Term Project 390.4- 2019

R Markdown

```
pacman::p_load(dplyr, tidyr, ggplot2, magrittr, stringr, mlr)
housing_data = read.csv("housing_data_2016_2017.csv")
```

##Delete features that are irrelevant to sale price

```
housing_data %<>%
  select(-c(HITId, HITTypeId, Title, Description, Keywords, Reward, CreationTime, MaxAssignments,   Req
```

Clean Data

```
housing_data %<>%
  mutate( zip_code = str_extract(full_address_or_zip_code, "[0-9]{5}"))
housing_data %<>%
  mutate(dogs_allowed = ifelse(substr(housing_data$dogs_allowed, 1, 3) == "yes", 1, 0)) %>%
  mutate(cats_allowed = ifelse(substr(housing_data$cats_allowed, 1, 3) == "yes", 1, 0)) %>%
  mutate( pets_allowed = ifelse( cats_allowed + dogs_allowed > 0, 1, 0)) %>%
  mutate(coop_condo = factor(tolower(coop_condo)))
housing_data %<>%
  select(-c(dogs_allowed,cats_allowed, fuel_type))
d = housing_data
d %<>%
  mutate(maintenance_cost = sjmisc::rec(maintenance_cost, rec = "NA = 0 ; else = copy")) %<>%
  mutate(common_charges = sjmisc::rec(common_charges, rec = "NA = 0 ; else = copy"))##recode from NA to
# combine maintaince cost and common charges
d %<>%
  mutate( monthly_cost = common_charges + maintenance_cost)
d %<>%
  mutate(monthly_cost = sjmisc::rec(monthly_cost, rec = "0 = NA ; else = copy"))
## convert garage_exists feature to binary
d %<>%
  mutate(garage_exists = sjmisc::rec(garage_exists, rec = "NA = 0 ; else = copy")) ##recode from NA to
d %<>%
  mutate(garage_exists = sjmisc::rec(garage_exists, rec = " eys = 1; UG = 1 ; Underground = 1; yes = 1
d %<>%
  select(-c(maintenance_cost , common_charges, model_type))
```

##Change variable types

```
d %<>%
  mutate( dining_room_type = as.factor(dining_room_type)) %>%
  mutate(garage_exists = as.character(garage_exists)) %>%
  mutate(garage_exists = as.numeric(garage_exists)) %>%
  mutate( parking_charges = as.character(parking_charges)) %>%
  mutate( parking_charges = as.numeric(parking_charges)) %>%
  mutate(sale_price = as.character(sale_price)) %>%
  mutate(sale_price = as.numeric(sale_price)) %>%
  mutate(total_taxes = as.character(total_taxes)) %>%
```

```r
  mutate(total_taxes = as.numeric(total_taxes)) %>%
  mutate(price_persqft = listing_price_to_nearest_1000 / sq_footage)
```

```
## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion
```

#Added latitude and longitude features using ggmap

```r
#Already run and included in the data
#pacman::p_load(ggmap)
#d %<>%
#  mutate(lat = geocode(full_address_or_zip_code)$lat, lon = #geocode(full_address_or_zip_code)$lon )
#geocoordinates for relevant LIRR stations
lirr_coord = read.csv("coord.csv")
RAD_EARTH = 3958.8
degrees_to_radians = function(angle_degrees){
  for(i in 1:length(angle_degrees))
    angle_degrees[i] = angle_degrees[i]*pi/180
  return(angle_degrees)
}
compute_globe_distance = function(destination, origin){
  destination_rad = degrees_to_radians(destination)
  origin_rad = degrees_to_radians(origin)
  delta_lat = destination_rad[1] - origin_rad[1]
  delta_lon = destination_rad[2] - origin_rad[2]
  h = (sin(delta_lat/2))^2 + cos(origin_rad[1]) * cos(destination_rad[1]) * (sin(delta_lon/2))^2
  central_angle = 2 * asin(sqrt(h))
  return(RAD_EARTH * central_angle)
}
#find the closest LIRR station and compute distance
shortest_lirr_distance = function(all_lirr_coords, house_coords){
  shortest_dist = Inf
  for (i in 1: nrow(all_lirr_coords)){
    ith_lirr = c(all_lirr_coords$lat[i], all_lirr_coords$lon[i])
    new_dist = compute_globe_distance(ith_lirr, house_coords)
    if( new_dist < shortest_dist){
      shortest_dist = new_dist
    }
  }
  return(shortest_dist)
}
d %<>%
  rowwise() %>%
  mutate(shortest_dist = shortest_lirr_distance(lirr_coord, c(lat, lon)) )
#makes any other addresses redundant
d %<>%
  select(-c(zip_code, full_address_or_zip_code, listing_price_to_nearest_1000))
```

