STK-IN4300 Mandatory Assignment 1

Gabriel Sigurd Cabrera

October 15, 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` script[1] was used in conjunction with the `rpy2`, `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:

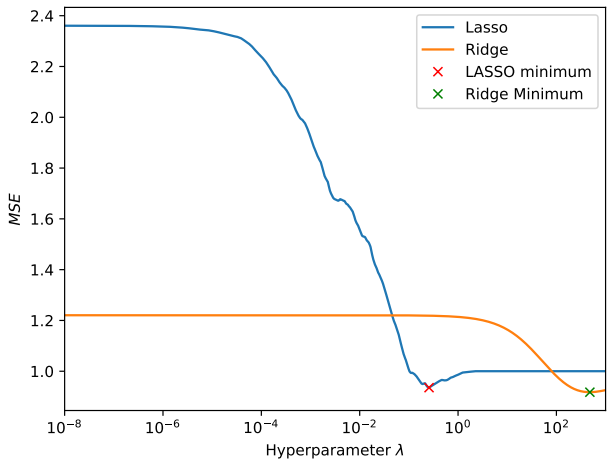

Figure 1: The MSE as a function of λ for a 1-D polynomial regression using the LASSO and ridge algorithms.

We see that the MSE for LASSO is consistently superior to the ridge's MSE, for all hyperparameters in the selected range.

This is likely due to the fact that the `lasso` algorithm is allowed to iterate one-thousand times before breaking the convergence loop; we see further proof in Figure 2, where the maximum number of iterations was set to five:

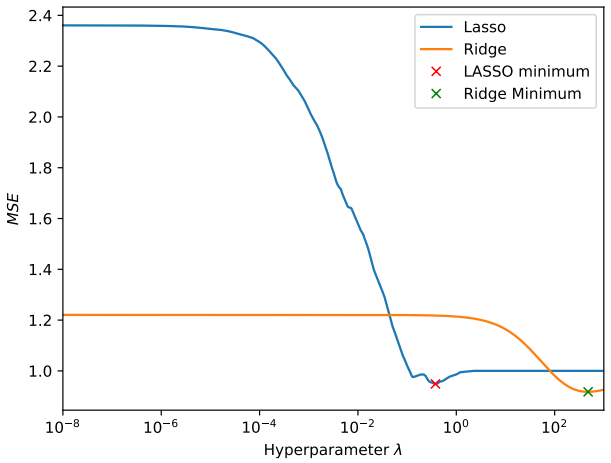

Figure 2: The MSE as a function of λ for a 1-D polynomial regression using the LASSO and ridge algorithms; with five LASSO iterations.

Here, we see that LASSO is not consistently better than ridge, and that it is only competitive for λ

Problem 2

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

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

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 $i = 1, 2, ..., N$. 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 $\mathbf{w}_{\text{min}}$.

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.

[1] Available in the **Appendix**.

$$\mathbf{x}^{\mathrm{T}} \equiv \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{\mathrm{N}} \end{bmatrix}$$

$$p_i \equiv \frac{y_i - g(\mathbf{w}_{\mathrm{old}}^{\mathrm{T}} \mathbf{x}_i)}{g'(\mathbf{w}_{\mathrm{old}}^{\mathrm{T}} \mathbf{x}_i)} + \mathbf{w}_{\mathrm{old}}^{\mathrm{T}} \mathbf{x}_i$$

$$\mathbf{p} \equiv \begin{bmatrix} p_1 & p_2 & \cdots & p_{\mathrm{N}} \end{bmatrix}$$

$$q_i \equiv g'(\mathbf{w}_{\mathrm{old}}^{\mathrm{T}} \mathbf{x}_i)^2$$

$$\mathbf{q} \equiv \begin{bmatrix} q_1 & q_2 & \cdots & q_{\mathrm{N}} \end{bmatrix}$$

$$\mathbf{r} \equiv \mathrm{diag}(\mathbf{q})^2$$

This gives us[2]

