Machine Learning Extra Assignment

Agustin Valencia

Assignment 1. Spam classification with random LASSO

LASSO classifiers are efficient in regression in classification but have some undesired properties. First, there is no good way of extracting importance of each feature. In addition, when there are highly correlated variables that have a relationship to the target, only one of these variables may be picked up by LASSO. This has very negative consequences in many applications. For instance, when classifying between healthy and sick subjects based on gene expressions, only some influential genes are selected but many other important ones are missed.

Random LASSO is one of proposed solutions that eliminates some disadvantages of LASSO: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3445423/ In this assignment, you will need to use spambase data from Lab 1 where Spam is target and all other columns are features.

1. Implement random LASSO for spambase data by using glmnet package functionality, in each bootstrap step compute LASSO by cross-validation with 4 folds and λ corresponding to the smallest CV score. Write `set.seed(12345)` before each bootstrap loop. Use the following settings in the random LASSO: $q_1 = q_2 = round(p/2)$, where p is the amount of features, $B = 100$ (hint: when debugging your code, use a much smaller B). Extract 10 features having the highest importance and make a bar chart showing importance values of these features. Finally, print a confusion matrix obtained from predicting the training data by the random LASSO.

    a. The report should contain: a) Bar chart requested b) Confusion matrix requested.

Assignment 2. Model-Based decision trees

Traditional decision trees are sometimes criticized because they fit a constant term in each terminal node which makes the resulting trees to be piecewise constant functions in a regression setting. A remedy in this case are model-based trees that allow a user to fit any model within each terminal node:

https://amstat.tandfonline.com/doi/abs/10.1198/106186008X319331#.Xc6rFW5FxaQ

1. Use R package for model-based trees `partykit` and function `mob()` in order to implement decision trees in which in each node you fit a linear regression including all features and all their possible interactions (which therefore becomes a nonlinear regression)
    a. Hint: use something like "Y~.^2" in lm() function
    b. Can be useful to check `vignette("mob", package = "partykit")`
2. Fit the implemented model to Women.csv data in which Blood.Systolic is the target and Weight and Height are variables that are used as possible splitting variables in the decision tree and also as features in the embedded nonlinear regressions. Set 5000 to be the smallest possible number of observations in a tree node. Plot the resulting tree showing regression coefficients in each terminal node.
3. Create a grid of values for Height and Weight that covers the data points and that has 100 levels in each dimension – Height and Weight. Compute predictions from the model for each combination of Height and Weight from the grid and use the result in order to create a raster plot using geom_raster() in ggplot2 (color intensity should correspond to the predicted Blood.Systolic).
    a. Hint: check carefully "type" parameter of "predict" function.
4. Include the decision tree and the raster plot in your report.

Assignment 3. Neural networks

Implement the backpropagation algorithm for fitting the parameters of a NN for regression. The NN has one input unit, 10 hidden units, and one output unit. Use the tanh activation function. Recall that you have an example in Bishop's book as well as in the course slides. Feel free to use stochastic or batch gradient descent. Please use only basic R functions in your solution. Please comment your code appropriately. Please use the following template.

```r
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin = sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
# plot(trva)
# plot(tr)
# plot(va)
w_j <- runif(10, -1, 1)
b_j <- runif(10, -1, 1)
w_k <- runif(10, -1, 1)
b_k <- runif(1, -1, 1)
l_rate <- 1 / nrow(tr) ^ 2
n_ite = 5000
error <- rep(0, n_ite)
error_va <- rep(0, n_ite)
for (i in 1:n_ite) {
    # error computation: Your code here
    cat("i: ", i, ", error: ", error[i] / 2, ", error_va: ", error_va[i] / 2, "\n")
    flush.console()
    for (n in 1:nrow(tr)) {
        # forward propagation: Your code here
        # backward propagation: Your code here
    }
}
# print final weights and errors
w_j
b_j
w_k
b_k
plot(error / 2, ylim = c(0, 5))
points(error_va / 2, col = "red")
# plot prediction on training data
pred <- matrix(nrow = nrow(tr), ncol = 2)
for (n in 1:nrow(tr)) {
    z_j <-
    y_k <-
    pred[n,] <- c(tr[n,]$Var, y_k)
}
plot(pred)
points(tr, col = "red")
# plot prediction on validation data
pred <- matrix(nrow = nrow(tr), ncol = 2)
for (n in 1:nrow(va)) {
    z_j <-
    y_k <-
    pred[n,] <- c(va[n,]$Var, y_k)
}
plot(pred)
points(va, col = "red")
```