REPORT TREE BASED MODELS

Tigran Karamyan

5 2019

So let's begin from what I have. My data consists of 10051 observations and 9 variables

```
'data.frame':   10051 obs. of  12 variables:
 $ X       : int  0 1 2 3 4 5 6 7 8 9 ... Serial Number
 $ A0      : num  44.5 44.5 44.5 44.5 44.5 ... Coordinates
 $ A1      : num  40.2 40.2 40.2 40.2 40.2 ... Coordinates
 $ B0      : num  44.5 44.5 44.5 44.4 44.5 ... Coordinates
 $ B1      : num  40.2 40.2 40.2 40.2 40.1 ... Coordinates
 $ Distance: num  8400 4600 8600 9700 15900 5900 2500 ... Distances
 $ Weekday : int  2 2 2 2 2 2 2 2 2 2 ... Day Of A Week
 $ Hour    : int  17 17 17 17 17 17 17 17 17 17 ...
 $ Dists   : Factor w/ 10050 levels "['0', '1', '100', '250', '300',  ... Path Distances
 $ Path    : Factor w/ 10038 levels "arshakuniats+avenue+4th+lane head+ ... Path Names
 $ ETA     : num  22 12 19 21 30 15 9 21 24 17 ... Estimated Time Of Arrival on the S=V*T base
 $ ETA_JAM : num  23 12 21 22 41 24 9 25 32 23 ... Estimated Time Of Arrival with the effect of traffic
```

As my initial data consists of variables that I wouldn't use building a model I reduced the dimensions of initial data by eliminating the Path and Dists variables and split the data into 3 parts(TEST, TRAIN AND VALIDATION).

I decided to predict ETA Jam in Yerevan with the Tree based Models. First I Use decision tree regression model to predict ETA_JAM by training its hyperparameters. Let's talk a bit about Tree Based Models. Decision tree learning is a method commonly used in data mining. The goal is to create a model that predicts the value of a target variable based on several input variables. Each interior node corresponds to one of the input variables; there are edges to children for each of the possible values of that input variable. Each leaf represents a value of the target variable given the values of the input variables represented by the path from the root to the leaf.

Decision trees used in data mining are of two main types:

- Classification tree analysis is when the predicted outcome is the class (discrete) to which the data belongs.
- Regression tree analysis is when the predicted outcome can be considered a real number (e.g. the price of a house, or a patient's length of stay in a hospital).

Some techniques, often called ensemble methods, construct more than one decision tree:

- Boosted trees Incrementally building an ensemble by training each new instance to emphasize the training instances previously mis-modeled. A typical example is AdaBoost. These can be used for regression-type and classification-type problems.
- Bootstrap aggregated (or bagged) decision trees, an early ensemble method, builds multiple decision trees by repeatedly resampling training data with replacement, and voting the trees for a consensus prediction.
- A random forest is a specific type of bootstrap aggregating. Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model. Therefore, in Random Forest, only a

random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).Another difference is that „deep“ decision trees might suffer from overfitting. Random Forest prevents overfitting most of the time, by creating random subsets of the features and building smaller trees using these subsets.

```
Data <- read.csv("Data.csv")
ClData <- select(Data, - c("X", "Path", "Dists"))
str(ClData)
```

```
## 'data.frame':    10051 obs. of  9 variables:
##  $ A0      : num  44.5 44.5 44.5 44.5 44.5 ...
##  $ A1      : num  40.2 40.2 40.2 40.2 40.2 ...
##  $ B0      : num  44.5 44.5 44.5 44.4 44.5 ...
##  $ B1      : num  40.2 40.2 40.2 40.2 40.1 ...
##  $ Distance: num  8400 4600 8600 9700 15900 5900 2500 9600 10400 8400 ...
##  $ Weekday : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ Hour    : int  17 17 17 17 17 17 17 17 17 17 ...
##  $ ETA     : num  22 12 19 21 30 15 9 21 24 17 ...
##  $ ETA_JAM : num  23 12 21 22 41 24 9 25 32 23 ...
```

```
# Set seed and create assignment
set.seed(1)
assign <- sample(1:3, size = nrow(ClData), prob = c(0.7, 0.15, 0.15), replace = TRUE)

# Create a train and tests from the original data frame
Data_train <- ClData[assign == 1, ]    # subset ETA_JAM to training indices only
Data_valid <- ClData[assign == 2, ]  # subset ETA_JAM to validation set only
Data_test <- ClData[assign == 3, ]   # subset ETA_JAM to test indices only
```

