homework4_4

syh

June 1, 2017

```r
# in this section, we are gonna try to build random forest
# also, we would like to use mlr to find optimal parameters
```

```r
# read all data (train + prediction) without missing value
real_all_data <- read.csv(file = "H:/kaggle/houseprice/data/real_all_data_hybrid.csv",
                          stringsAsFactors = FALSE)[,-c(1,2)]
```

```r
# transform sale price to log sale price
real_all_data[,"SalePrice"] <- log(real_all_data[,"SalePrice"])
```

```r
# we would like to train a linear regression model with regulation
# 1. convert categorical ones to factors
for(i in 1:dim(real_all_data)[2]){
  if(is.character(real_all_data[,i])){
    real_all_data[,i] <- as.factor(real_all_data[,i])
  }
}
```

```r
# 1. split all data into train and prediction

model_data <- real_all_data[1:1460,]

pre_x <- real_all_data[-c(1:1460),]

# 2. split model data into train and test
set.seed(1000)
train_ind <- sample(1:dim(model_data)[1], size = dim(model_data)[1] * 0.7)

train_data <- model_data[train_ind,]
test_data <- model_data[-train_ind,]
```

```r
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```r
library(mlr)
```

```
## Loading required package: ParamHelpers
```

```r
#create a task
traintask <- makeRegrTask(data = train_data,target = "SalePrice")
```

```
## Warning in makeTask(type = type, data = data, weights = weights,
## blocking = blocking, : Empty factor levels were dropped for columns:
## RoofMatl,Exterior1st,Exterior2nd,HeatingQC,Functional
```

```r
testtask <- makeRegrTask(data = test_data,target = "SalePrice")
```

```
## Warning in makeTask(type = type, data = data, weights = weights,
## blocking = blocking, : Empty factor levels were dropped for columns:
```

```
## Utilities,Condition2,RoofStyle,RoofMatl,Exterior1st,Exterior2nd,ExterCond,BsmtCond,Heating,Electrical
```

```
#create learner
rf_lrn <- makeLearner("regr.randomForest",predict.type = "response")
rf_lrn$par.vals <- list(ntree = 100L, importance=TRUE)
```

```
#set 5 fold cross validation
rdesc <- makeResampleDesc("CV",iters=5L)

# Fit models according to a resampling strategy.
r <- resample(learner = rf_lrn, task = traintask, resampling = rdesc, measures = mse, show.info = T)
```

```
## [Resample] cross-validation iter 1:

## mse.test.mean=0.0218

## [Resample] cross-validation iter 2:

## mse.test.mean=0.0292

## [Resample] cross-validation iter 3:

## mse.test.mean=0.0203

## [Resample] cross-validation iter 4:

## mse.test.mean=0.0222

## [Resample] cross-validation iter 5:

## mse.test.mean=0.0149

## [Resample] Aggr. Result: mse.test.mean=0.0217
```