We are trying to predict `sale_price`. So let's section our dataset:

```r
####CREATE A COLUMN ID
d %<>%
```

```r
  ungroup(d) %>%
  mutate(id = 1 : 2230)
d %<>%
  mutate(total_taxes = ifelse(d$total_taxes < 1000, NA, total_taxes))
real_y = data.frame(d$id, d$sale_price)
real_d = subset(d, (!is.na(d$sale_price)))
fake_d = subset(d, (is.na(d$sale_price)))
real_d$sale_price = NULL
fake_d$sale_price = NULL
```

#Split the data that has y into train and test sets

```r
train_indices = sample(1 : nrow(real_d), nrow(real_d)*4/5)
training_data = real_d[train_indices, ]
testing_data = real_d[-train_indices, ]
X = rbind(training_data, testing_data, fake_d)
```

#Let's first create a matrix with p columns that represents missingness

```r
M = tbl_df(apply(is.na(X), 2, as.numeric))
colnames(M) = paste("is_missing_", colnames(X), sep = "")
```

#Some of these missing indicators are collinear because they share all the rows they are missing on. Let's filter those out:

```r
M = tbl_df(t(unique(t(M))))
```

#Some featuers did not have missingness so let's remove them:

```r
M %<>% select_if(function(x){sum(x) > 0})
```

#Now let's impute missing data using the package. we cannot fit RF models to the entire dataset (it's 26,000! observations) so we will sample 5 for X1 and for each of the trees and then average. That will be good enough.

```r
pacman::p_load(missForest)
Ximp = missForest(data.frame(X), sampsize = rep(172, ncol(X)))$ximp
```

```
##   missForest iteration 1 in progress...

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want
## to do regression?

## done!
##   missForest iteration 2 in progress...

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want
## to do regression?

## done!
##   missForest iteration 3 in progress...

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want
## to do regression?

## done!
##   missForest iteration 4 in progress...
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want
## to do regression?

## done!
##   missForest iteration 5 in progress...

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want
## to do regression?

## done!
##   missForest iteration 6 in progress...

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want
## to do regression?

## done!
##   missForest iteration 7 in progress...

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want
## to do regression?

## done!
##   missForest iteration 8 in progress...

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want
## to do regression?

## done!
##   missForest iteration 9 in progress...

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want
## to do regression?

## done!
```

```r
Ximp %<>%
  arrange(id)
Xnew = data.frame(cbind(Ximp, M, real_y))
Xnew %<>%
  mutate(price = d.sale_price) %>%
  select(-c(id, d.id, d.sale_price))

linear_mod_impute_and_missing_dummies = lm(price ~ ., data = Xnew)
```

REMOVING MISSING Y SECTION

```r
Data = Xnew
### sale price is our imputed Y
Y = Data$price
Data %<>%
  filter(!is.na(price)) %>%
  select(-price)
```

```r
Xtrain = Data[1:422, ]
Xtest = Data[423:528, ]
Ytrain = Y[1:422]
Ytest = Y[423:528]
dtrain = cbind(Xtrain, Ytrain) ## combine x train with y train, x test with y test
dtest = cbind(Xtest, Ytest)
```

Dropping colinear features

```r
Xtrain %<>%
  select(-c(is_missing_num_total_rooms, is_missing_num_bedrooms, is_missing_price_persqft))
```