I Use validation dataset to tune all used model hyperparameters with hyper_grid. In other wards I ceated a matrix with all “possible” values of hyperparameters and built many models and chose the one that had minimum rmse. My tuned hyperparameters are * Minsplit - the minimum number of observations that must exist in a node in order for a split to be attempted. * cp - complexity parameter. Any split that does not decrease the overall lack of fit by a factor of cp is not attempted. For instance, with anova splitting, this means that the overall R-squared must increase by cp at each step. The main role of this parameter is to save computing time by pruning off splits that are obviously not worthwhile. * Maxdepth - Set the maximum depth of any node of the final tree, with the root node counted as depth 0. Computed RMSE on Best model is 3.17, that means our ETA_JAM prediction in average differs from the real life ETA by 3.17 minutes.

```
minsplit <- c(14,15,16)
maxdepth <- c(9,10,11)
cp <- c(0,0.0001,0.001,0.01)

# Create a data frame containing all combinations
hyper_gridDT <- expand.grid(minsplit = minsplit, maxdepth = maxdepth, cp = cp)
nrow(hyper_gridDT)
```

```
## [1] 36
```

```r
# Create an empty list to store models
DT_models <- list()

# Write a loop over the rows of hyper_grid to train the grid of models
for (i in 1:nrow(hyper_gridDT)) {

  # Train a model and store in the list
  DT_models[[i]] <- rpart(formula = ETA_JAM ~ .,
                          data = Data_train,
                          method = "anova",
                          minsplit = hyper_gridDT$minsplit[i],
                          maxdepth = hyper_gridDT$maxdepth[i],
                          cp = hyper_gridDT$cp[i])

}

# Create an empty vector to store RMSE values
rmse_DT <- c()

# Write a loop over the models to compute validation RMSE
for (i in 1:nrow(hyper_gridDT)) {

  # Generate predictions on grade_valid
  pred_DT <- predict(object = DT_models[[i]],
                     newdata = Data_valid)

  # Compute validation RMSE and add to the
  rmse_DT[i] <- rmse(actual = Data_valid$ETA_JAM,
                     predicted = pred_DT)
}

# Identify the model with smallest validation set RMSE
Best_DT <- DT_models[[which.min(rmse_DT)]]

# Print the model paramters of the best model
Best_DT$control
```

```
## $minsplit
## [1] 15
##
## $minbucket
## [1] 5
##
## $cp
## [1] 0
##
## $maxcompete
## [1] 4
##
## $maxsurrogate
## [1] 5
##
## $usesurrogate
```

```
## [1] 2
##
## $surrogatestyle
## [1] 0
##
## $maxdepth
## [1] 10
##
## $xval
## [1] 10
```

```
Best_DT$ordered
```

```
##       A0       A1       B0       B1 Distance  Weekday     Hour      ETA
##    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE
```

```
# Compute test set RMSE on best_model
pred_DT1 <- predict(object = Best_DT,
                newdata = Data_test)
rmse(actual = Data_test$ETA_JAM,
     predicted = pred_DT1)
```

```
## [1] 3.177001
```

