

STK-IN4300 Mandatory Assignment 1

Gabriel Sigurd Cabrera

October 30, 2019

Problem 1

In this exercise, we will be performing a comparison of regression methods on a dataset provided by the *European Bioinformatics Institute* at <https://www.ebi.ac.uk/arrayexpress/experiments/E-GEOD-12288/>. We will be exploring LASSO and ridge methods for a selection of hyperparameter (λ) values; with the help of visual aides, we will compare and contrast the dependency of the *mean squared error* (or *MSE*) on λ , in order to gauge the models' relative efficacies.

The dataset we will be working with is comprised of an input array $\mathbf{X} \in \mathbb{R}^{222 \times 22283}$ with corresponding outputs $\mathbf{y} \in \mathbb{R}^{222}$; to analyze this data, a `python` script¹ was used in conjunction with the `rp2`, `NumPy`, `multiprocessing`, and `sklearn` modules. The data is normalized as follows:

$$\mathbf{x}_{\text{norm}} = \frac{\mathbf{x} - \text{mean}(\mathbf{x})}{\text{std}(\mathbf{x})}$$

Using `sklearn`, implementing the LASSO and ridge algorithms is simple; in Figure 1 we can see that each method's *MSE* behaves differently as a function of the hyperparameter:

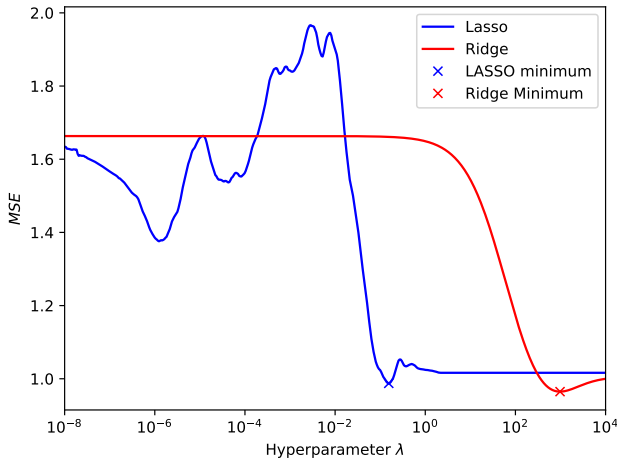


Figure 1: The *MSE* as a function of λ for a 1-D polynomial regression using the LASSO and ridge algorithms; averaged over 10-fold cross validation. With maximum 1000 LASSO iterations

We see that the *MSE* for LASSO is often superior to the ridge's *MSE*, but not for all values. Additionally, ridge regression becomes permanently superior for hyperparameter

values greater than $\lambda \approx 300$. Things change somewhat once the iteration maximum is restricted to smaller values:

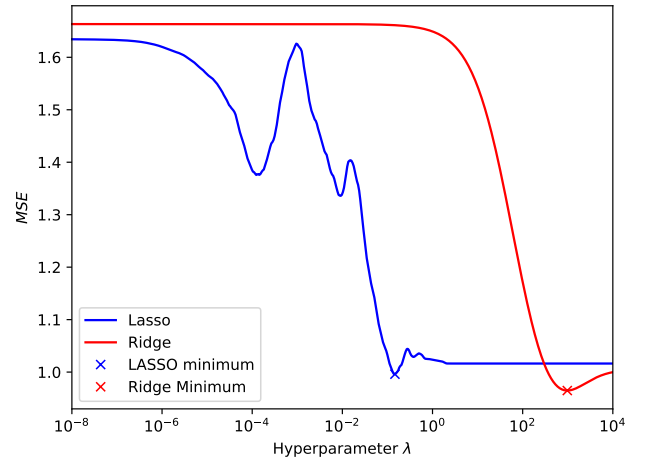


Figure 2: The *MSE* as a function of λ for a 1-D polynomial regression using the LASSO and ridge algorithms; averaged over 10-fold cross validation. With ten LASSO iterations.

