CS 534 – Implementation Assignment 1

Due 11:59pm, October 8^{th}, 2016

---

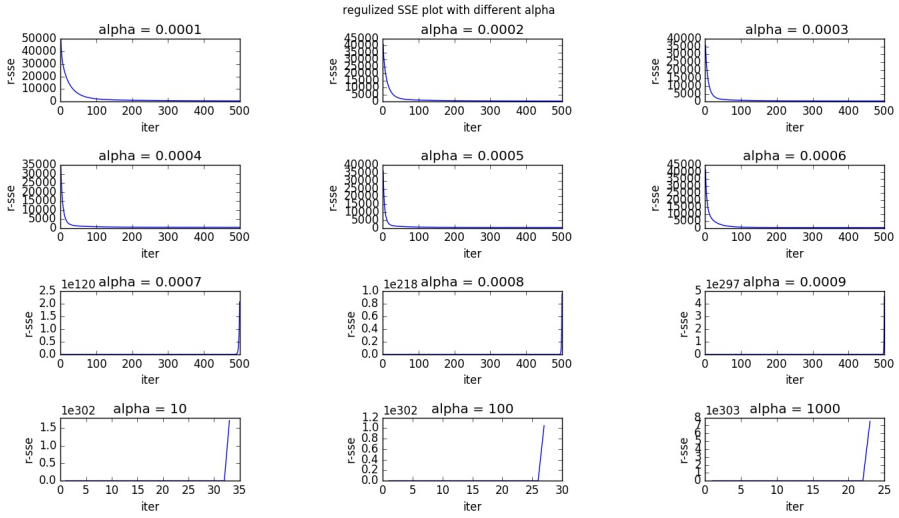
1 Introduction

In this project, we are exploring the effect of learning rate α and regularization parameter λ under linear regression with L_2 regularization context using provided training dataset and test dataset. There are three parts as follows,

- for a fixed regularization parameter λ_0($\lambda_0 = 0.1$ in our case), explore different learning rates for batch gradient algorithm and pick up the best learning rate for the training data
- for different λ value, minimize the regularized sum of square error for training data to find estimate for $\mathbf{w}$, then use the fitted model to calculate sse for test data to find the best λ
- for training data, use 10-fold cross-validation to find the best λ which minimize sse

2 Learning rate for gradient descent

To choose the most appropriate learning rate, we conduct the exploration in the fixed λ context. In our case, we use $\lambda = 0.1$, and optimize the regularized sum of square error with learning rate α in $[0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008, 0.0009, 10, 100, 1000]$. The r-sse versus iteration time for each α is as follows,



To choose the most appropriate learning rate, firstly we need the r-sse decreases and eventually converges as iteration time grows. Then we consider the convergence rate and the limit value of r-sse. From plot, when α in $[0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008, 0.0009]$, r-sse converges. Among these α's, with value 0.0005, the r-sse converges with rapidest speed and

smallest limit value. So under this setting, 0.0005 is the most appropriate learning value. For other setting, though the result will differ, still we believe a small learning rate will be appropriate for this training data.

3 Experiments with different λ values

We already know form Homework 1 that the solution of this ridge regression problem is

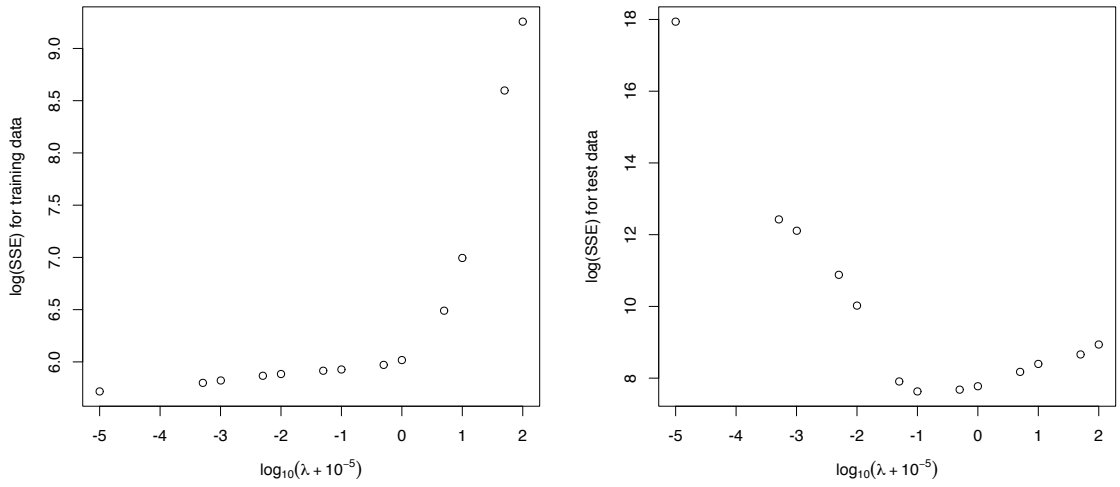
$$\widehat{w}_{ridge}^{\lambda} = \left(X^T X + \lambda I\right)^{-1} X^T Y$$