```
getParamSet(rf_lrn)
```

```
##                       Type  len       Def                 Constr Req
## ntree              integer    -       500               1 to Inf   -
## se.ntree           integer    -       100               1 to Inf   Y
## se.method         discrete    - jackknife bootstrap,jackknife,sd   Y
## se.boot            integer    -        50               1 to Inf   -
## mtry               integer    -         -               1 to Inf   -
## replace            logical    -      TRUE                      -   -
## strata             untyped    -         -                      -   -
## sampsize     integervector <NA>         -               1 to Inf   -
## nodesize           integer    -         5               1 to Inf   -
## maxnodes           integer    -         -               1 to Inf   -
## importance         logical    -     FALSE                      -   -
## localImp           logical    -     FALSE                      -   -
## nPerm              integer    -         1            -Inf to Inf   -
## proximity          logical    -     FALSE                      -   -
## oob.prox           logical    -         -                      -   Y
## do.trace           logical    -     FALSE                      -   -
## keep.forest        logical    -      TRUE                      -   -
## keep.inbag         logical    -     FALSE                      -   -
##              Tunable Trafo
## ntree           TRUE     -
## se.ntree        TRUE     -
## se.method       TRUE     -
## se.boot         TRUE     -
## mtry            TRUE     -
## replace        TRUE     -
## strata        FALSE     -
## sampsize       TRUE     -
## nodesize       TRUE     -
## maxnodes       TRUE     -
## importance     TRUE     -
## localImp       TRUE     -
## nPerm          TRUE     -
## proximity     FALSE     -
## oob.prox      FALSE     -
## do.trace      FALSE     -
## keep.forest   FALSE     -
## keep.inbag    FALSE     -
```

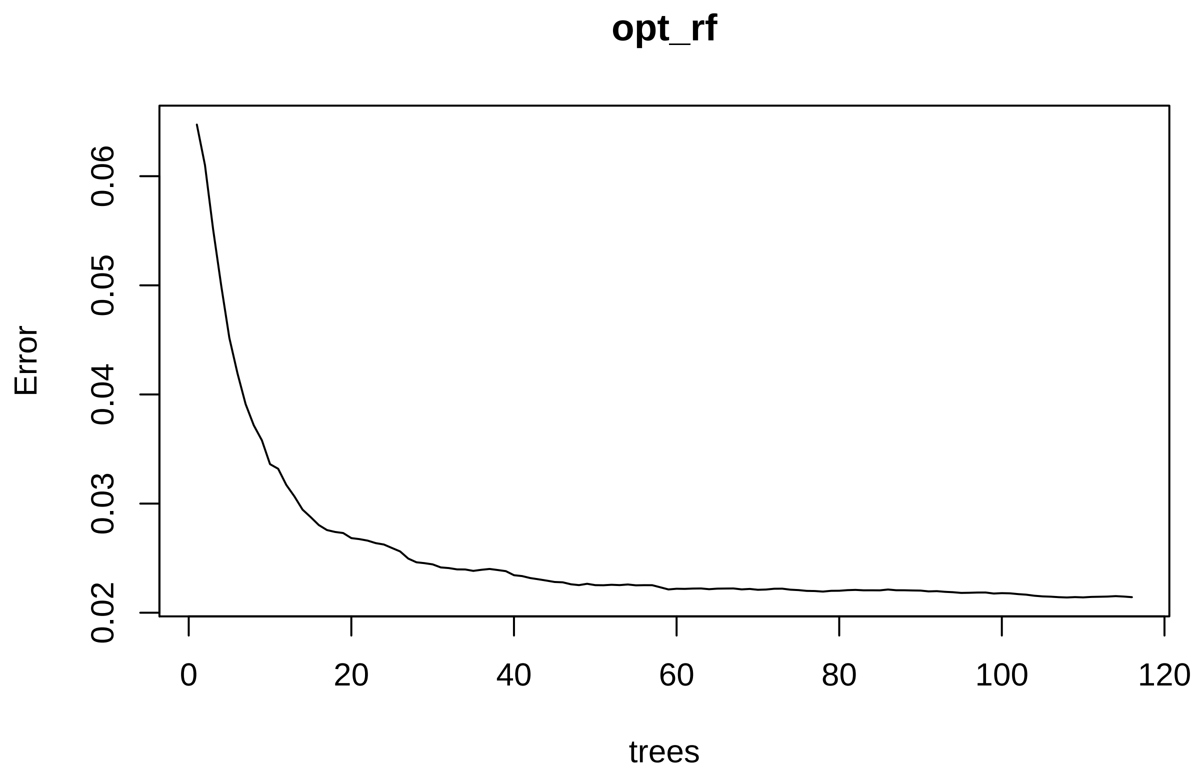
```
# define parameters we want to tune -- you may want to adjust the bounds
ps <- makeParamSet(
  makeIntegerLearnerParam(id = "ntree", default = 100L, lower = 80L, upper = 150L),
  makeIntegerLearnerParam(id = "nodesize", default = 15L, lower = 10L, upper = 50L),
