# mlr3 Usecase Day 2 - Ames Housing Dataset

This exercise is intended to use `mlr3tuning` to improve the results of the benchmark analysis on the *Ames Housing Dataset* of day 1.

The main objective of this exercise is as follows:

- To apply an appropriate tuning technique to a learning algorithm in order to improve its predictive performance.

Keep in mind that you are asked to use the Mean Absolute Error (MAE) as a performance measure to evaluate the performance of a tuned algorithm.

## Accessing the dataset

The dataset is available on Kaggle http://www.kaggle.com/c/ames-day2. Kaggle is a platform which provides data science competitions and datasets which can be used to get familiar with typical machine learning methods.

## Importing the data

```
housing = read.csv("data/ames_housing_train_numeric.csv")
```

1. Load the `mlr3` and `mlr3learners` packages.

```
library(mlr3)
library(mlr3learners)
```

2. Create a regression task object.

```
task = TaskRegr$new(id = "ames_housing", backend = housing, target = "SalePrice")
task
```

```
## <TaskRegr:ames_housing> (1953 x 38)
## * Target: SalePrice
## * Properties: -
## * Features (37):
##   - int (37): Bedroom.AbvGr, Bsmt.Full.Bath, Bsmt.Half.Bath,
##     Bsmt.Unf.SF, BsmtFin.SF.1, BsmtFin.SF.2, Enclosed.Porch,
##     Fireplaces, Full.Bath, Garage.Area, Garage.Cars,
##     Garage.Yr.Blt, Gr.Liv.Area, Half.Bath, Kitchen.AbvGr,
##     Lot.Area, Lot.Frontage, Low.Qual.Fin.SF, MS.SubClass,
##     Mas.Vnr.Area, Misc.Val, Mo.Sold, Open.Porch.SF, Overall.Cond,
##     Overall.Qual, PID, Pool.Area, Screen.Porch, TotRms.AbvGrd,
##     Total.Bsmt.SF, Wood.Deck.SF, X1st.Flr.SF, X2nd.Flr.SF,
##     X3Ssn.Porch, Year.Built, Year.Remod.Add, Yr.Sold
```

3. Create a list of learning algorithms and their parameter sets which we want to tune.

```
# get a featureless learner and a regression tree
library(mlr3learners)
knn = lrn("regr.kknn")
xgboost = lrn("regr.xgboost", nrounds = 100L)
```

```
library(paradox)
knn_tune_ps = ParamSet$new(list(
ParamDbl$new("log_k", log(1), log(100))
```

```
))
knn_tune_ps$trafo = function(x, param_set) {
return(list(k = round(exp(x$log_k))))
}
knn_tune_ps
```

```
## ParamSet:
##       id    class lower upper levels     default value
## 1: log_k ParamDbl     0 4.605        <NoDefault>
## Trafo is set.
```

```
xgboost_tune_ps = ParamSet$new(list(
  ParamDbl$new("eta", lower = -7, upper = 0),
  ParamDbl$new("gamma", lower = -5, upper = 6)
))
xgboost_tune_ps$trafo = function(x, param_set) {
return(list(eta = 2^(x$eta), gamma = 2^(x$gamma)))
}
xgboost_tune_ps
```

```
## ParamSet:
##       id    class lower upper levels     default value
## 1:   eta ParamDbl    -7     0        <NoDefault>
## 2: gamma ParamDbl    -5     6        <NoDefault>
## Trafo is set.
```

4. Define all necessary tuning settings.

```
library(mlr3tuning)
# we want to use an inner 3-fold cross validation
inner_resampling = rsmp("cv", folds = 3) # inner resampling
terminator = term("model_time", secs = 60L) # set terminator to 60 seconds
measure = msr("regr.mae") # we want to optimize the MAE
tuner = tnr("grid_search", resolution = 10) # we use grid search here
```

