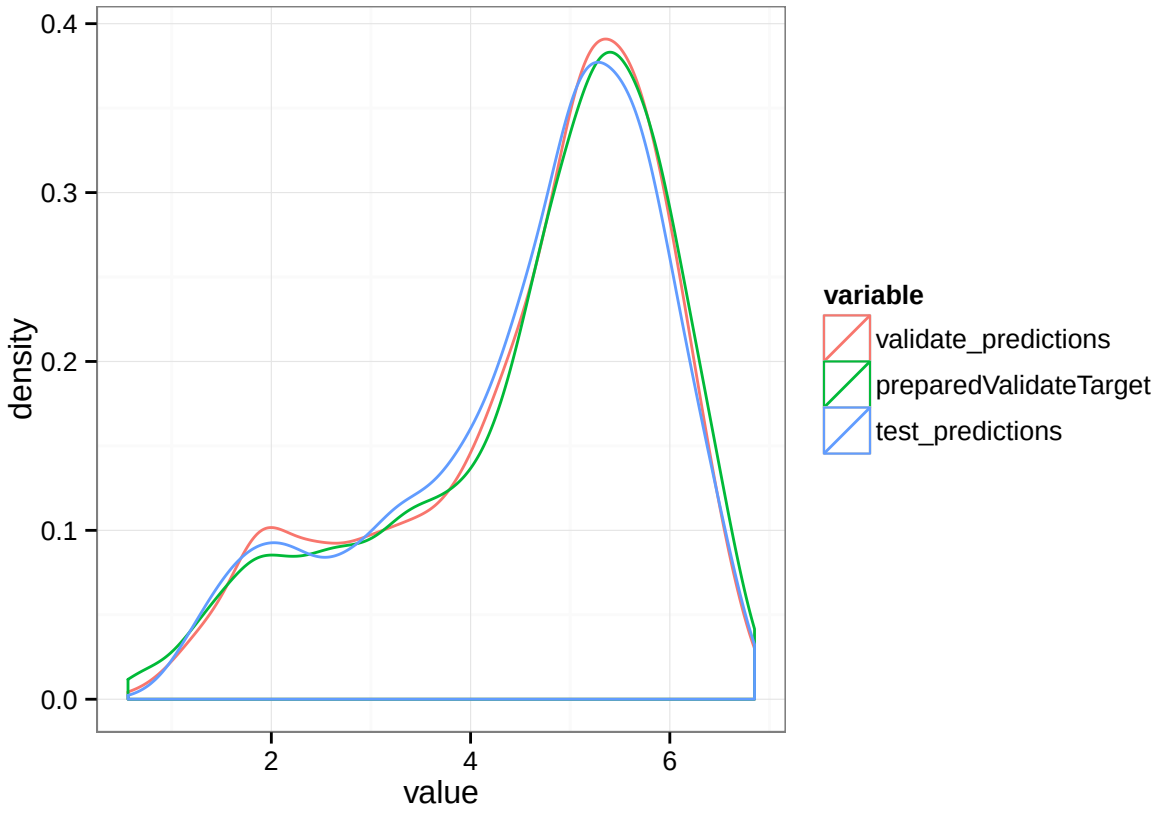
```
predictions = predict(model_final, test_matrix)
fixed_predictions = exp(predictions) - 1
```

Prediction vs Target density plot

```
test_predictions = sample(predictions, length(validate_predictions))
count_value = data.frame(validate_predictions, preparedValidateTarget, test_predictions)
count_value = melt(count_value)
```

```
## No id variables; using all as measure variables
```

```
ggplot(count_value, aes(group = variable, color = variable, x = value)) + geom_density()
```



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)
```