# STAT 542 / CS 598: Homework 1

*Fall 2019, by John Moran (jfmoran2)*

*Due: Monday, Sep 9 by 11:59 PM Pacific Time*

## Contents



### Question 1 [50 Points] KNN

Write an R function to fit a KNN regression model. Complete the following steps

a. [15 Points] Write a function `myknn(xtest, xtrain, ytrain, k)` that fits a KNN model that predict a target point or multiple target points `xtest`. Here `xtrain` is the training dataset covariate value, `ytrain` is the training data outcome, and `k` is the number of nearest neighbors. Use the $\ell_2$ norm to evaluate the distance between two points. Please note that you cannot use any additional `R` package within this function.

```
#---------------------------------------
# euclidean distance function from scratch
#---------------------------------------
euc.dist <- function(x1, x2) {
  sqrt(sum((x1 - x2) ^ 2))
}

#---------------
# myknn function
#---------------
myknn <- function(xtest, xtrain, ytrain, k) {
  ypred <- rep(0, nrow(xtest))

  for(i in c(1:nrow(xtest))) {
    #-------------------------------------------------------------------------
    #calculate the distance from the given row instance of the test data to each
    #row in the training data
    #-------------------------------------------------------------------------
    distances <- apply(xtrain, 1, euc.dist, x2=xtest[i,])

    evalframe <- cbind(xtrain,distances,ytrain)
    orderedframe <- evalframe[order(evalframe[,6]),c(6,7)]

    pred = mean(orderedframe[1:k,2])

    ypred[i] <- pred
  }

  return(ypred)
}
```

b. [10 Points] Generate 1000 observations from a five-dimensional normally distribution:

$$\mathcal{N}(\mu, \Sigma_{5\times 5})$$

where $\mu = (1, 2, 3, 4, 5)^{\mathrm{T}}$ and $\Sigma_{5\times 5}$ is an autoregressive covariance matrix, with the $(i, j)$th entry equal to $0.5^{|i-j|}$. Then, generate outcome values $Y$ based on the linear model

$$Y = X_1 + X_2 + (X_3 - 2.5)^2 + \epsilon$$

where $\epsilon$ follows i.i.d. standard normal distribution. Use `set.seed(1)` right before you generate this entire data. Print the first 3 entries of your data.

```
library(MASS)

set.seed(1)

mu <- c(1,2,3,4,5)
sigma <- matrix(0, nrow=5, ncol=5)
sigma <- outer(1:5, 1:5, function(i, j) 0.5^abs(i-j))

xdata <- mvrnorm(1000, mu=mu, Sigma=sigma)

eps = rnorm(1000)
Y <- matrix(0,nrow=1000, ncol=1)
Y <- xdata[,1] + xdata[,2] + (xdata[,3] - 2.5)^2 + eps

printdata <- cbind(xdata,Y)
colnames(printdata) <- c("X1", "X2", "X3", "X4", "X5", "Y")

printdata[1:3,]
```

```
##              X1       X2       X3       X4       X5        Y
## [1,]  2.0770490 3.555163 2.641969 3.902436 5.108741 4.135994
## [2,]  2.4780195 2.161175 2.188487 2.796376 5.330744 5.365376
## [3,] -0.1413538 2.630428 4.666608 4.493909 5.698190 5.505069
```

c. [10 Points] Use the first 400 observations of your data as the training data and the rest as testing data. Predict the $Y$ values using your KNN function with `k = 5`. Evaluate the prediction accuracy using mean squared error