for any fixed λ. However we don't know which λ is the best one to choose. In this section we explore the behavior of different λ. First we choose a set of λ to be

$$\{0, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 0.5, 1, 5, 10, 50, 100\},$$

and calculate the SSE for training data and the test data. And we summarize the result in Figure 1. In the figure, the x-axis is $\log_{10}\left(\lambda + 10^{-5}\right)$, and y-axis is the $\log\left(SSE\right)$ (sum of squared error) for different data sets (training data on the left panel and test data on the right panel).

Figure 1: $log\left(SSE\right)$ for different $log\left(\lambda + 10^{-5}\right)$



From the figure, we can see that

I. For training data set, $\log\left(SSE_{train}\right)$ is increasing with $\log_{10}\left(\lambda + 10^{-5}\right)$. Since log is a monotone increasing function, we can say SSE_{train} is increasing with λ. The reason for this is because $\widehat{w}_{LS} = \left(X^T X\right)^{-1} X^T Y = \arg\min_w \sum_{i=1}^{N} \left(y_i - w^T x_i\right)^2$, and this optimization problem is convex. The larger λ is, the farther away $\widehat{w}_{ridge}^{\lambda}$ is from $\widehat{w}_{LS}$. Then due to convexity, $SSE_{train}\left(\widehat{w}_{ridge}^{\lambda}\right) = \sum_{i=1}^{N} \left(y_i - \left(\widehat{w}_{ridge}^{\lambda}\right)^T x_i\right)^2$ digresses more from the minimum SSE_{train}, i.e., SSE_{train} grows larger and larger as λ increases.

II. For test data set, when $\log_{10}\left(\lambda+10^{-5}\right)$ increases, $\log\left(SSE_{test}\right)$ first decreases and then increases. Therefore when λ increases, SSE has the same trend with $\log\left(SSE_{test}\right)$, i.e., first decreases and then increases. Denote Y_0, X_0 as the response and design matrix for test data, respectively. Then

$$\begin{aligned}
\mathbb{E}\left(SSE_{test}\right) &= \mathbb{E}\left\|Y_0 - X_0^T\widehat{w}_{ridge}^{\lambda}\right\|^2 \\
&= \mathbb{E}\left\|Y_0 - X_0^T w + X_0^T w - X_0^T\widehat{w}_{ridge}^{\lambda}\right\|^2 \\
&= \mathbb{E}\left\|\varepsilon_0 + X_0^T w - X_0^T\widehat{w}_{ridge}^{\lambda}\right\|^2 \\
&= \mathbb{E}\left(\varepsilon_0\right)^2 + \mathbb{E}\left\|X_0^T w - X_0^T\widehat{w}_{ridge}^{\lambda}\right\|^2 \\
&= Var\left(\varepsilon_0\right) + \mathbb{E}\left\|X_0^T\widehat{w}_{ridge}^{\lambda} - \mathbb{E}\left(X_0^T\widehat{w}_{ridge}^{\lambda}\right) + \mathbb{E}\left(X_0^T\widehat{w}_{ridge}^{\lambda}\right) - X_0^T w\right\|^2 \\
&= Var\left(Y_0\right) + \mathbb{E}\left\|X_0^T\widehat{w}_{ridge}^{\lambda} - \mathbb{E}\left(X_0^T\widehat{w}_{ridge}^{\lambda}\right)\right\|^2 + \mathbb{E}\left\|\mathbb{E}\left(X_0^T\widehat{w}_{ridge}^{\lambda}\right) - X_0^T w\right\|^2 \\
&= Var\left(Y_0\right) + Var\left(X_0^T\widehat{w}_{ridge}^{\lambda}\right) + Bias^2\left(X_0^T\widehat{w}_{ridge}^{\lambda}\right)
\end{aligned}$$

For large λ, $Var\left(X_0^T\widehat{w}_{ridge}^{\lambda}\right) = \sigma^2 X_0^T\left(X^T X+\lambda I\right)^{-1} X_0$ would be small, but the estimator is more biased, i.e., $Bias^2\left(X_0^T\widehat{w}_{ridge}^{\lambda}\right)$ would increase. Therefore there exists a bias-variance trade-off. Too large λ would cause smaller variance but large absolute bias, while too small λ will lead to smaller absolute bias but large variance. In both cases, $\mathbb{E}\left(SSE_{test}\right)$ will be large. Therefore the best λ that has the smallest SSE_{test} should be a value that's neither too large nor too small. This well explains the trend of SSE_{test} v.s. λ displayed in right panel of Figure 1.

