homework4

```
library(reticulate)
use_condaenv("r-reticulte")
```

Problem 1

In Python, implement a numerically-stable ridge regression that takes into account collinear (or nearly collinear) regression variables. Show that it works by comparing it to the output of your R implementation.
For ridge regression, we have the estimated beta $\hat{\beta} = (X^T X + \lambda I)^{-1} X^T Y$. To avoid collinearity collapsing the model computation, I use singular value decomposition to remedy.

```
#data preparation
#implement R self-defined function
lm_ridge<-function(form,d,lambda){
  #grab design matrix and response vector from given input
  d_no_na<-model.frame(form,d)
  X<-model.matrix(form,d_no_na)
  y_name <-as.character(form)[2]
  Y<-matrix(d_no_na[,y_name],ncol=1)
  #singular value decomposition of X to take care of collinearity
  svd_x<- svd(X)
  # beta = V (sigma + lambda I)^{-1} *sigma U^T* Y
  sigma <- diag(svd_x$d)
  lambda_I <-diag(rep(lambda,length(svd_x$d)))
  beta <- svd_x$v %*% solve(sigma^2 +lambda_I) %*% sigma %*% t(svd_x$u) %*% Y
  ret<- list(coefficients= beta,form=form)
  class(ret)<- "ridge_object"
  ret
}
data(iris)
my_rigde = lm_ridge(form=Sepal.Length~Sepal.Width+ Petal.Length+ Petal.Width ,d=iris,lambda=0.001
my_rigdeScoefficients
#>           [,1]
#> [1,]  1.8550400
#> [2,]  0.6510894
#> [3,]  0.7091986
#> [4,] -0.5565463
#python implementation
#import seaborn as sns
#iris = sns.load_dataset("iris")
#X= iris[["sepal_width","petal_length","petal_width"]].values
#y = iris[["sepal_length"]].values.flatten()
#result = ridge_regression(X,y,lambda_param=0.001)
source_python("/Users/kangxinwang/Desktop/ridge_regression.py")
```

```
cor(iris[,c(2:4)])
#>              sepal_width petal_length petal_width
#> sepal_width    1.0000000   -0.4284401  -0.3661259
#> petal_length  -0.4284401    1.0000000   0.9628654
#> petal_width   -0.3661259    0.9628654   1.0000000
```

By inspecting the correlation matrix of our testing data, we can see that we'll encounter collinearity when fitting the models since the correlation between petal width and petal length is very high. The resulting coefficients from both models are rather consistent, with only minor differences.
For python implementation: 0.70035366, 0.73609505, -0.60311265, 1.657345748259607
For r implementation: intercept: 1.8550400
0.6510894
0.7091986
-0.5565463
Both functions work well.

Problem 2

Create an "out-of-core" implementation of the linear model that reads in contiguous rows of a data frame from a file, updates the model. You may read the data from R and send it to your Python functions fo fitting.

```
#data preparation
X = r_to_py(iris[,c(2:4)])
y= r_to_py(iris[,1])
```

For the out-of-core implementation, I retained the stochastic gradient descent algorithm, but optimized memory requirement by splitting the original dataset into small batches, each containing only one row of the data and iterates through all rows to update the model

```
beta = np.zeros(X.shape[1])
data_size = y.shape[0]
for i in range(data_size):
    curr_X = np.asmatrix(X[i,:]).T
    curr_y = np.asmatrix(y[i])
    beta = beta - 0.01*(2*np.matmul(np.matmul(curr_X,curr_X.T),beta)
    -2*curr_X*curr_y )

print(beta[:,0])
#> [[ 1.06294947]
#>  [ 0.52774987]
#>  [-0.01352874]]
```

Problem 3

Implement your own LASSO regression function in Python. Show that the results are the same as the function implemented in the `casl` package.
casl functions prep

```
casl_util_soft_thresh <-
  function(a, b)
  {
    a[abs(a) <= b] <- 0
    a[a > 0] <- a[a > 0] - b
    a[a < 0] <- a[a < 0] + b
    a
  }
casl_lenet_update_beta <-
  function(X, y, lambda, alpha, b, W)
  {
    WX <- W * X
    WX2 <- W * X^2
    Xb <- X %*% b

    for (i in seq_along(b))
    {
      Xb <- Xb - X[, i] * b[i]
      b[i] <- casl_util_soft_thresh(sum(WX[,i, drop=FALSE] *
                                          (y - Xb)),
                                    lambda*alpha)
      b[i] <- b[i] / (sum(WX2[, i]) + lambda * (1 - alpha))
      Xb <- Xb + X[, i] * b[i]
    }
    b
  }
casl_lenet <-
  function(X, y, lambda, alpha = 1, b=matrix(0, nrow=ncol(X), ncol=1),
           tol = 1e-5, maxit=50L, W=rep(1, length(y))/length(y))
  {
    for (j in seq_along(lambda))
    {
      if (j > 1)
      {
        b[,j] <- b[, j-1, drop = FALSE]
      }

      # Update the slope coefficients until they converge.
      for (i in seq(1, maxit))
      {
        b_old <- b[, j]
        b[, j] <- casl_lenet_update_beta(X, y, lambda[j], alpha,
                                         b[, j], W)
        if (all(abs(b[, j] - b_old) < tol)) {
          break
        }
      }
      if (i == maxit)
      {
        warning("Function lenet did not converge.")
      }
    }
    b
  }
```

```
x = iris[,c(2:4)]
y= iris[,1]
casl_lenet(as.matrix(x),y,lambda=0.05)
#>           [,1]
#> [1,] 1.1953001
#> [2,] 0.5710936
#> [3,] 0.0000000
source_python("/Users/kangxinwang/Desktop/my_lasso.py")
```

I implemented my lasso regression in python, somehow it doesn't show the result when it's knitted, but the coefficients are very similar to casl package's results.

Final project proposal

For the final project, I want to study the impact of regularization on deep learning models. When the model is suffering from high variance problem, the neural network may overfit to training data but fails to generalize new features. I want to test the impact on deeplearning models with different regularization stratigies, and study under what circumstances should we regularize the model.