##Linear Regression

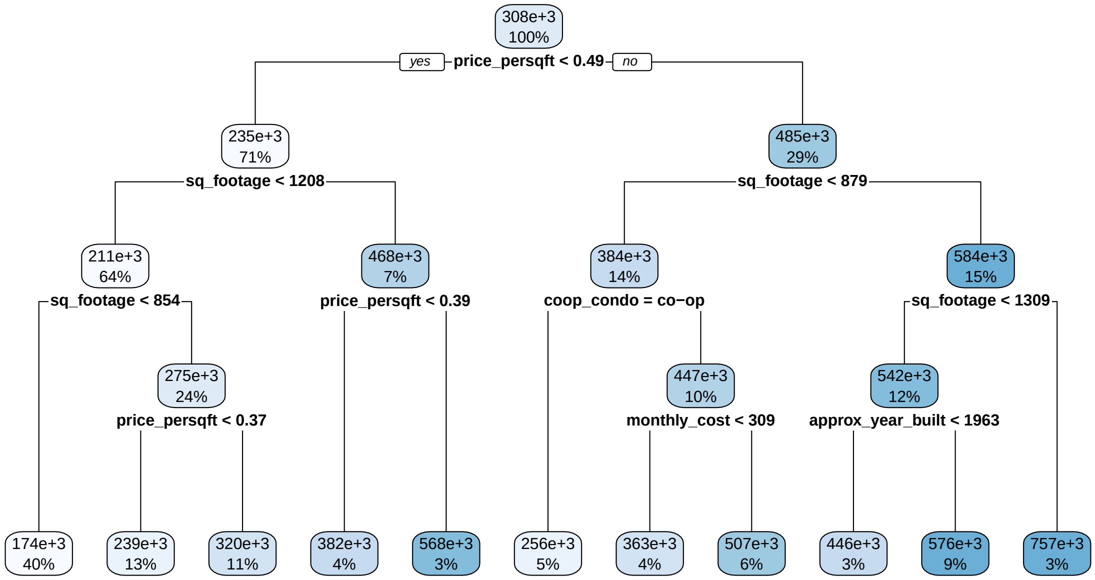
```r
linear = lm(Ytrain ~ ., data = Xtrain)## simple linear model
```