For Random Forest model I chose below mentioned parameters * Mtry - Number of variables randomly sampled as candidates at each split. Note that the default value for regression is (p/3). * Ntree - Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times. * Nodesize - Minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown (and thus take less time). Computed RMSE on Best model is 2.64, that means our ETA_JAM prediction in average differs from the real life ETA by 2.64 minutes.

```
# Train a Random Forest
set.seed(2)  # for reproducibility
# Establish a list of possible values for mtry, nodesize and sampsize
mtry <- c(5,6,7)
nodesize <- c(5,7,9,11)
ntree <- c(35)

# Create a data frame containing all combinations
hyper_gridRF <- expand.grid(mtry = mtry, nodesize = nodesize, ntree = ntree)
nrow(hyper_gridRF)
```

```
## [1] 12
```

```
# Create an empty vector to store RF_models
RF_models <- c()

# Write a loop over the rows of hyper_grid to train the grid of models
for (i in 1:nrow(hyper_gridRF)) {

# Train a Random Forest model
```

```
RF_models[[i]] <- randomForest(formula = ETA_JAM ~ .,
                      data = Data_train,
                      mtry = hyper_gridRF$mtry[i],
                      nodesize = hyper_gridRF$nodesize[i],
                      ntree = hyper_gridRF$ntree[i])

}
# Create an empty vector to store RMSE values
rmse_RF <- c()

# Write a loop over the models to compute validation RMSE
for (i in 1:nrow(hyper_gridRF)) {

  # Generate predictions on grade_valid
  pred_RF <- predict(object = RF_models[[i]],
                     newdata = Data_valid)

  # Compute validation RMSE and add to the
  rmse_RF[i] <- rmse(actual = Data_valid$ETA_JAM,
                     predicted = pred_RF)
}

# Identify the model with smallest validation set RMSE
Best_RF <- RF_models[[which.min(rmse_RF)]]
Best_RF
```

```
##
## Call:
##  randomForest(formula = ETA_JAM ~ ., data = Data_train, mtry = hyper_gridRF$mtry[i],      nodesize =
##                Type of random forest: regression
##                      Number of trees: 35
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 6.383786
##                     % Var explained: 94.27
```

```
# Compute test set RMSE on best_model
pred_RF1 <- predict(object = Best_RF,
                    newdata = Data_test)
rmse(actual = Data_test$ETA_JAM,
     predicted = pred_RF1)
```

```
## [1] 2.644275
```

For XGBoosting model I chose below mentioned parameters * Eta - control the learning rate: scale the contribution of each tree by a factor of $0 <$ eta < 1 when it is added to the current approximation. Used to prevent overfitting by making the boosting process more conservative. Lower value for eta implies larger value for nrounds: low eta value means model more robust to overfitting but slower to compute. Default: 0.3 * Max_depth - maximum depth of a tree. Default: 6 * Subsample - subsample ratio of the training instance. Setting it to 0.5 means that xgboost randomly collected half of the data instances to grow trees and this will prevent overfitting. It makes computation shorter (because less data to analyse). It is advised to use this parameter with eta and increase nrounds. Default: 1 * Min_child_weight - minimum sum of

instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. In linear regression mode, this simply corresponds to minimum number of instances needed to be in each node. The larger, the more conservative the algorithm will be. Default: 1 * Early_stopping_rounds - If NULL, the early stopping function is not triggered. If set to an integer k, training with a validation set will stop if the performance doesn't improve for k rounds. * Nrounds - max number of boosting iterations.