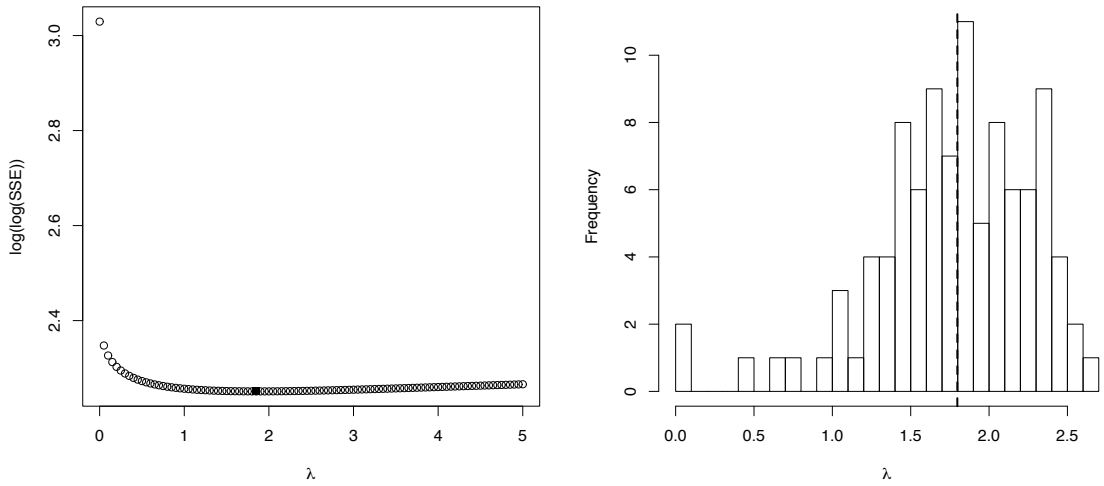
4 Selecting the best λ

We have run the 10-fold cross validation and get the best λ to be 1.85. See Figure 2 left panel. The figure displays the relationship of $\log\left\{\log\left(SSE\right)\right\}$ (y-axis) with λ (x-axis). The trend is: cross-validation *SSE* first increases and then decreases as λ increases. The solid point in the figure is the one with the smallest cross-validation SSE, and the corresponding λ is 1.85.

Note that every time we run the cross validation, we are randomly splitting the whole data set into 10 folds, so we may get a different result every time we run it. We have replicated the cross-validation process 100 times and get a histogram of the best λ on the right panel of Figure 2. Note that the vertical dashed line is the mean of theses 100 $\lambda's$, which is 1.7975. Therefore we say the best λ is 1.7975.

The range of cross validation SSE is larger than the training SSE. The reason is that the training SSE is calculated via the $\widehat{w}_{ridge}^{\lambda}$, which is also obtained from the whole training set; i.e., we are using the learning out come from the whole training set to predict the set itself, which should be pretty accurate. For the cross validation SSE, each time we are training 9 folds, and predicting the 10^{th} fold, i.e., we are always predicting something new, which is always less accurate compared with predicting the set itself.

Figure 2: $\log(\log(SSE))$ v.s. λ (left panel); Histogram of best λ for 100 replications (right panel)



As we change λ from small to large, the cross validation SSE first decreases and then increases (See left panel of Figure 2), which resembles more the trend of test SSE. The reason behind this is the same as described above: both cross-validation and test SSE are gained from predicting a new data set. More specifically

$$\mathbb{E}(SSE_{new}) = Var(Y_{new}) + Var\left(X_{new}^T \widehat{w}_{ridge}^{\lambda}\right) + Bias^2\left(X_{new}^T \widehat{w}_{ridge}^{\lambda}\right),$$

as λ changes from small to large, $Var\left(X_{new}^T \widehat{w}_{ridge}^{\lambda}\right)$ decreases, while $Bias^2\left(X_{new}^T \widehat{w}_{ridge}^{\lambda}\right)$ increases. Therefore too small λ yields large $Var\left(X_{new}^T \widehat{w}_{ridge}^{\lambda}\right)$, and too large λ yields large $Bias^2\left(X_{new}^T \widehat{w}_{ridge}^{\lambda}\right)$, both of which will lead to a large SSE. That's why the smallest SSE is obtained in the middle.

