1 Exercise 1

In this exercise, you will implement a first version of *your own gradient descent algorithm* to solve the ridge regression problem. Throughout the homeworks, you will keep improving and extending your gradient descent optimization algorithm. In this homework, you will implement a basic version of the algorithm.

Recall from Week 1 and Week 2 Lectures that the ridge regression problem writes as

$$\min_{\beta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \lambda \|\beta\|_2^2 , \tag{1}$$

that is, if you expand

$$\min_{\beta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{d} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{d} \beta_j^2 . \tag{2}$$

1.1 Remarks

Several remarks are in order.

Normalization Note that there is a $1/n$ normalization factor in the empirical risk term in the equations, while there is not in *An Introduction to Statistical Learning*. Note also that there is a λ multiplicative factor in the regularization penalty term in the equations. Sometimes, in articles, you may see the normalization $\lambda/2$ instead for the ℓ_2^2-regularization penalty. This is convenient when you compute the gradient of that term because the 2 and the $1/2$ cancel.

You can actually normalize the terms any way you want *as long as you are consistent all the way through* in your mathematical derivations, your codes, and your experiments (especially when you do cross-validation).

So here is my general advice:

- do normalize the empirical risk term so that it is an average, not a sum; this normalization will be important for large scale problems where the sum can become very large.
- check what optimization problem exactly is solved when you use a library, so you can compare your solution to the optimization problem to the solution found by the library and compare the optimal value of the regularization found by your cross-validation to the one found the library's cross-validation.

Intercept It is common in traditional statistics and machine learning books and libraries to include an intercept β_0 in the statistical model. Having a separate intercept coefficient is actually not that important, and provably so, especially if the data was properly centered and standardized beforehand.

There is actually a simple way to bypass the issue of having a separate intercept coefficient by adding a constant variable 1 in the variables. See Sec. 2.3.1 of *The Elements of Statistical Learning*. So the d variables in the equations correspond to the $(d-1)$ original variables plus 1 dummy variable equal to 1.

1.2 Gradient descent

The gradient descent algorithm is an iterative algorithm that is able to solve differentiable optimization problems such as (1). Define

$$F(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i^T\beta)^2 \ + \lambda\|\beta\|_2^2 \,. \tag{3}$$

Gradient descent generates a sequence of iterates[1] (β_t) that converges to the optimal solution $\beta^\star$ of (1). The optimal solution of (1) is defined as

$$F(\beta^\star) = \min_{\beta\in\mathbb{R}^d} F(\beta) \,. \tag{4}$$

Gradient descent is outlined in Algorithm 1. The algorithm requires a sub-routine that computes the gradient for any β. The algorithm also takes as input the value of the constant step-size η.

- Assume that $d = 1$ and $n = 1$. The sample is then of size 1 and boils down to just (x, y). The function F writes simply as

  $$F(\beta) = (y - x\,\beta)^2 \ + \lambda\beta^2 \,. \tag{5}$$

  Compute the gradient ∇F of F.

[1]The subscript t refers to the iteration counter here, not to the coordinates of the vector β.

Algorithm 1 Gradient Descent algorithm with fixed constant step-size

input step-size η
initialization $\beta_0 = 0$
repeat for $t = 0, 1, 2, \dots$
$\beta_{t+1} = \beta_t - \eta \nabla F(\beta_t)$
until the stopping criterion is satisfied.

- Assume now that $d > 1$ and $n > 1$. Using the previous result and the linearity of differentiation, compute the gradient $\nabla F(\beta)$ of F.
- Consider the Hitters dataset considered in the Week 2 Lecture+Lab. Consider the training-test split from Sec. 3.2 of Week 2 Lecture+Lab (before the cross-validation part). Standardize the data.
- Write a function *computegrad* that computes and returns $\nabla F(\beta)$ for any β.
- Write a function *graddescent* that implements the gradient descent algorithm described in Algorithm 1. The function *graddescent* calls the function *computegrad* as a subroutine. The function takes as input the initial point, the constant step-size value, and the maximum number of iterations. The stopping criterion is the maximum number of iterations.
- Set the constant step-size to $\eta = 0.1$ and the maximum number of iterations to 1000. Run *graddescent* on the training set of the Hitters dataset for $\lambda = 0.1$. Plot the curve of the objective value $F(\beta_t)$ versus the iteration counter t. What do you observe?
- Denote β_T the final iterate of your gradient descent algorithm. Compare β_T to the $\beta^\star$ found by *glmnet*. Compare the objective value for β_T to the one for $\beta^\star$. What do you observe?
- Run your gradient algorithm for many values of η on a logarithmic scale. Find the final iterate, across all runs for all the values of η, that achieves the smallest value of the objective. Compare β_T to the $\beta^\star$ found by *glmnet*. Compare the objective value for β_T to the $\beta^\star$. What conclusion to you draw?