##Linear Model Errors

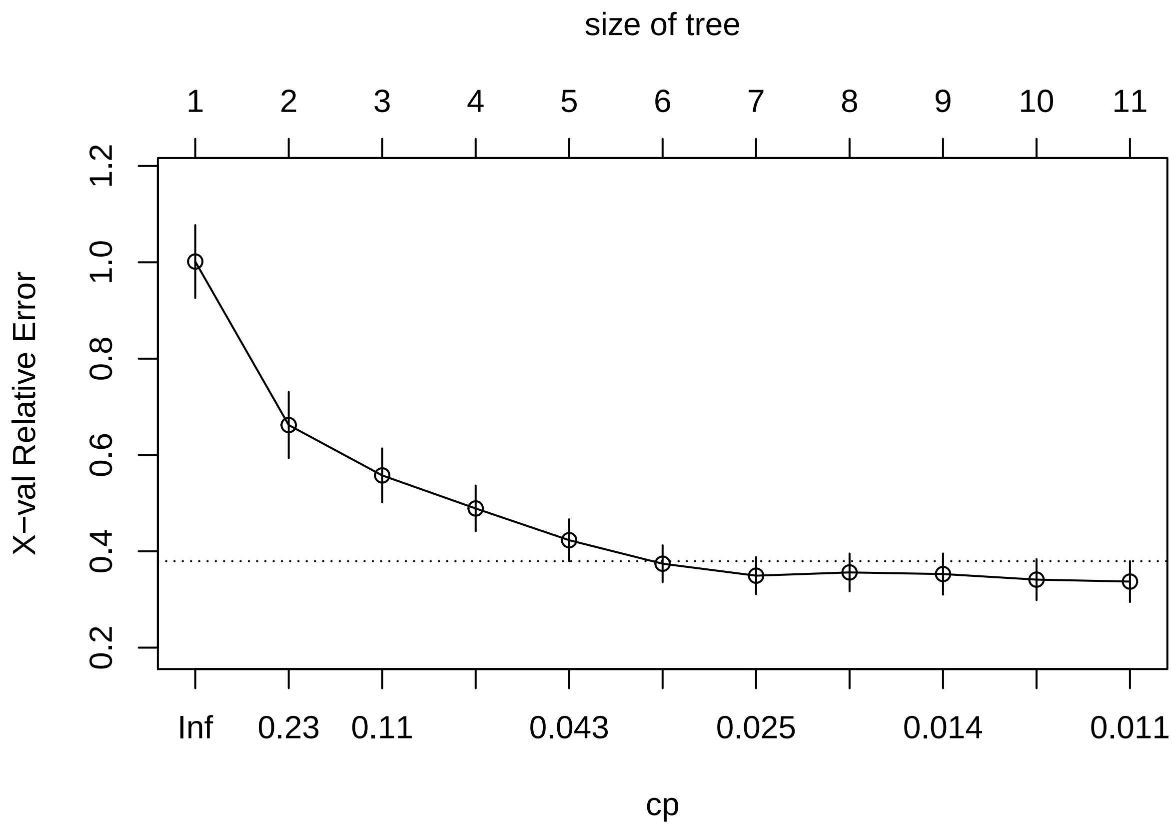
```r
yhat = predict(linear, Xtest)
e = yhat - Ytest
sqrt(sum(e^2) / nrow(Xtest))
```

```
## [1] 84480.55
```

```r
#REGRESSION TREE
pacman::p_load(rsample)#data spliting
pacman::p_load(rpart) #performing reg tree
pacman::p_load(rpart.plot) #ploting reg tree
pacman::p_load(ipred) #bagging
pacman::p_load(caret) #bagging
m1 = rpart(
  formula = Ytrain ~ .,
  data    = Xtrain,
  method  = "anova"
  )
rpart.plot(m1)
```


308e+3
100%
yes
price_persqft < 0.49
no
235e+3
71%
sq_footage < 1208
485e+3
29%
sq_footage < 879
211e+3
64%
sq_footage < 854
468e+3
7%
price_persqft < 0.39
384e+3
14%
coop_condo = co−op
584e+3
15%
sq_footage < 1309
275e+3
24%
price_persqft < 0.37
447e+3
10%
monthly_cost < 309
542e+3
12%
approx_year_built < 1963
174e+3
40%
239e+3
13%
320e+3
11%
382e+3
4%
568e+3
3%
256e+3
5%
363e+3
4%
507e+3
6%
446e+3
3%
576e+3
9%
757e+3
3%