5 Appendix

We have coded Section 4 and Section 5 in R, and the codes are given below:

```
########## Data Input #########
normalize <- function(x)  {
  for(i in 1:ncol(x)){
    x[,i] = (x[,i] - mean(x[,i]))/sd(x[,i])
  }
  return(x)
}

train <- read.csv("train p1-16.csv", head = F)
d <- ncol(train) - 1
X <- train[, 1:d]
X[,-1] <- normalize(X[,-1])
X <- as.matrix(X)
Y <- train[, d+1]

test <- read.csv("test p1-16.csv", head = F)
```

```
testX <- test[, 1:d]
testX[,-1] <- normalize(testX[,-1])
testX <- as.matrix(testX)
testY <- test[, d+1]
```

```
######## Part 1 ##########
from pandas import read_csv
from statistics import mean, stdev
import numpy as np
from numpy.linalg import inv, norm
import matplotlib.pyplot as plt
from random import shuffle



train = read_csv('/Users/wangchunxiao/Desktop/cs534/project1/train.csv', header = None)
train = train.as_matrix()
for i in range(1, 45):
    train[:, i] = (train[:, i] - mean(train[:, i])) / stdev(train[:, i])

test = read_csv('/Users/wangchunxiao/Desktop/cs534/project1/test.csv', header = None)
test = test.as_matrix()
for i in range(1, 45):
    test[:, i] = (test[:, i] - mean(test[:, i])) / stdev(test[:, i])

#Part 1

x = np.matrix(train[:, 0:45])
y = np.matrix(train[:, 45]).T
w0 = np.matrix(np.append([0.01], np.zeros(44))).T

#set a fixed lambda = 0.1
lambda0 = 0.1

  ##batch gradient descent
wtrue = inv(lambda0 * np.matrix(np.identity(45)) + x.T * x) * x.T * y
alpha = np.arange(0, 0.001, 0.0001)
alpha = np.append(alpha, [10, 100, 1000])
alpha = [i for i in alpha if i != 0]
f = np.empty(shape = (500, len(alpha)))
f[:] = np.nan

for i in range(len(alpha)):
  grad = -2 * x.T * (y - x * w0) + 2 * lambda0 * w0
  step = 0
  w_new = w0
  while(norm(grad) > 1e-04 and step < 500):
    w_new = w_new - alpha[i] * grad
    f[step, i] = (y - x * w_new).T * (y - x * w_new) + lambda0 * w_new.T * w_new
    step = step + 1
    grad = -2 * x.T * (y - x * w_new) + 2 * lambda0 * w_new

fig = plt.figure()

ax1 = fig.add_subplot(431)
x = [i + 1 for i in range(500)]
y = [i for i in f[:,0]]
ax1.plot(x,y)
ax1.set_xlabel('iter')
ax1.set_ylabel('r-sse')
ax1.set_title('alpha = 0.0001')

ax2 = fig.add_subplot(432)
x = [i + 1 for i in range(500)]
y = [i for i in f[:,1]]
ax2.plot(x,y)
ax2.set_xlabel('iter')
ax2.set_ylabel('r-sse')
```

```
ax2.set_title('alpha = 0.0002')

ax3 = fig.add_subplot(433)
x = [i + 1 for i in range(500)]
y = [i for i in f[:,2]]
ax3.plot(x,y)
ax3.set_xlabel('iter')
ax3.set_ylabel('r-sse')
ax3.set_title(' alpha = 0.0003')

ax4 = fig.add_subplot(434)
x = [i + 1 for i in range(500)]
y = [i for i in f[:,3]]
ax4.plot(x,y)
ax4.set_xlabel('iter')
ax4.set_ylabel('r-sse')
ax4.set_title('alpha = 0.0004')

ax5 = fig.add_subplot(435)
x = [i + 1 for i in range(500)]
y = [i for i in f[:,4]]
ax5.plot(x,y)
ax5.set_xlabel('iter')
ax5.set_ylabel('r-sse')
ax5.set_title('alpha = 0.0005')

ax6 = fig.add_subplot(436)
x = [i + 1 for i in range(500)]
y = [i for i in f[:,5]]
ax6.plot(x,y)
ax6.set_xlabel('iter')
ax6.set_ylabel('r-sse')
ax6.set_title('alpha = 0.0006')

ax7 = fig.add_subplot(437)
