Machine Learning – I

Ramakrishna Mission Vivekananda University

Assignment 1

Posted on 16 Feb 2017 | Seek clarification by 21 Feb 2017 | Submit by 28 Feb 2017

Your solutions for the problems in this assignment should be in a single `R` file `GroupXX_a1.R` or in a single Python file `GroupXX_a1.py`, where `XX` is your group number (i.e., `XX` = 01 to 10). The code must be properly commented. The solution file should be emailed to `sg.sourav@gmail.com`, with a copy to all your group members, *before* the lecture session on the submission deadline.

Properly acknowledge every source of information that you refer to, including discussions with your friends outside the group. Verbatim copy from any source is strongly discouraged, and plagiarism will be heavily penalized. You should try to write the codes completely on your own.

Problem 1 [20 points]

Background: Given an n-sample-p-feature training set $\{x_1^{(i)}, x_2^{(i)}, \ldots, x_p^{(i)}, y^{(i)}\}$ for $i = 1, \ldots, n$, one may assume an initial linear relation between the features $\{x_1, x_2, \ldots, x_p\}$ and y as follows:

$$y \;\approx\; h_\beta(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p,$$

and try to learn the optimal values of the relational coefficients $\beta = \{\beta_0, \beta_1, \ldots, \beta_p\}$ using an *ordinary least squares* model for linear regression, which minimizes the following cost function:

$$J(\beta) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_\beta(x^{(i)}) - y^{(i)}\right)^2 = \frac{1}{2n}\sum_{i=1}^{n}\left(\beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \cdots + \beta_p x_p^{(i)} - y^{(i)}\right)^2.$$

Batch Gradient Descent is a well-known representative for a broad class of iterative algorithms for minimizing $J(\beta)$, based on the least mean squares (LMS) update rule, also known as the Widrow-Hoff learning rule, with the parameter α controlling the learning rate, as follows:

$$\beta_j \;\leftarrow\; \beta_j - \alpha\frac{\partial J}{\partial \beta_j} = \beta_j - \alpha \cdot \frac{1}{n}\sum_{i=1}^{n}\left(h_\beta(x^{(i)}) - y^{(i)}\right) x_j^{(i)} \qquad \text{for each } j = 0, 1, \ldots, p.$$

Problem statement: Write an `R` (or a Python) function for *batch gradient descent*, as discussed above, which takes into consideration *all* n training samples $\{x_1^{(i)}, x_2^{(i)}, \ldots, x_p^{(i)}, y^{(i)}\}$ at every iteration of the update rule for *each* parameter β_j (denoted by the summation). Your function should take as input the training dataset, the learning rate α, the initial value $\beta = \{\beta_0, \beta_1, \ldots, \beta_p\}$, and an accuracy margin ϵ that controls the termination of the algorithm. The function should output the final value of the coefficients $\beta = \{\beta_0, \beta_1, \ldots, \beta_p\}$ once the accuracy margin ϵ is met.

Problem 2 [20 points]

Background: Notice that in the batch gradient descent algorithm, each update of β requires you to scan through the entire set of n training samples (note the summation), making it slow for large n. One may modify the algorithm for gradient descent so that it starts updating β_j (for every j) by scanning a single training sample at a time, and continue this way for the complete training set. This modified version of the algorithm is also known as *stochastic gradient descent*.

Problem statement: Write an `R` (or a Python) function for *stochastic gradient descent*, as above, which takes into account *one* of the n training samples $\{x_1^{(i)}, x_2^{(i)}, \ldots, x_p^{(i)}, y^{(i)}\}$ at every iteration of the update rule for *each* parameter β_j (omitting the summation). Your function should take as input the training dataset, the learning rate α, the initial value $\beta = \{\beta_0, \beta_1, \ldots, \beta_p\}$, and an accuracy margin ϵ that controls the termination of the algorithm. The function should output the final value of the coefficients $\beta = \{\beta_0, \beta_1, \ldots, \beta_p\}$ once the accuracy margin ϵ is met.

Problem 3 [10 points]

Background: Write the RSS (similar to cost function) of gradient descent in vectorial format:

$$\text{RSS}(\beta) = 2n \times J(\beta) = \sum_{i=1}^{n} \left(h_\beta(x^{(i)}) - y^{(i)} \right)^2 = (X\beta - y)^T (X\beta - y),$$

where X is an $n \times (p+1)$ matrix built of rows $(1, x_1^{(i)}, x_2^{(i)}, \ldots, x_p^{(i)})$, feature vector β is a $(p+1) \times 1$ vector $(\beta_0, \beta_1, \ldots, \beta_p)^T$, and the actual values of y is an $n \times 1$ vector $(y^{(1)}, y^{(2)}, \ldots, y^{(n)})^T$.

Problem statement: Minimize RSS(β) using the notion of matrix calculus, by setting the 'derivative' of RSS(β) to zero, and obtain the analytic solution for β that minimizes RSS(β). The equation set representing such an analytic solution is called the *normal equations* for β. Write an `R` (or a Python) function for linear regression using the analytic formula for β you just computed. Your function should take as input the training data set, and output the final value of the coefficients $\beta = \{\beta_0, \beta_1, \ldots, \beta_p\}$, computed using the analytic form of *normal equations*.