5. Let's use the `autotuner` to use the tuned learner as a regular one within our previous benchmark

```
knn_tuned = AutoTuner$new(
  learner = knn,
  resampling = inner_resampling,
  measures = measure,
  tune_ps = knn_tune_ps,
  terminator = terminator,
  tuner = tuner
)
knn_tuned
```

```
## <AutoTuner:regr.kknn.tuned>
## * Model: -
## * Parameters: list()
## * Packages: withr, kknn
## * Predict Type: response
## * Feature types: logical, integer, numeric, factor, ordered
## * Properties: -
```

```
xgboost_tuned = AutoTuner$new(
  learner = xgboost,
```

```
  resampling = inner_resampling,
  measures = measure,
  tune_ps = xgboost_tune_ps,
  terminator = terminator,
  tuner = tuner
)
xgboost_tuned
```

```
## <AutoTuner:regr.xgboost.tuned>
## * Model: -
## * Parameters: nrounds=100, verbose=0
## * Packages: xgboost
## * Predict Type: response
## * Feature types: integer, numeric
## * Properties: importance, missings, weights
```

5. Create a grid corresponding to the planned benchmark including the task, all learners and the resampling strategy.

```
# create a BenchmarkDesign object
learners = list(
  featureless = lrn("regr.featureless"),
  lm = lrn("regr.lm"),
  knn = lrn("regr.kknn"),
  knn_tuned = knn_tuned,
  tree = lrn("regr.rpart"),
  random_forest = lrn("regr.ranger"),
  xgboost = xgboost,
  xgboost_tuned = xgboost_tuned
)
resampling = rsmp("cv", folds = 5)
design = benchmark_grid(task, learners, resampling)
print(design)
```

```
##          task                  learner     resampling
## 1: <TaskRegr> <LearnerRegrFeatureless> <ResamplingCV>
## 2: <TaskRegr>          <LearnerRegrLM> <ResamplingCV>
## 3: <TaskRegr>        <LearnerRegrKKNN> <ResamplingCV>
## 4: <TaskRegr>              <AutoTuner> <ResamplingCV>
## 5: <TaskRegr>       <LearnerRegrRpart> <ResamplingCV>
## 6: <TaskRegr>      <LearnerRegrRanger> <ResamplingCV>
## 7: <TaskRegr>     <LearnerRegrXgboost> <ResamplingCV>
## 8: <TaskRegr>              <AutoTuner> <ResamplingCV>
```

6. Run the benchmark

```
# run the benchmark
bmr = benchmark(design)
```

```
## Warning in predict.lm(self$model, newdata = newdata, se.fit = FALSE):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(self$model, newdata = newdata, se.fit = FALSE):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(self$model, newdata = newdata, se.fit = FALSE):
```

```
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(self$model, newdata = newdata, se.fit = FALSE):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(self$model, newdata = newdata, se.fit = FALSE):
## prediction from a rank-deficient fit may be misleading
```