x = [i + 1 for i in range(500)]
y = [i for i in f[:,6]]
ax7.plot(x,y)
ax7.set_xlabel('iter')
ax7.set_ylabel('r-sse')
ax7.set_title('alpha = 0.0007')

ax8 = fig.add_subplot(438)
x = [i + 1 for i in range(500)]
y = [i for i in f[:,7]]
ax8.plot(x,y)
ax8.set_xlabel('iter')
ax8.set_ylabel('r-sse')
ax8.set_title('alpha = 0.0008')

ax9 = fig.add_subplot(439)
x = [i + 1 for i in range(500)]
y = [i for i in f[:,8]]
ax9.plot(x,y)
ax9.set_xlabel('iter')
ax9.set_ylabel('r-sse')
ax9.set_title('alpha = 0.0009')

ax10 = fig.add_subplot(4,3,10)
x = [i + 1 for i in range(500)]
y = [i for i in f[:,9]]
ax10.plot(x,y)
ax10.set_xlabel('iter')
ax10.set_ylabel('r-sse')
ax10.set_title('alpha = 10')

ax11 = fig.add_subplot(4,3,11)
x = [i + 1 for i in range(500)]
```

```
y = [i for i in f[:,10]]
ax11.plot(x,y)
ax11.set_xlabel('iter')
ax11.set_ylabel('r-sse')
ax11.set_title('alpha = 100')

ax12 = fig.add_subplot(4,3,12)
x = [i + 1 for i in range(500)]
y = [i for i in f[:,11]]
ax12.plot(x,y)
ax12.set_xlabel('iter')
ax12.set_ylabel('r-sse')
ax12.set_title('alpha = 1000')

fig.suptitle('regulized SSE plot with different alpha')

fig.tight_layout()


######## Part 2 ##########

lamvec <- c(0, 10^(-3:2), 0.5*10^(-3:2))
sse_train <- rep(NA, length(lamvec))
sse_test <- rep(NA, length(lamvec))

for(i in 1:length(lamvec)){
  w_new <- solve(t(X) %*% X + lamvec[i]*diag(ncol(X))) %*% t(X) %*% Y
  predY <- testX %*% w_new
  Yhat <- X %*% w_new
  sse_train[i] <- sum((Yhat - Y)^2)
  sse_test[i] <- sum((predY - testY)^2)
}
par(mfrow = c(1,2))
plot(log(lamvec + 10^(-5), base = 10), log(sse_train), xlab = expression(log[10](lambda+10^-5)),
     ylab = "log(SSE) for training data")
plot(log(lamvec + 10^(-5), base = 10), log(sse_test), xlab = expression(log[10](lambda+10^-5)),
     ylab = "log(SSE) for test data")

######## Part 3 ##########

lamvec <- seq(0.0, 5, length.out = 101)
CV_sse <- rep(0, length(lamvec))
n <- length(Y)
idx <- sample(1:n, n)

for(j in 1:length(lamvec)){
  for(i in 1:10){
    ids <- idx[(10*(i-1)+1):(10*i)]
    trainX <- X[-ids,]
    trainY <- Y[-ids]
    w_train <- solve(t(trainX) %*% trainX + lamvec[j]*diag(ncol(trainX))) %*% t(trainX) %*% trainY
    CV_sse[j] <- CV_sse[j] + sum((X[ids,] %*% w_train - Y[ids])^2)
  }
}
idx_small <- which.min(CV_sse)
lamvec[idx_small]

par(mfrow = c(1,1))
plot(lamvec, log(log(CV_sse)), xlab = expression(lambda), ylab = "log(log(SSE))")
points(lamvec[idx_small], log(log(CV_sse[idx_small])), pch = 15)

# 100 reps
best_lam <- rep(NA, 100)
for(k in 1:100){
  idx <- sample(1:n, n)
  CV_sse <- rep(0, length(lamvec))
  for(j in 1:length(lamvec)){
    for(i in 1:10){
```

```
      ids <- idx[(10*(i-1)+1):(10*i)]
      trainX <- X[-ids,]
      trainY <- Y[-ids]
      w_train <- solve(t(trainX) %*% trainX + lamvec[j]*diag(ncol(trainX))) %*% t(trainX) %*% trainY
      CV_sse[j] <- CV_sse[j] + sum((X[ids,] %*% w_train - Y[ids])^2)
    }
  }
  best_lam[k] <- lamvec[which.min(CV_sse)]
}

hist(best_lam, breaks = 20, main = "", xlab = expression(lambda))
abline(v = mean(best_lam), lty = 2, lwd = 2)
mean(best_lam)
```