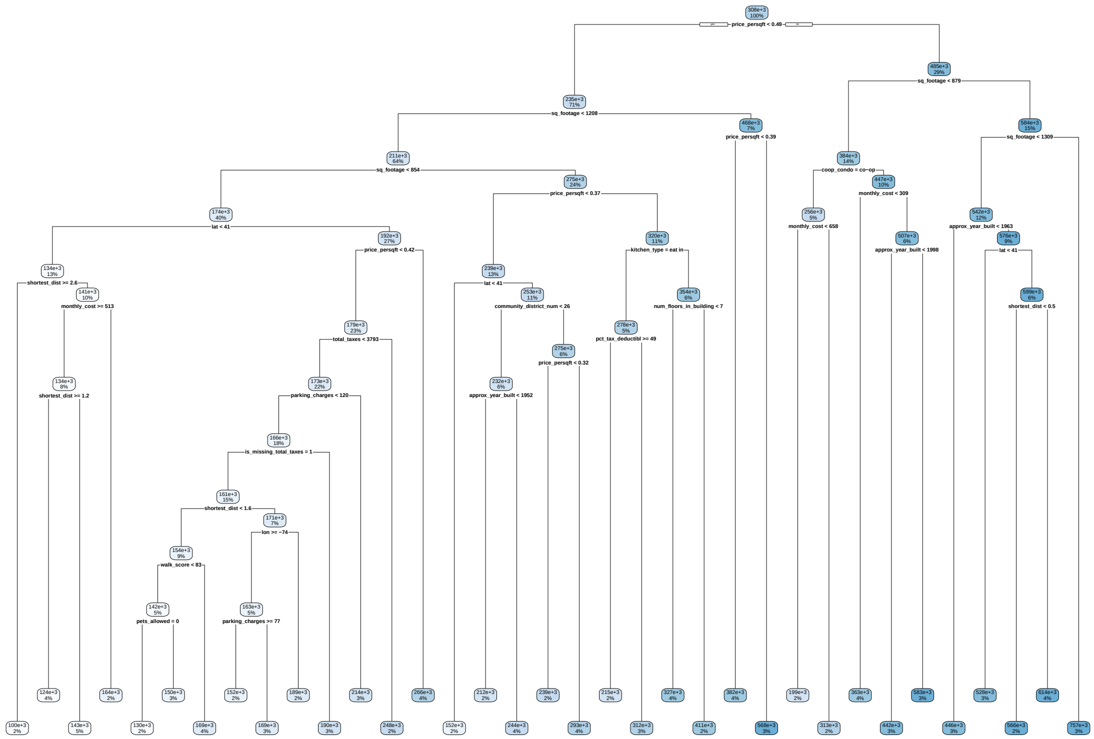
```
plotcp(m1)
```



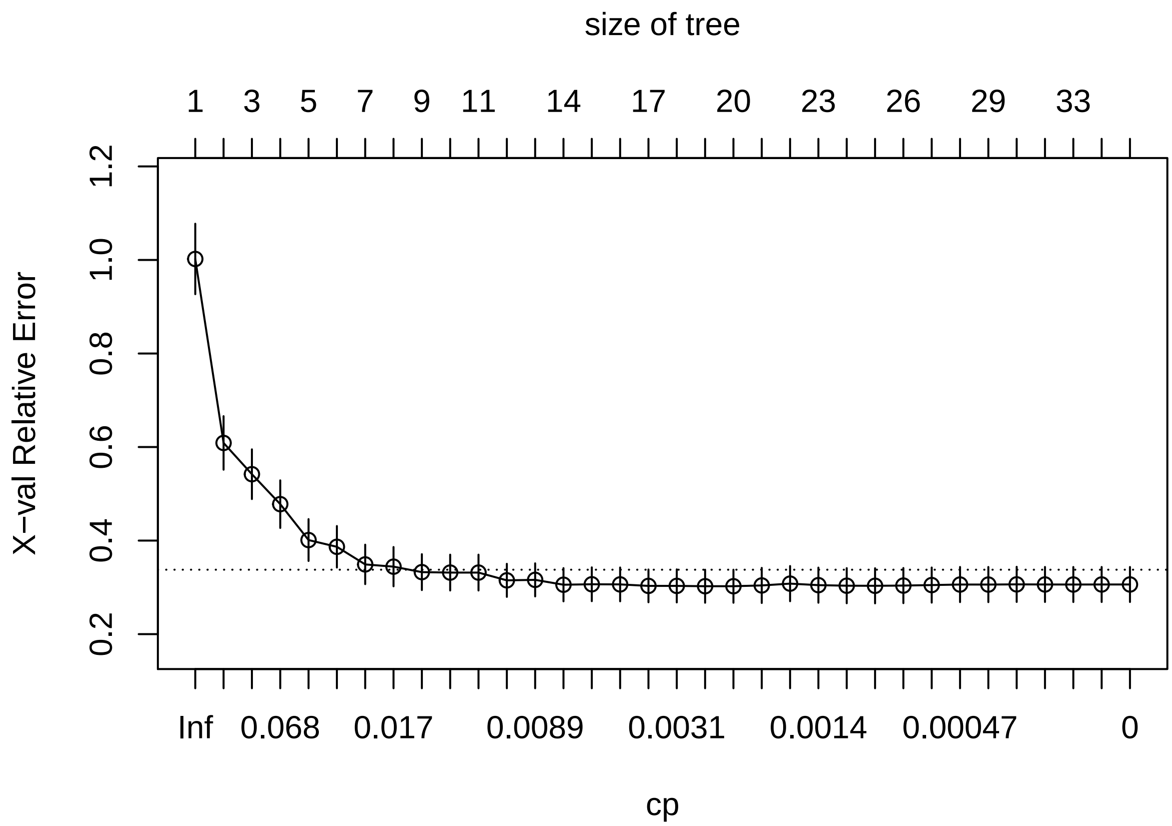

```
yhat = predict(m1, Xtest)
e = yhat - Ytest
sqrt(sum(e^2)/106)
```

```
## [1] 124680.6
```

```
m2 <- rpart(
    formula = Ytrain ~ .,
    data    = Xtrain,
    method  = "anova",
    control = list(cp = 0, xval = 10)
)
rpart.plot(m2)
```



```
plotcp(m2)
```




```r
yhat = predict(m2, Xtest)
e = yhat - Ytest
sqrt(sum(e^2)/106)
```

```
## [1] 107729.4
```

```r
jpeg(file = "save_m2.jpeg")
```