  makeIntegerLearnerParam(id = "mtry", default = 8L, lower = 6, upper = 12L)
)

# random sampling of the configuration space with at most 100 samples
ctrl <- makeTuneControlRandom(maxit = 100L)


tune <- tuneParams(learner = rf_lrn, task = traintask, resampling = rdesc, measures = list(mse), par.se
```

```
## [Tune] Started tuning learner regr.randomForest for parameter set:

##              Type len Def    Constr Req Tunable Trafo
## ntree     integer   - 100 80 to 150   -    TRUE     -
## nodesize integer   -  15  10 to 50   -    TRUE     -
## mtry     integer   -   8   6 to 12   -    TRUE     -

## With control class: TuneControlRandom

## Imputation value: Inf

## [Tune-x] 1: ntree=133; nodesize=12; mtry=8

## [Tune-y] 1: mse.test.mean=0.023; time: 0.1 min

## [Tune-x] 2: ntree=137; nodesize=40; mtry=11

## [Tune-y] 2: mse.test.mean=0.0268; time: 0.0 min

## [Tune-x] 3: ntree=104; nodesize=23; mtry=12

## [Tune-y] 3: mse.test.mean=0.0238; time: 0.0 min

## [Tune-x] 4: ntree=102; nodesize=14; mtry=8

## [Tune-y] 4: mse.test.mean=0.0232; time: 0.0 min

## [Tune-x] 5: ntree=147; nodesize=50; mtry=11

## [Tune-y] 5: mse.test.mean=0.0282; time: 0.0 min

## [Tune-x] 6: ntree=139; nodesize=40; mtry=11

## [Tune-y] 6: mse.test.mean=0.0264; time: 0.0 min

## [Tune-x] 7: ntree=139; nodesize=20; mtry=9
## [Tune-y] 7: mse.test.mean=0.0238; time: 0.0 min
## [Tune-x] 8: ntree=96; nodesize=14; mtry=6
## [Tune-y] 8: mse.test.mean=0.0238; time: 0.0 min
## [Tune-x] 9: ntree=102; nodesize=28; mtry=8
## [Tune-y] 9: mse.test.mean=0.0254; time: 0.0 min
## [Tune-x] 10: ntree=146; nodesize=50; mtry=11
## [Tune-y] 10: mse.test.mean=0.0287; time: 0.0 min
## [Tune-x] 11: ntree=148; nodesize=20; mtry=8
## [Tune-y] 11: mse.test.mean=0.0242; time: 0.0 min
## [Tune-x] 12: ntree=143; nodesize=41; mtry=10
## [Tune-y] 12: mse.test.mean=0.0272; time: 0.0 min
## [Tune-x] 13: ntree=116; nodesize=37; mtry=8
## [Tune-y] 13: mse.test.mean=0.0272; time: 0.0 min
## [Tune-x] 14: ntree=136; nodesize=31; mtry=8
## [Tune-y] 14: mse.test.mean=0.026; time: 0.0 min
## [Tune-x] 15: ntree=103; nodesize=40; mtry=7
## [Tune-y] 15: mse.test.mean=0.0283; time: 0.0 min
## [Tune-x] 16: ntree=97; nodesize=28; mtry=8
## [Tune-y] 16: mse.test.mean=0.0254; time: 0.0 min
## [Tune-x] 17: ntree=138; nodesize=43; mtry=12
## [Tune-y] 17: mse.test.mean=0.0269; time: 0.0 min
## [Tune-x] 18: ntree=142; nodesize=10; mtry=10
## [Tune-y] 18: mse.test.mean=0.0225; time: 0.1 min
## [Tune-x] 19: ntree=110; nodesize=21; mtry=9
## [Tune-y] 19: mse.test.mean=0.024; time: 0.0 min
## [Tune-x] 20: ntree=144; nodesize=35; mtry=9
## [Tune-y] 20: mse.test.mean=0.0264; time: 0.0 min
## [Tune-x] 21: ntree=84; nodesize=44; mtry=10
## [Tune-y] 21: mse.test.mean=0.0277; time: 0.0 min
## [Tune-x] 22: ntree=89; nodesize=18; mtry=6
## [Tune-y] 22: mse.test.mean=0.0252; time: 0.0 min
## [Tune-x] 23: ntree=127; nodesize=15; mtry=12
## [Tune-y] 23: mse.test.mean=0.0222; time: 0.1 min
## [Tune-x] 24: ntree=133; nodesize=39; mtry=9
## [Tune-y] 24: mse.test.mean=0.0274; time: 0.0 min
## [Tune-x] 25: ntree=127; nodesize=43; mtry=10
## [Tune-y] 25: mse.test.mean=0.0278; time: 0.0 min
## [Tune-x] 26: ntree=134; nodesize=46; mtry=7
## [Tune-y] 26: mse.test.mean=0.0295; time: 0.0 min
## [Tune-x] 27: ntree=92; nodesize=31; mtry=10
## [Tune-y] 27: mse.test.mean=0.0257; time: 0.0 min
## [Tune-x] 28: ntree=113; nodesize=25; mtry=9
## [Tune-y] 28: mse.test.mean=0.0246; time: 0.0 min
## [Tune-x] 29: ntree=81; nodesize=26; mtry=6
## [Tune-y] 29: mse.test.mean=0.0269; time: 0.0 min
## [Tune-x] 30: ntree=104; nodesize=22; mtry=7
## [Tune-y] 30: mse.test.mean=0.0251; time: 0.0 min
## [Tune-x] 31: ntree=104; nodesize=16; mtry=12
## [Tune-y] 31: mse.test.mean=0.0223; time: 0.1 min
## [Tune-x] 32: ntree=109; nodesize=41; mtry=6
## [Tune-y] 32: mse.test.mean=0.0293; time: 0.0 min
## [Tune-x] 33: ntree=108; nodesize=23; mtry=10
## [Tune-y] 33: mse.test.mean=0.0241; time: 0.0 min
## [Tune-x] 34: ntree=144; nodesize=45; mtry=8
## [Tune-y] 34: mse.test.mean=0.029; time: 0.0 min
## [Tune-x] 35: ntree=149; nodesize=23; mtry=6
## [Tune-y] 35: mse.test.mean=0.0256; time: 0.0 min
## [Tune-x] 36: ntree=107; nodesize=45; mtry=11
## [Tune-y] 36: mse.test.mean=0.0277; time: 0.0 min
## [Tune-x] 37: ntree=118; nodesize=39; mtry=10
## [Tune-y] 37: mse.test.mean=0.0269; time: 0.0 min
## [Tune-x] 38: ntree=90; nodesize=49; mtry=10
## [Tune-y] 38: mse.test.mean=0.0283; time: 0.0 min
## [Tune-x] 39: ntree=129; nodesize=47; mtry=11
## [Tune-y] 39: mse.test.mean=0.0277; time: 0.0 min
## [Tune-x] 40: ntree=92; nodesize=20; mtry=8
## [Tune-y] 40: mse.test.mean=0.0249; time: 0.0 min
## [Tune-x] 41: ntree=82; nodesize=24; mtry=8
## [Tune-y] 41: mse.test.mean=0.025; time: 0.0 min
## [Tune-x] 42: ntree=88; nodesize=22; mtry=11
## [Tune-y] 42: mse.test.mean=0.0238; time: 0.0 min
## [Tune-x] 43: ntree=100; nodesize=43; mtry=9
## [Tune-y] 43: mse.test.mean=0.0279; time: 0.0 min