Computed RMSE on Best model is 1.4-1.6, that means our ETA_JAM prediction in average differs from the real life ETA by 1.4-1.6 minutes.

```
#XGBOOST
set.seed(3)
# Establish a list of possible values for mtry, nodesize and sampsize
maxdepthXGB <- c(6,7,8)
eta <- c(0.08)
nthreads <- c(6,7)
nrounds <- c(34)
subsample <- c(0.5)
min_child_weight <- c(7)
earlystoppingrounds <- c(3)
# Create a data frame containing all combinations
hyper_gridXGB <- expand.grid(earlystoppingrounds=earlystoppingrounds, maxdepth = maxdepthXGB, eta = eta
nrow(hyper_gridXGB)
```

```
## [1] 6
```

```
# Create an empty vector to store RF_models
XGB_models <- c()

# Write a loop over the rows of hyper_grid to train the grid of models
for (i in 1:nrow(hyper_gridXGB)) {

  # Train a Random Forest model
  XGB_models[[i]] <- xgboost(data = as.matrix(Data_train),
                             label = Data_train$ETA_JAM,
                             max.depth = hyper_gridXGB$maxdepth[i],
                             eta = hyper_gridXGB$eta[i],
                             nthread = hyper_gridXGB$nthreads[i],
                             nrounds = hyper_gridXGB$nrounds[i],
                             min_child_weight = hyper_gridXGB$min_child_weight[i],
                             subsample = hyper_gridXGB$subsample[i],
                             early_stopping_rounds = hyper_gridXGB$earlystoppingrounds[i],

                             objective = "reg:linear")

}
```

```
## [1]  train-rmse:21.930248
## Will train until train_rmse hasn't improved in 3 rounds.
##
## [2]  train-rmse:20.188684
## [3]  train-rmse:18.585546
## [4]  train-rmse:17.107838
```

```
## [5]  train-rmse:15.748262
## [6]  train-rmse:14.496161
## [7]  train-rmse:13.344439
## [8]  train-rmse:12.284863
## [9]  train-rmse:11.308429
## [10] train-rmse:10.410357
## [11] train-rmse:9.584089
## [12] train-rmse:8.823274
## [13] train-rmse:8.123830
## [14] train-rmse:7.480310
## [15] train-rmse:6.886019
## [16] train-rmse:6.340073
## [17] train-rmse:5.837327
## [18] train-rmse:5.374892
## [19] train-rmse:4.948432
## [20] train-rmse:4.555552
## [21] train-rmse:4.194599
## [22] train-rmse:3.863683
## [23] train-rmse:3.557408
## [24] train-rmse:3.277711
## [25] train-rmse:3.017929
## [26] train-rmse:2.779454
## [27] train-rmse:2.559425
## [28] train-rmse:2.356544
## [29] train-rmse:2.170150
## [30] train-rmse:1.998637
## [31] train-rmse:1.840759
## [32] train-rmse:1.695174
## [33] train-rmse:1.561750
## [34] train-rmse:1.438405
## [1]  train-rmse:21.932522
## Will train until train_rmse hasn't improved in 3 rounds.
##
## [2]  train-rmse:20.188597
## [3]  train-rmse:18.583174
## [4]  train-rmse:17.108034
## [5]  train-rmse:15.749803
## [6]  train-rmse:14.499596
## [7]  train-rmse:13.349567
## [8]  train-rmse:12.289840
## [9]  train-rmse:11.313811
## [10] train-rmse:10.416547
## [11] train-rmse:9.590149
## [12] train-rmse:8.829681
## [13] train-rmse:8.127969
## [14] train-rmse:7.483381
## [15] train-rmse:6.890924
## [16] train-rmse:6.344022
## [17] train-rmse:5.840854
## [18] train-rmse:5.378239
## [19] train-rmse:4.951884
## [20] train-rmse:4.559726
## [21] train-rmse:4.198593
## [22] train-rmse:3.866024
```

```
## [23] train-rmse:3.559840
## [24] train-rmse:3.277887
## [25] train-rmse:3.019877
## [26] train-rmse:2.782407
## [27] train-rmse:2.562652
## [28] train-rmse:2.360066
## [29] train-rmse:2.173733
## [30] train-rmse:2.002656
## [31] train-rmse:1.844236
## [32] train-rmse:1.699950
## [33] train-rmse:1.565093
## [34] train-rmse:1.441877
## [1]  train-rmse:21.934635
## Will train until train_rmse hasn't improved in 3 rounds.