In Figure 2, we see that LASSO consistently becomes the better choice until $\lambda \approx 300$ – once the hyperparameter gets large enough, ridge regression is once again optimal.

Iteration Max.	Min. LASSO <i>MSE</i>	Optimal λ
10	0.996	0.147
100	0.988	0.155
1000	0.987	0.155

Table 1: Comparing the LASSO algorithm's benchmark values for varying iteration maxima. The minimum *MSE* for ridge regression is 0.965 at $\lambda = 977.192$.

In conclusion, it appears that ridge is the regression method that overall leads to the best performance, with an important caveat: for $\lambda < 300$, LASSO might be superior, depending of the iteration maximum. In addition, perhaps further increasing the iteration maximum would decrease the minimum LASSO *MSE*, as we see a trend implying this in Table 1; of course, the performance impact would be absolutely non-negligible, as LASSO is already computationally heavier than ridge for the cases tested in this report.

The program used to calculate and plot the *MSE* is available online at <https://github.com/GabrielSCabrera/MachineLearning/blob/master/STK-IN4300/Oblig1/a.py>

¹Available in the **Appendix**.

Problem 2

We are given the *linearized* expression for the *object function*:

$$A \equiv \sum_{i=1}^N g'(\mathbf{w}_{\text{old}}^T \mathbf{x}_i)^2 \left(\frac{y_i - g(\mathbf{w}_{\text{old}}^T \mathbf{x}_i)}{g'(\mathbf{w}_{\text{old}}^T \mathbf{x}_i)} + \mathbf{w}_{\text{old}}^T \mathbf{x}_i - \mathbf{w}^T \mathbf{x}_i \right)^2 \quad (1)$$

Where $y_i, g, g' \in \mathbb{R}$; we also have that $\mathbf{w}, \mathbf{w}_{\text{old}}, \mathbf{x}_i \in \mathbb{R}^{p \times 1}$ for $i = 1, 2, \dots, N$. In a practical sense, N might represent the number of points in a dataset, with p representing the number of features present in said dataset.

We are interested in minimizing the scalar-valued A ; to accomplish this, we must take the derivative of A with respect to \mathbf{w} . We will then set this derivative to zero, and solve for the smallest possible \mathbf{w} ; we will call this value \mathbf{w}_{min} .

To accomplish this, we must redefine (1) such that its *summation notation* is replaced with a *vector/matrix* expression; we begin by redefining some terms.

$$\begin{aligned} \mathbf{x}^T &\equiv [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \dots \quad \mathbf{x}_N] \\ p_i &\equiv \frac{y_i - g(\mathbf{w}_{\text{old}}^T \mathbf{x}_i)}{g'(\mathbf{w}_{\text{old}}^T \mathbf{x}_i)} + \mathbf{w}_{\text{old}}^T \mathbf{x}_i \\ \mathbf{p} &\equiv [p_1 \quad p_2 \quad \dots \quad p_N] \\ q_i &\equiv g'(\mathbf{w}_{\text{old}}^T \mathbf{x}_i)^2 \\ \mathbf{q} &\equiv [q_1 \quad q_2 \quad \dots \quad q_N] \\ \mathbf{r} &\equiv \text{diag}(\mathbf{q})^2 \end{aligned}$$

This gives us²

$$A = \sum_{i=1}^N q_i (p_i - \mathbf{w}^T \mathbf{x}_i)^2 = (\mathbf{p} - \mathbf{w}^T \mathbf{x}^T) \mathbf{r} (\mathbf{p} - \mathbf{w}^T \mathbf{x}^T)^T \quad (2)$$

We then expand the above, giving us several easily-differentiable terms:

$$A = \mathbf{p} \mathbf{r} \mathbf{p}^T - \mathbf{p} \mathbf{r} \mathbf{x} \mathbf{w} - \mathbf{w}^T \mathbf{x}^T \mathbf{r} \mathbf{p}^T + \mathbf{w}^T \mathbf{x}^T \mathbf{r} \mathbf{x} \mathbf{w}$$