## [Tune-x] 44: ntree=136; nodesize=14; mtry=10

## [Tune-y] 44: mse.test.mean=0.0226; time: 0.1 min

## [Tune-x] 45: ntree=127; nodesize=31; mtry=6

## [Tune-y] 45: mse.test.mean=0.0279; time: 0.0 min

## [Tune-x] 46: ntree=145; nodesize=20; mtry=9

## [Tune-y] 46: mse.test.mean=0.0239; time: 0.1 min

## [Tune-x] 47: ntree=140; nodesize=24; mtry=8

## [Tune-y] 47: mse.test.mean=0.0251; time: 0.0 min

## [Tune-x] 48: ntree=93; nodesize=47; mtry=9

## [Tune-y] 48: mse.test.mean=0.0282; time: 0.0 min

## [Tune-x] 49: ntree=129; nodesize=43; mtry=10

## [Tune-y] 49: mse.test.mean=0.0272; time: 0.0 min

## [Tune-x] 50: ntree=142; nodesize=36; mtry=10

## [Tune-y] 50: mse.test.mean=0.026; time: 0.0 min

## [Tune-x] 51: ntree=88; nodesize=30; mtry=12

## [Tune-y] 51: mse.test.mean=0.0248; time: 0.0 min

## [Tune-x] 52: ntree=128; nodesize=40; mtry=8

## [Tune-y] 52: mse.test.mean=0.0281; time: 0.0 min

## [Tune-x] 53: ntree=112; nodesize=27; mtry=6

## [Tune-y] 53: mse.test.mean=0.0273; time: 0.0 min

## [Tune-x] 54: ntree=120; nodesize=36; mtry=12

## [Tune-y] 54: mse.test.mean=0.0261; time: 0.0 min

## [Tune-x] 55: ntree=108; nodesize=24; mtry=11

## [Tune-y] 55: mse.test.mean=0.0237; time: 0.0 min

## [Tune-x] 56: ntree=93; nodesize=47; mtry=7

## [Tune-y] 56: mse.test.mean=0.0296; time: 0.0 min

## [Tune-x] 57: ntree=150; nodesize=12; mtry=8

## [Tune-y] 57: mse.test.mean=0.0227; time: 0.1 min

## [Tune-x] 58: ntree=97; nodesize=23; mtry=7

## [Tune-y] 58: mse.test.mean=0.0245; time: 0.0 min

## [Tune-x] 59: ntree=119; nodesize=20; mtry=12

## [Tune-y] 59: mse.test.mean=0.0229; time: 0.1 min

## [Tune-x] 60: ntree=106; nodesize=23; mtry=7

## [Tune-y] 60: mse.test.mean=0.0254; time: 0.0 min

## [Tune-x] 61: ntree=129; nodesize=40; mtry=6
## [Tune-y] 61: mse.test.mean=0.0295; time: 0.0 min
## [Tune-x] 62: ntree=119; nodesize=20; mtry=11
## [Tune-y] 62: mse.test.mean=0.0232; time: 0.0 min
## [Tune-x] 63: ntree=105; nodesize=34; mtry=10
## [Tune-y] 63: mse.test.mean=0.0265; time: 0.0 min
## [Tune-x] 64: ntree=141; nodesize=15; mtry=10
## [Tune-y] 64: mse.test.mean=0.0227; time: 0.1 min
## [Tune-x] 65: ntree=136; nodesize=18; mtry=11
## [Tune-y] 65: mse.test.mean=0.0231; time: 0.1 min
## [Tune-x] 66: ntree=91; nodesize=17; mtry=12
## [Tune-y] 66: mse.test.mean=0.0229; time: 0.0 min
## [Tune-x] 67: ntree=130; nodesize=45; mtry=9
## [Tune-y] 67: mse.test.mean=0.0282; time: 0.0 min
## [Tune-x] 68: ntree=95; nodesize=43; mtry=6