##
## [2]  train-rmse:20.195190
## [3]  train-rmse:18.590771
## [4]  train-rmse:17.114328
## [5]  train-rmse:15.757767
## [6]  train-rmse:14.504956
## [7]  train-rmse:13.352541
## [8]  train-rmse:12.292759
## [9]  train-rmse:11.317386
## [10] train-rmse:10.417401
## [11] train-rmse:9.591260
## [12] train-rmse:8.830742
## [13] train-rmse:8.130025
## [14] train-rmse:7.484313
## [15] train-rmse:6.890393
## [16] train-rmse:6.345737
## [17] train-rmse:5.842839
## [18] train-rmse:5.378574
## [19] train-rmse:4.952161
## [20] train-rmse:4.559306
## [21] train-rmse:4.196945
## [22] train-rmse:3.864586
## [23] train-rmse:3.559337
## [24] train-rmse:3.278050
## [25] train-rmse:3.018386
## [26] train-rmse:2.780909
## [27] train-rmse:2.560229
## [28] train-rmse:2.358439
## [29] train-rmse:2.172328
## [30] train-rmse:2.001195
## [31] train-rmse:1.843316
## [32] train-rmse:1.698090
## [33] train-rmse:1.563398
## [34] train-rmse:1.439796
## [1]  train-rmse:21.930332
## Will train until train_rmse hasn't improved in 3 rounds.
##
## [2]  train-rmse:20.187632
## [3]  train-rmse:18.582680
## [4]  train-rmse:17.109364
```

```
## [5]  train-rmse:15.749098
## [6]  train-rmse:14.496234
## [7]  train-rmse:13.346741
## [8]  train-rmse:12.286182
## [9]  train-rmse:11.309376
## [10] train-rmse:10.411325
## [11] train-rmse:9.585166
## [12] train-rmse:8.824866
## [13] train-rmse:8.125954
## [14] train-rmse:7.480861
## [15] train-rmse:6.887393
## [16] train-rmse:6.340707
## [17] train-rmse:5.839080
## [18] train-rmse:5.376076
## [19] train-rmse:4.950227
## [20] train-rmse:4.557824
## [21] train-rmse:4.196466
## [22] train-rmse:3.863836
## [23] train-rmse:3.558080
## [24] train-rmse:3.276776
## [25] train-rmse:3.017021
## [26] train-rmse:2.777827
## [27] train-rmse:2.558384
## [28] train-rmse:2.356093
## [29] train-rmse:2.169995
## [30] train-rmse:1.998002
## [31] train-rmse:1.841017
## [32] train-rmse:1.695732
## [33] train-rmse:1.562120
## [34] train-rmse:1.438805
## [1]  train-rmse:21.930462
## Will train until train_rmse hasn't improved in 3 rounds.
##
## [2]  train-rmse:20.186975
## [3]  train-rmse:18.583010
## [4]  train-rmse:17.105957
## [5]  train-rmse:15.745624
## [6]  train-rmse:14.498042
## [7]  train-rmse:13.346692
## [8]  train-rmse:12.287611
## [9]  train-rmse:11.312678
## [10] train-rmse:10.414936
## [11] train-rmse:9.588446
## [12] train-rmse:8.827198
## [13] train-rmse:8.126793
## [14] train-rmse:7.482005
## [15] train-rmse:6.887630
## [16] train-rmse:6.341697
## [17] train-rmse:5.837798
## [18] train-rmse:5.375587
## [19] train-rmse:4.950111
## [20] train-rmse:4.558254
## [21] train-rmse:4.197706
## [22] train-rmse:3.865899
```

```
## [23] train-rmse:3.559017
## [24] train-rmse:3.278395
## [25] train-rmse:3.019666
## [26] train-rmse:2.781065
## [27] train-rmse:2.562109
## [28] train-rmse:2.359229
## [29] train-rmse:2.172707
## [30] train-rmse:2.000589
## [31] train-rmse:1.842763
## [32] train-rmse:1.697374
## [33] train-rmse:1.563326
## [34] train-rmse:1.441257
## [1]  train-rmse:21.937334
## Will train until train_rmse hasn't improved in 3 rounds.
##
## [2]  train-rmse:20.197767
## [3]  train-rmse:18.593437
## [4]  train-rmse:17.116007
## [5]  train-rmse:15.756345
## [6]  train-rmse:14.506552
## [7]  train-rmse:13.357025
## [8]  train-rmse:12.297288
## [9]  train-rmse:11.320125
## [10] train-rmse:10.421971
## [11] train-rmse:9.595101
## [12] train-rmse:8.831970
## [13] train-rmse:8.129140
## [14] train-rmse:7.483629
## [15] train-rmse:6.889904
## [16] train-rmse:6.344508
## [17] train-rmse:5.840924
## [18] train-rmse:5.377888
## [19] train-rmse:4.951675
## [20] train-rmse:4.559465
## [21] train-rmse:4.198092
## [22] train-rmse:3.865559
## [23] train-rmse:3.559687
## [24] train-rmse:3.277710
## [25] train-rmse:3.019293
## [26] train-rmse:2.780004
## [27] train-rmse:2.560086
## [28] train-rmse:2.357430
## [29] train-rmse:2.170891
## [30] train-rmse:1.999342
## [31] train-rmse:1.841454
## [32] train-rmse:1.696102
## [33] train-rmse:1.562366
## [34] train-rmse:1.439250
```

```
# Create an empty vector to store RMSE values
rmse_XGB <- c()

# Write a loop over the models to compute validation RMSE
for (i in 1:nrow(hyper_gridXGB)) {
```

```
  # Generate predictions on grade_valid
  pred_XGB <- predict(object = XGB_models[[i]],
                      newdata = as.matrix(Data_valid))

