Module 9: Recommended Exercises

TMA4268 Statistical Learning V2024

Kenneth Aase, Daesoo Lee, Sara Martino, Stefanie Muff
Department of Mathematical Sciences, NTNU

March 14, 2024

We recommend you to work through the Section 8.3.4 in the course book (Lab: Boosting)

Problem 1

Both **bagging** and **boosting** are *ensemble* methods.

- What is an ensemble method?
- What are the main conceptual differences between bagging and boosting?

Problem 2

Consider the Gradient Tree Boosting Algorithm (given on slide 35 in this week's lectures).

Explain the meaning behind each of the following symbols. If relevant, also explain what the impact of changing them will be, and explain how one might find or choose them.

- $L(.)$
- $f_m(.)$
- M
- J_m

Problem 3 - Learning rate

The algorithm mentioned in Problem 2 does not include the so-called *learning rate* parameter ν. In the context of boosting methods, what is the interpretation behind this parameter, and how would one choose it? If you were to include the parameter in the Gradient Tree Boosting Algorithm, which equation(s) in the algorithm would you change and how?

Problem 4 - Boosting methods

In the code chunk below we simulate data meant to mimic a genomic data set for a set of markers (locations in the human genome) that underlie a complex trait (such as height, or whether you have cancer or not). You don't need to understand the details of the simulation, but the main takeaways are

- y is a continuous response, representing some biological trait (for example height) depending on both genetic (the predictors) and environmental (here included as random noise) factors.
- We have many more predictors (genetic markers) M than observations (individuals) N.
- The predictors have a complicated structure: most of them have a very small effect on y, but a few of them have a larger effect.

```
# install.packages("DirichletReg")
# install.packages("magrittr")
library(DirichletReg)
library(magrittr)

# Simulate
set.seed(1)
N <- 1000 # number of individuals
M <- 10000 # number of genotyped markers
p <- rbeta(n = M, shape1 = 2.3, shape2 = 7.4) # Allele frequencies
p %<>% ifelse(. < 0.5, ., 1 - .)
X <- sapply(p, rbinom, n = N, size = 2) # genotype matrix

effect_weights <- rdirichlet(M, c(0.9, 0.08, 0.015, 0.005))

marker_effects <- apply(effect_weights, 1, function(w) {
  effects <- c(0, sapply(sqrt(c(1e-4, 1e-3, 1e-2)), rnorm, n = 1, mean = 0))
  sum(w * effects)
})

genetics <- as.vector(X %*% marker_effects)
environment <- rnorm(N, sd = sqrt(var(genetics) / 2))
y <- genetics + environment
dat <- cbind(y, as.data.frame(X))

# Looking at a few entries in the data:
dat[1:6,1:20]
```

We will now try to predict y using various tree boosting methods, adapted from code provided in https://bradleyboehmke.github.io/HOML/gbm.html.

a) A basic GBM

We first implement a standard gradient boosting tree using the package `gbm`. To make sure everything works, we run a very small ($M = 3$) model below.

```
# install.packages("dplyr")
library(dplyr)    # for general data wrangling needs

# Modeling packages
# install.packages("gbm")
# install.packages("h2o")
# install.packages("xgboost")
library(gbm)      # for original implementation of regular and stochastic GBMs
library(h2o)      # for a java-based implementation of GBM variants
library(xgboost)  # for fitting extreme gradient boosting

gbm1 <- gbm(
  formula = y ~ .,
  data = dat,
  distribution = "gaussian",  # SSE loss function
  n.trees = 3,
  shrinkage = 0.1,
  interaction.depth = 7,
```

```r
  n.minobsinnode = 10,
  cv.folds = 10
)
```

We note that running the above code block is very slow - even for a *very* low number of trees $M = 3$ (M is typically in the thousands). How could we modify the algorithm to be less computationally intensive?

b) Stochastic GBMs

We now move on to stochastic gradient boosting tree models for the same data set, as implemented in the `h2o` package. Explain what is done in the below code (`?h2o.grid`, `h2o.gbm` and https://bradleyboehmke.github.io/HOML/gbm.html might be helpful).

```r
# Set maximum memory usage
h2o.init(max_mem_size = "14g")

# refined hyperparameter grid
hyper_grid <- list(
  sample_rate = c(0.2, 0.5),              # row subsampling
  col_sample_rate = 0.1,          # col subsampling for each split
  col_sample_rate_per_tree = 0.1,  # col subsampling for each tree
  min_rows = 10,
  learn_rate = 0.05,
  max_depth = c(3, 5, 7),
  ntrees = 10000
)

# random grid search strategy
search_criteria <- list(
  strategy = "RandomDiscrete",
  stopping_metric = "mse",
  stopping_tolerance = 0.001,
  stopping_rounds = 10,
  max_runtime_secs = 20*60
)

# perform grid search
grid <- h2o.grid(
  algorithm = "gbm",
  y = "y",
  training_frame = as.h2o(dat),
  hyper_params = hyper_grid,
  nfolds = 5,
  search_criteria = search_criteria,
  seed = 123,
  parallelism = 0
)

# collect the results and sort by our model performance metric of choice
grid_perf <- h2o.getGrid(
  grid_id = grid@grid_id,
  sort_by = "mse",
  decreasing = FALSE
)
```

```
grid_perf
```

c) XGboost

Below we also provide code for applying cross-validated XGBoost to the same data set. Expand this code to perform a search of the hyperparameter space, similar to b).

```
set.seed(123)
xgb <- xgb.cv(
  data = X,
  label = y,
  nrounds = 6000,
  early_stopping_rounds = 50,
  nfold = 5,
  params = list(
    eta = 0.05,
    max_depth = 3,
    min_child_weight = 3,
    subsample = 0.2,
    colsample_bytree = 0.1),
  verbose = TRUE
)

# minimum test CV RMSE
min(xgb$evaluation_log$test_rmse_mean)
```

d) Model evaluation

Finally, experimenting with the code from a), b) and c), what is the best model you are able to find? Fit that model using all available training data (no cross-validation).