## [Tune-y] 68: mse.test.mean=0.0301; time: 0.0 min
## [Tune-x] 69: ntree=121; nodesize=46; mtry=7
## [Tune-y] 69: mse.test.mean=0.03; time: 0.0 min
## [Tune-x] 70: ntree=140; nodesize=33; mtry=7
## [Tune-y] 70: mse.test.mean=0.0267; time: 0.0 min
## [Tune-x] 71: ntree=131; nodesize=28; mtry=7
## [Tune-y] 71: mse.test.mean=0.0264; time: 0.0 min
## [Tune-x] 72: ntree=119; nodesize=46; mtry=10
## [Tune-y] 72: mse.test.mean=0.0279; time: 0.0 min
## [Tune-x] 73: ntree=119; nodesize=29; mtry=9
## [Tune-y] 73: mse.test.mean=0.0256; time: 0.0 min
## [Tune-x] 74: ntree=92; nodesize=46; mtry=6
## [Tune-y] 74: mse.test.mean=0.0307; time: 0.0 min
## [Tune-x] 75: ntree=134; nodesize=33; mtry=11
## [Tune-y] 75: mse.test.mean=0.0256; time: 0.0 min
## [Tune-x] 76: ntree=137; nodesize=30; mtry=8
## [Tune-y] 76: mse.test.mean=0.026; time: 0.0 min
## [Tune-x] 77: ntree=145; nodesize=25; mtry=6
## [Tune-y] 77: mse.test.mean=0.026; time: 0.0 min
## [Tune-x] 78: ntree=102; nodesize=19; mtry=10
## [Tune-y] 78: mse.test.mean=0.0234; time: 0.0 min
## [Tune-x] 79: ntree=99; nodesize=48; mtry=7
## [Tune-y] 79: mse.test.mean=0.0301; time: 0.0 min
## [Tune-x] 80: ntree=111; nodesize=25; mtry=8
## [Tune-y] 80: mse.test.mean=0.0255; time: 0.0 min
## [Tune-x] 81: ntree=125; nodesize=43; mtry=10
## [Tune-y] 81: mse.test.mean=0.0276; time: 0.0 min
## [Tune-x] 82: ntree=106; nodesize=17; mtry=9
## [Tune-y] 82: mse.test.mean=0.0234; time: 0.0 min
## [Tune-x] 83: ntree=90; nodesize=24; mtry=10
## [Tune-y] 83: mse.test.mean=0.025; time: 0.0 min
## [Tune-x] 84: ntree=121; nodesize=31; mtry=11
## [Tune-y] 84: mse.test.mean=0.0253; time: 0.0 min
## [Tune-x] 85: ntree=92; nodesize=19; mtry=12
## [Tune-y] 85: mse.test.mean=0.0231; time: 0.0 min
## [Tune-x] 86: ntree=119; nodesize=20; mtry=10
## [Tune-y] 86: mse.test.mean=0.0233; time: 0.0 min
## [Tune-x] 87: ntree=110; nodesize=31; mtry=11
## [Tune-y] 87: mse.test.mean=0.025; time: 0.0 min
## [Tune-x] 88: ntree=121; nodesize=17; mtry=6
## [Tune-y] 88: mse.test.mean=0.0244; time: 0.0 min
## [Tune-x] 89: ntree=113; nodesize=50; mtry=7
## [Tune-y] 89: mse.test.mean=0.0303; time: 0.0 min
## [Tune-x] 90: ntree=105; nodesize=35; mtry=8
## [Tune-y] 90: mse.test.mean=0.0271; time: 0.0 min
## [Tune-x] 91: ntree=127; nodesize=48; mtry=6
## [Tune-y] 91: mse.test.mean=0.0309; time: 0.0 min
## [Tune-x] 92: ntree=125; nodesize=30; mtry=10
## [Tune-y] 92: mse.test.mean=0.0254; time: 0.0 min
## [Tune-x] 93: ntree=134; nodesize=44; mtry=12
## [Tune-y] 93: mse.test.mean=0.0268; time: 0.0 min
## [Tune-x] 94: ntree=82; nodesize=13; mtry=6
## [Tune-y] 94: mse.test.mean=0.0234; time: 0.0 min
## [Tune-x] 95: ntree=135; nodesize=10; mtry=10
## [Tune-y] 95: mse.test.mean=0.0223; time: 0.1 min
## [Tune-x] 96: ntree=102; nodesize=35; mtry=7
## [Tune-y] 96: mse.test.mean=0.0277; time: 0.0 min
## [Tune-x] 97: ntree=80; nodesize=42; mtry=10
## [Tune-y] 97: mse.test.mean=0.0274; time: 0.0 min

## [Tune-x] 98: ntree=86; nodesize=28; mtry=9

## [Tune-y] 98: mse.test.mean=0.0258; time: 0.0 min

## [Tune-x] 99: ntree=106; nodesize=29; mtry=11

## [Tune-y] 99: mse.test.mean=0.0251; time: 0.0 min

## [Tune-x] 100: ntree=93; nodesize=43; mtry=8

## [Tune-y] 100: mse.test.mean=0.0289; time: 0.0 min

## [Tune] Result: ntree=127; nodesize=15; mtry=12 : mse.test.mean=0.0222
```

```
# the mse is little better than previous
```

```
set.seed(2)
formula <- "SalePrice ~.-SalePrice"
opt_rf <- randomForest(as.formula(formula),data = train_data, ntree = 116,
                       mtry = 11, nodesize = 10, importance = T)
```

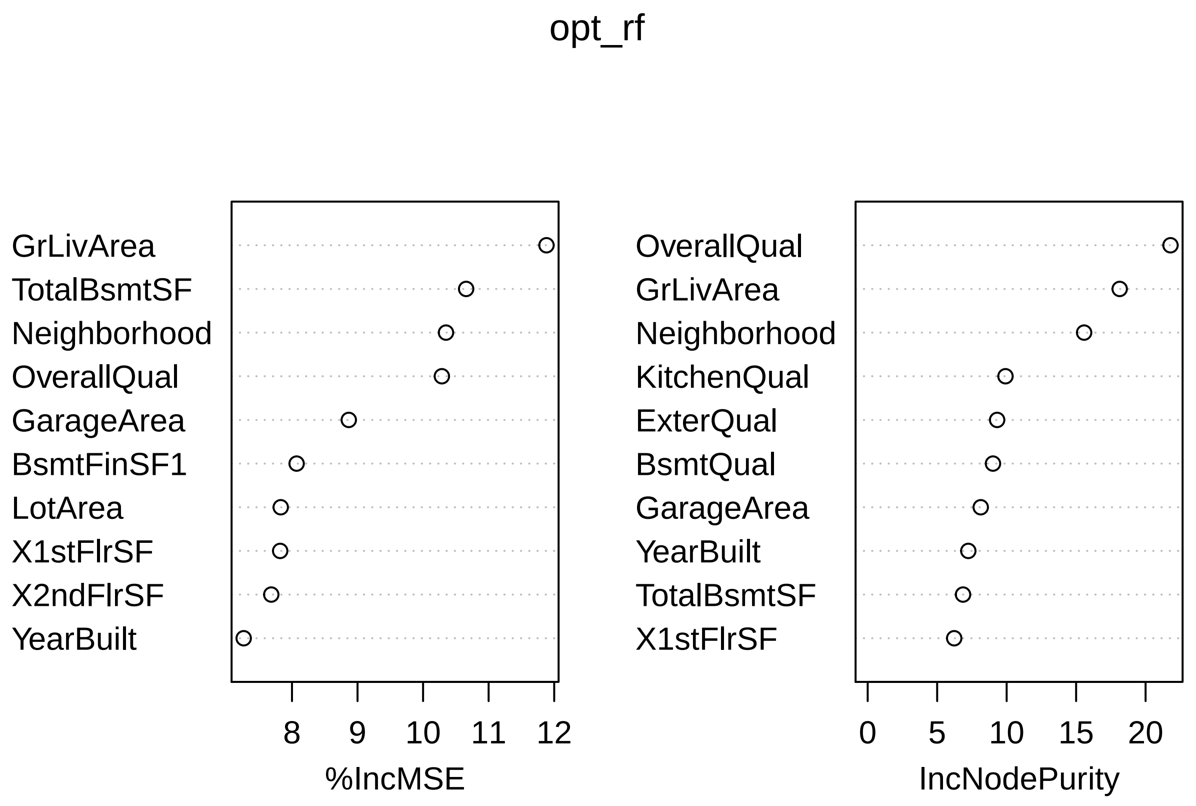
```
plot(opt_rf)
```



```
imp_ord <- order(-opt_rf$importance[, "%IncMSE"])
opt_rf$importance[imp_ord,]
```

```
##                      %IncMSE IncNodePurity
## GrLivArea       2.367863e-02  18.136640329
## OverallQual     2.122303e-02  21.792647864
## Neighborhood   1.502647e-02  15.575687853
## TotalBsmtSF    8.663080e-03   6.860633671
## X1stFlrSF      8.360224e-03   6.225175140
## YearBuilt      7.393279e-03   7.241520394
## ExterQual      6.649120e-03   9.316029787
## KitchenQual    6.572209e-03   9.916480530
## GarageArea     6.407549e-03   8.134457552
## BsmtQual       5.707335e-03   9.016202758
## X2ndFlrSF      5.376383e-03   3.479254853
## GarageCars     5.364011e-03   6.200513172
## Fireplaces     2.962712e-03   3.394502213
## LotArea        2.941782e-03   3.346094132
## FullBath       2.719500e-03   3.697792303
## BsmtFinSF1     2.544189e-03   2.604917576
## YearRemodAdd   2.463571e-03   2.877003998
## TotRmsAbvGrd   1.884634e-03   1.636918392
## OverallCond    1.565411e-03   1.266795974
## BedroomAbvGr   1.084434e-03   0.851978523
## HouseStyle     1.071423e-03   0.729866460
## Exterior2nd    9.990248e-04   1.282109338
## MSSubClass     9.666413e-04   0.661279141
## Exterior1st    8.583689e-04   1.302554346
## CentralAir     8.535730e-04   1.549511264
## BsmtFinType1   8.528545e-04   