##Tuning

```r
m3 <- rpart(
    formula = Ytrain ~ .,
    data    = Xtrain,
    method  = "anova",
    control = list(minsplit = 10, maxdepth = 12, xval = 10)
)
yhat = predict(m3, Xtest)
e = yhat - Ytest
sqrt(sum(e^2)/106)
```

```
## [1] 124680.6
```

```r
m3$cptable
```

```
##            CP nsplit rel error    xerror       xstd
## 1  0.41178122      0 1.0000000 1.0034543 0.07589990
## 2  0.12693151      1 0.5882188 0.6643831 0.06115611
## 3  0.09283490      2 0.4612873 0.5923246 0.05845056
## 4  0.04979016      3 0.3684524 0.4999903 0.04805964
```

```
## 5  0.03740845      4 0.3186622 0.4470845 0.04635923
## 6  0.03384382      5 0.2812538 0.3857698 0.04100744
## 7  0.01825815      6 0.2474099 0.3802651 0.04042839
## 8  0.01556267      7 0.2291518 0.3792111 0.03781015
## 9  0.01250920      8 0.2135891 0.3636770 0.03611701
## 10 0.01226119      9 0.2010799 0.3586581 0.03924726
## 11 0.01000000     10 0.1888187 0.3423144 0.03858264
```

```r
# function to get optimal cp
get_cp <- function(x) {
  min    <- which.min(x$cptable[, "xerror"])
  cp <- x$cptable[min, "CP"]
}
# function to get minimum error
get_min_error <- function(x) {
  min    <- which.min(x$cptable[, "xerror"])
  xerror <- x$cptable[min, "xerror"]
}
```

```r
optimal_tree <- rpart(
    formula = Ytrain ~ .,
    data    = Xtrain,
    method  = "anova",
    control = list(minsplit = 11, maxdepth = 8, cp = 0.01)
    )
pred <- predict(optimal_tree, newdata = Xtrain)
RMSE(pred = pred, obs = Ytrain)
```

```
## [1] 76766.17
```

##RANDOM FOREST

```r
m1 <- randomForest(
  formula = Ytrain ~ .,
  data    = Xtrain
)
m1
```

```
##
## Call:
##  randomForest(formula = Ytrain ~ ., data = Xtrain)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 11
##
##           Mean of squared residuals: 5019464088
##                     % Var explained: 83.92
```

```r
which.min(m1$mse)
```

```
## [1] 281
```

```r
# RMSE of this optimal random forest
sqrt(m1$mse[which.min(m1$mse)])
```

```
## [1] 70680.71
```

```r
features <- setdiff(names(Xtrain), Ytrain)
set.seed(1989)
```

```r
m2 <- tuneRF(
  x          = Xtrain,
  y          = Ytrain,
  ntreeTry   = 500,
  mtryStart  = 5,
  stepFactor = 1.5,
  improve    = 0.01,
  trace      = FALSE      # to not show real-time progress
)
```

```
## -0.03117163 0.01
## 0.02312618 0.01
## 0.04316821 0.01
## -0.002202444 0.01
```