Since each term is a scalar, it is valid to replace each of them with their own transpose if need be. Consider the fact that $\mathbf{w}^T \mathbf{x}^T \mathbf{r} \mathbf{p}^T = (\mathbf{p} \mathbf{r} \mathbf{x} \mathbf{w})^T$; since \mathbf{r} is a *diagonal* matrix, we have that $\mathbf{r} = \mathbf{r}^T$, and so we can combine some terms:

$$A = \mathbf{p} \mathbf{r} \mathbf{p}^T - 2 \mathbf{p} \mathbf{r} \mathbf{x} \mathbf{w} + \mathbf{w}^T \mathbf{x}^T \mathbf{r} \mathbf{x} \mathbf{w}$$

Next, we differentiate with respect to \mathbf{w} , keeping in mind the rule $\nabla_{\mathbf{x}} \mathbf{x}^T \mathbf{v} \mathbf{x} = 2 \mathbf{v} \mathbf{x}$ for \mathbf{v} independent of \mathbf{x} :

$$\nabla_{\mathbf{w}} A = -2 \mathbf{p} \mathbf{r} \mathbf{x} + 2 \mathbf{x}^T \mathbf{r} \mathbf{x} \mathbf{w}$$

Setting the above equal to zero allows us to minimize \mathbf{w} :

$$0 = -2 \mathbf{p} \mathbf{r} \mathbf{x} + 2 \mathbf{x}^T \mathbf{r} \mathbf{x} \mathbf{w}_{\text{min}}$$

Finally in (3), we are left with our desired result:

$$\mathbf{w}_{\text{min}} = (\mathbf{x}^T \mathbf{r} \mathbf{x})^{-1} (\mathbf{p} \mathbf{r} \mathbf{x}) \quad (3)$$

Appendix

We can use `python` to verify that (2) holds; below is a script that will generate ten-thousand sets of randomly shaped arrays containing random values – these sets consist of the matrix \mathbf{x} , and vectors \mathbf{w} , \mathbf{p} , and \mathbf{q} . Using the `NumPy` and `multiprocessing` modules, we calculate the difference between the values calculated by both sides of (2) for each set of initial conditions.

```
from multiprocessing import Pool
import numpy as np

np.random.seed(123)

def test(dummy):
    N = np.random.randint(5, 50) # Number of datapoints
    p = np.random.randint(2, 6) # Number of features
    X = np.random.random((N,p)) # Matrix x
    W = np.random.random((p,1)) # Vector w
    P = np.random.random((1,N)) # Vector p
    Q = np.random.random((1,N)) # Vector q

    # Evaluating the summation form
    summation_total = 0
    for i in range(N):
        x_i = X[i,:,np.newaxis] # Vector x_i
        summation_total += (Q[:,i]**2)*((P[:,i] - W.T @ x_i)**2)

    root_R = np.zeros((N,N))
    for i in range(N):
        root_R[i,i] = Q[:,i]
    R = root_R @ root_R # Matrix r

    # Evaluating the vector form
    vector_total = (P - W.T @ X.T) @ R @ (P - W.T @ X.T).T

    # Calculating and saving the difference between each total
    difference = np.squeeze(np.abs(summation_total - vector_total))
    return difference

N_tests = 1E4

# Running tests a total of "N_tests" times
pool = Pool()
results = np.array(pool.map(test, (None for i in range(int(N_tests)))))

# Gathering information on the results and printing
max_res, mean_res, median_res = \
np.max(results), np.mean(results), np.median(results)
print("Information on differences between summation total and vector total")
print(f"\tMaximum: {max_res}\n\tMean: {mean_res}\n\tMedian: {median_res}")
```

The output is as follows:

```
Information on differences between summation total and vector total
Maximum: 1.0658141036401503e-14
Mean: 5.56243939797696e-16
Median: 1.1102230246251565e-16
```

Clearly, the only differences are due to numerical error; our assertion that (2) holds is therefore near-certain to be true.

²This can be verified to be true programmatically – see the **Appendix**.