0.950507473
## MasVnrArea     7.536152e-04   1.151923016
## MSZoning       6.873372e-04   0.906121875
## BsmtUnfSF      6.707278e-04   0.780359242
## OpenPorchSF    6.413831e-04   1.119735406
## HalfBath       6.144555e-04   0.546415119
## HeatingQC      6.040683e-04   0.661758105
## Foundation     4.362618e-04   0.895889578
## BldgType       4.360713e-04   0.336348949
## Condition1     4.294459e-04   0.294949077
## WoodDeckSF     4.153729e-04   0.596105903
## BsmtExposure   3.681887e-04   0.664899128
## RoofStyle      3.311654e-04   0.551652606
## MasVnrType     2.858594e-04   0.410426134
## BsmtCond       2.745713e-04   0.417485981
## BsmtFullBath   2.681217e-04   0.189436631
## LotShape       2.216560e-04   0.257571680
## LandContour    2.172331e-04   0.327661364
## SaleCondition  1.767370e-04   0.410283566
## KitchenAbvGr   1.719637e-04   0.114948595
## MoSold         1.683455e-04   0.445702443
## PavedDrive     1.460764e-04   0.231625526
## Functional     1.107583e-04   0.298146648
## SaleType       1.027071e-04   0.227532814
## ScreenPorch    1.005317e-04   0.150364793
## BsmtFinSF2     7.437427e-05   0.105493972
## BsmtFinType2   5.490209e-05   0.180245561
## LandSlope      4.622693e-05   0.170645795
## Heating        3.080981e-05   0.148586793
## BsmtHalfBath   1.922503e-05   0.030838400
## Electrical     1.719824e-05   0.251272369
## MiscVal        1.710765e-05   0.046181512
## YrSold         8.011551e-06   0.226252874
## LotConfig      1.486484e-06   0.164410402
## Street         0.000000e+00   0.000000000
## Utilities      0.000000e+00   0.001792195
## PoolArea      -3.890612e-06   0.012325738
## Condition2    -6.728372e-06   0.022594986
## ExterCond     -8.941454e-06   0.211854718
## X3SsnPorch    -1.338779e-05   0.018151780
## LowQualFinSF  -1.468969e-05   0.043858821
## EnclosedPorch -4.194602e-05   0.144253431
## RoofMatl      -5.161235e-05   0.177983977
```

```r
varImpPlot(x = opt_rf, sort = T, n.var = 10)
```



```r
# make estimate on test data
est_test_sp <- predict(object = opt_rf, newdata = test_data)
mean((est_test_sp - test_data[,"SalePrice"]) ^ 2)
```

```
## [1] 0.01614393
```

```r
# make prediction
pre_sale_price <- predict(object = opt_rf, newdata = pre_x)
result <- data.frame(Id = c(1461:2919), SalePrice = exp(pre_sale_price))
write.csv(x = result, file = "H:/kaggle/houseprice/data/submission_6.csv",
          row.names = FALSE)
# Your submission scored 0.14541, not good.
```