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

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

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

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 $\mathbf{w}^{\mathrm{T}}\mathbf{x}^{\mathrm{T}}\mathbf{r}^{\mathrm{T}}\mathbf{p}^{\mathrm{T}} = (\mathbf{p}\mathbf{r}\mathbf{x}\mathbf{w})^{\mathrm{T}}$; since $\mathbf{r}$ is a *diagonal* matrix, we have that $\mathbf{r} = \mathbf{r}^{\mathrm{T}}$, and so we can combine some terms:

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

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

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

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

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

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

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

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

Appendix

Problem 1

The program used to calculate and plot the MSE is given below:

```
from sklearn.preprocessing import ↩
    PolynomialFeatures as Poly
from sklearn.model_selection import ↩
    train_test_split
from sklearn.linear_model import Lasso, Ridge
from sklearn.preprocessing import normalize
from sklearn.model_selection import KFold
import rpy2.robjects as robjects
from multiprocessing import Pool
import matplotlib.pyplot as plt
import numpy as np

from warnings import filterwarnings
filterwarnings('ignore')

def calculate(data):
    MSE_lasso_step = []
    MSE_ridge_step = []
    a = data[0]
    max_iter = data[1]

    for X_train, X_test, y_train, y_test in data↩
        [2]:

        lasso = Lasso(alpha = a, max_iter = ↩
            max_iter)
        lasso.fit(X_train, y_train)

        ridge = Ridge(alpha = a)
        ridge.fit(X_train, y_train)

        y_lasso = lasso.predict(X_test)
        y_ridge = ridge.predict(X_test)

        MSE_lasso_step.append(np.mean((y_lasso - ↩
            y_test)**2))
        MSE_ridge_step.append(np.mean((y_ridge - ↩
            y_test)**2))

    return np.mean(MSE_lasso_step), np.mean(↩
        MSE_ridge_step)

robjects.r['load']("data_01.rdata")
X = np.array(robjects.r['X'])
y =  np.array(robjects.r['y'])

X = (X - np.mean(X))/np.std(X)
y = (y - np.mean(y))/np.std(y)

k = 10
degree = 1
N = 1E3
alphas = np.logspace(-8, 3, N)
max_iter_vals = [5, 1000]
filenames = ["a_5_iter", "a"]

kf = KFold(n_splits = k)
sets = []
poly = Poly(degree = degree)
for train_index, test_index in kf.split(X):
    sets.append((poly.fit_transform(X[train_index])↩
        ,
    poly.fit_transform(X[test_index]), y[↩
        train_index], y[test_index]))

for max_iter, filename in zip(max_iter_vals, ↩
    filenames):

    MSE_lasso = np.zeros_like(alphas)
    MSE_ridge = np.zeros_like(alphas)

    pool = Pool()
    n = 0
    for i in pool.imap(calculate, ([alphas[j], ↩
        max_iter, sets] for j in range(int(N)))):
        MSE_lasso[n], MSE_ridge[n] = i
        n += 1
        print(f"\r{100*n/len(alphas):.2f}", end = "↩
            ")
    print()

    argmin_lasso = np.argmin(MSE_lasso)
    argmin_ridge = np.argmin(MSE_ridge)
    print(f"LASSO lambda = {alphas[argmin_lasso]}, ↩
        MSE = {MSE_lasso[argmin_lasso]}")
    print(f"Ridge lambda = {alphas[argmin_ridge]}, ↩
        MSE = {MSE_ridge[argmin_ridge]}")

    plotfunc = plt.semilogx

    plotfunc(alphas, MSE_lasso, "b-", label = "↩
        LASSO")
    plotfunc(alphas, MSE_ridge, "r-", label = "↩
        Ridge")
    plotfunc([alphas[argmin_lasso]], [MSE_lasso[↩
        argmin_lasso]], "bx", label = "LASSO ↩
        minimum")
    plotfunc([alphas[argmin_ridge]], [MSE_ridge[↩
        argmin_ridge]], "rx", label = "Ridge ↩
        Minimum")
    plt.xlabel("Hyperparameter $\\lambda$")
    plt.ylabel("$MSE$")
    plt.legend()
    plt.xlim([np.min(alphas), np.max(alphas)])
    plt.savefig(f"{filename}.png", dpi = 250)
    plt.close()
```

Problem 2

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.