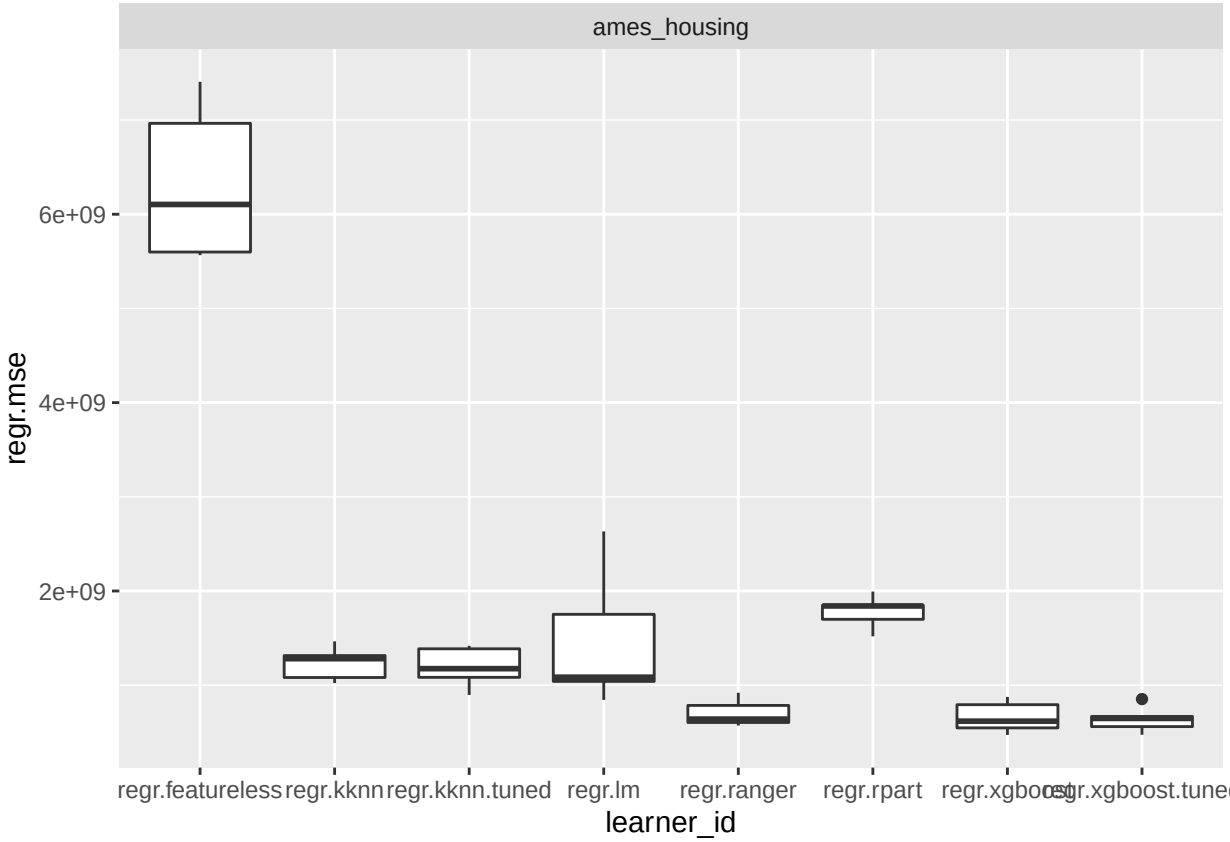
7. Use appropriate regression measures to measure the performance of each learner in the benchmark.

```
# get some measures: accuracy (acc) and area under the curve (auc)
measures = mlr_measures$mget(c("regr.mae", "regr.mse"))
bmr$aggregate(measures)
```

```
##    nr  resample_result      task_id         learner_id resampling_id iters
## 1:  1 <ResampleResult> ames_housing   regr.featureless            cv     5
## 2:  2 <ResampleResult> ames_housing            regr.lm            cv     5
## 3:  3 <ResampleResult> ames_housing          regr.kknn            cv     5
## 4:  4 <ResampleResult> ames_housing    regr.kknn.tuned            cv     5
## 5:  5 <ResampleResult> ames_housing         regr.rpart            cv     5
## 6:  6 <ResampleResult> ames_housing        regr.ranger            cv     5
## 7:  7 <ResampleResult> ames_housing       regr.xgboost            cv     5
## 8:  8 <ResampleResult> ames_housing regr.xgboost.tuned            cv     5
##    regr.mae   regr.mse
## 1:    58320 6327565483
## 2:    21571 1470694095
## 3:    20610 1232960026
## 4:    20120 1192011989
## 5:    28535 1780220260
## 6:    16107  703882045
## 7:    16515  661019668
## 8:    15671  640249517
```

8. Use an appropriate plot to illustrate the benchmark results. Here, we use the `mlr3viz` package.

```
library(mlr3viz)
autoplot(bmr)
```




9. We take the best performing algorithm - the xgboost autotuner - as final learner to predict on the test data.

```
test_set = read.csv("data/ames_housing_test_numeric.csv")
final_learner = learners$xgboost_tuned
final_learner$train(task)
pred = final_learner$predict_newdata(task, test_set)

# we can save the predictions as data.table and export them for Kaggle
pred = as.data.table(pred)
pred$truth = NULL
write.csv(pred, "data/ames_housing_submission_day2.csv", row.names = FALSE)
```