  # Compute validation RMSE and add to the
  rmse_XGB[i] <- rmse(actual = Data_valid$ETA_JAM,
                      predicted = pred_XGB)
}

# Identify the model with smallest validation set RMSE
Best_XGB <- XGB_models[[which.min(rmse_XGB)]]
Best_XGB
```

```
## ##### xgb.Booster
## raw: 51.6 Kb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, max.depth = ..1, eta = ..2, nthread = ..3,
##     min_child_weight = ..4, subsample = ..5, objective = "reg:linear")
## params (as set within xgb.train):
##   max_depth = "8", eta = "0.08", nthread = "6", min_child_weight = "7", subsample = "0.5", objective
## xgb.attributes:
##   best_iteration, best_msg, best_ntreelimit, best_score, niter
## callbacks:
##   cb.print.evaluation(period = print_every_n)
##   cb.evaluation.log()
##   cb.early.stop(stopping_rounds = early_stopping_rounds, maximize = maximize,
##     verbose = verbose)
## # of features: 9
## niter: 34
## best_iteration : 34
## best_ntreelimit : 34
## best_score : 1.439796
## nfeatures : 9
## evaluation_log:
##     iter train_rmse
##        1  21.934635
##        2  20.195190
## ---
##       33   1.563398
##       34   1.439796
```

```
# Compute test set RMSE on best_model
pred_XGB1 <- predict(object = Best_XGB,
                     newdata = as.matrix(Data_test))
rmse(actual = Data_test$ETA_JAM,
     predicted = pred_XGB1)
```

```
## [1] 1.470152
```

```
bst <- xgb.cv(data = as.matrix(ClData),
              label = ClData$ETA_JAM, nfold = 10,
              nrounds = 35, objective = "reg:linear")
```

```
## [1]  train-rmse:16.748665+0.018528   test-rmse:16.747436+0.168161
## [2]  train-rmse:11.742692+0.013026   test-rmse:11.739182+0.120862
## [3]  train-rmse:8.232843+0.009229    test-rmse:8.232260+0.082629
## [4]  train-rmse:5.772888+0.006443    test-rmse:5.774010+0.058257
## [5]  train-rmse:4.048569+0.004488    test-rmse:4.049982+0.041848
## [6]  train-rmse:2.839872+0.003113    test-rmse:2.840188+0.029861
## [7]  train-rmse:1.992441+0.002146    test-rmse:1.992730+0.021539
## [8]  train-rmse:1.398411+0.001538    test-rmse:1.398597+0.015351
## [9]  train-rmse:0.982027+0.001058    test-rmse:0.982572+0.010961
## [10] train-rmse:0.690224+0.000678    test-rmse:0.691054+0.008581
## [11] train-rmse:0.485739+0.000550    test-rmse:0.486229+0.006442
## [12] train-rmse:0.342435+0.000583    test-rmse:0.342883+0.004775
## [13] train-rmse:0.241815+0.000676    test-rmse:0.242460+0.003615
## [14] train-rmse:0.171177+0.000506    test-rmse:0.172326+0.003006
## [15] train-rmse:0.121788+0.000416    test-rmse:0.123871+0.003363
## [16] train-rmse:0.087289+0.000623    test-rmse:0.089870+0.003338
## [17] train-rmse:0.063024+0.000526    test-rmse:0.066960+0.004322
## [18] train-rmse:0.046462+0.000699    test-rmse:0.051736+0.005172
## [19] train-rmse:0.034479+0.000650    test-rmse:0.041130+0.005903
## [20] train-rmse:0.026297+0.000677    test-rmse:0.034539+0.007193
## [21] train-rmse:0.020472+0.000730    test-rmse:0.030337+0.007859
## [22] train-rmse:0.015876+0.000726    test-rmse:0.027293+0.008531
## [23] train-rmse:0.012926+0.000968    test-rmse:0.025397+0.008885
## [24] train-rmse:0.010821+0.001165    test-rmse:0.024158+0.009188
## [25] train-rmse:0.008750+0.000943    test-rmse:0.023065+0.009588
## [26] train-rmse:0.007335+0.000888    test-rmse:0.022210+0.009679
## [27] train-rmse:0.006279+0.000911    test-rmse:0.021700+0.009804
## [28] train-rmse:0.005363+0.000806    test-rmse:0.021220+0.010002
## [29] train-rmse:0.004729+0.000738    test-rmse:0.020988+0.010037
## [30] train-rmse:0.004209+0.000754    test-rmse:0.020686+0.009924
## [31] train-rmse:0.003757+0.000721    test-rmse:0.020552+0.009975
## [32] train-rmse:0.003354+0.000703    test-rmse:0.020403+0.009870
## [33] train-rmse:0.002993+0.000647    test-rmse:0.020285+0.009871
## [34] train-rmse:0.002720+0.000639    test-rmse:0.020253+0.009854
## [35] train-rmse:0.002416+0.000601    test-rmse:0.020173+0.009812
```

Deteils in Presentation. Should speak about lambda parameter.