$$\frac{1}{N}\sum_i (y_i - \widehat{y}_i)^2$$

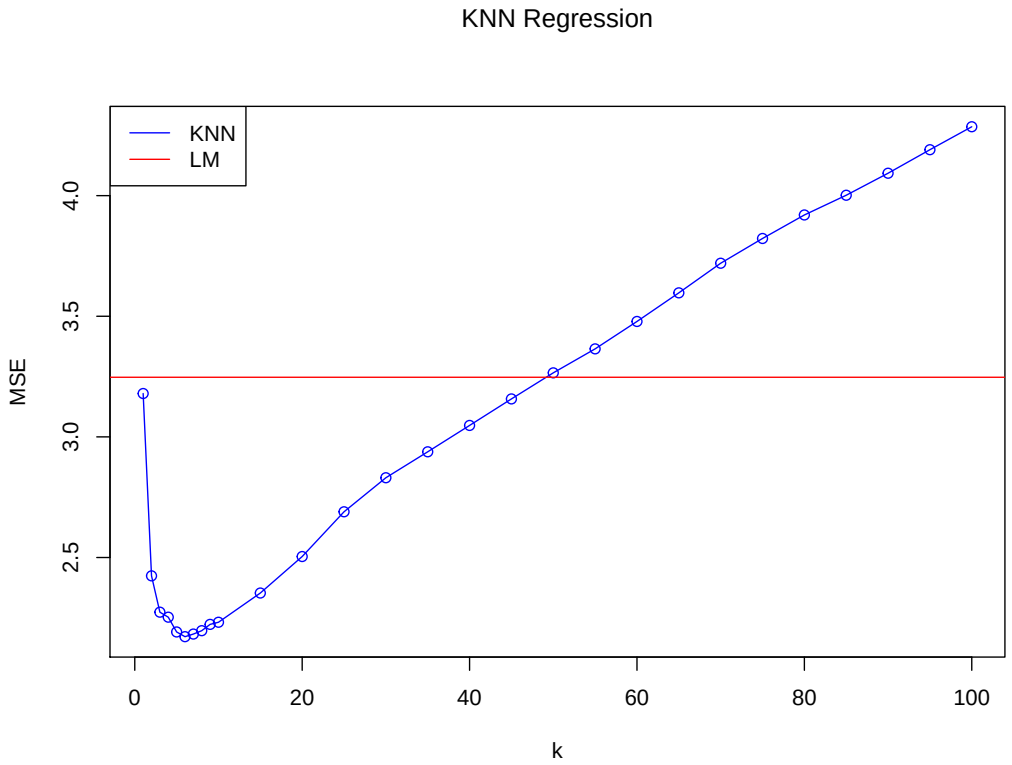
```
#-----------------------------
#  mean square error function
#-----------------------------
mse = function(actual, predicted) {
  mean((actual - predicted) ^ 2)
}

xtrain <- xdata[1:400,]
xtest <- xdata[401:1000,]
ytrain <- Y[1:400]
ytest <- Y[401:1000]

ypred <- myknn(xtest, xtrain, ytrain, 5)
```

```r
MSE_k5 <- mse(ytest,ypred)

print(sprintf("MSE for k=5: %f", MSE_k5))
```

```
## [1] "MSE for k=5: 2.191387"
```

d. [15 Points] Compare the prediction error of a linear model with your KNN model. Consider $k$ being 1, 2, 3, ..., 9, 10, 15, 20, ..., 95, 100. Demonstrate all results in a single, easily interpretable figure with proper legends.

```r
k <- c(1:9,seq(10,100,by=5))
n <- length(k)
knn.MSE <- rep(0,n)

for (i in 1:n) {
  ypred <- myknn(xtest, xtrain, ytrain, k[i])
  knn.MSE[i] <- mse(ytest, ypred)
}

# calculate linear model (LM)
xtrain.df = as.data.frame(xtrain)
xtest.df = as.data.frame(xtest)
colnames(xtest.df) <- c("X1", "X2", "X3", "X4", "X5")
ytrain.df = as.data.frame(ytrain)

master.df = cbind(xtrain.df, ytrain.df)
colnames(master.df) <- c("X1", "X2", "X3", "X4", "X5", "Y")

model.lm = lm(Y ~ ., data=master.df)

pred.lm = predict(model.lm, xtest.df)
lm.MSE <- mse(ytest, pred.lm)

plot(k,knn.MSE, type="o", col="blue", xlab="k", ylab="MSE")
title("KNN Regression", line=3, font.main=1)

legend("topleft", lty = c(1,1), col = c("blue", "red"),
       legend = c("KNN", "LM"))

abline(lm.MSE, 0, col="red")
```




```
print(sprintf("lm.MSE: %f", lm.MSE))
```

```
## [1] "lm.MSE: 3.247147"
```

Alternatively for this plot, since the LM MSE is a single Y axis point with no valid k to plot for the x, instead of using a straight line across all k for the LM MSE, I could have modified the X-axis to be degree of freedom (N/k) and then plotted the LM MSE as a single point on the graph at degree of freedom = 5, since there are 5 parameters in the linear model.

## Question 2 [50 Points] Linear Regression through Optimization

Linear regression is most popular statistical model, and the core technique for solving a linear regression is simply inverting a matrix:

$$\widehat{\boldsymbol{\beta}} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

However, lets consider alternative approaches to solve linear regression through optimization. We use a gradient descent approach. We know that $\widehat{\boldsymbol{\beta}}$ can also be expressed as

$$\widehat{\boldsymbol{\beta}} = \arg\min \ell(\boldsymbol{\beta}) = \arg\min \frac{1}{2n}\sum_{i=1}^{n}(y_i - x_i^{\mathrm{T}}\boldsymbol{\beta})^2.$$

And the gradient can be derived

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i^{\mathrm{T}}\boldsymbol{\beta})x_i.$$

To perform the optimization, we will first set an initial beta value, say $\boldsymbol{\beta} = \mathbf{0}$ for all entries, then proceed with the updating

$$\boldsymbol{\beta}^{\text{new}} = \boldsymbol{\beta}^{\text{old}} - \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \times \delta,$$

where $\delta$ is some small constant, say 0.1. We will keep updating the beta values by setting $\boldsymbol{\beta}^{\text{new}}$ as the old value and calcuting a new one untill the difference between $\boldsymbol{\beta}^{\text{new}}$ and $\boldsymbol{\beta}^{\text{old}}$ is less than a prespecified threshold $\epsilon$, e.g., $\epsilon = 10^{-6}$. You should also set a maximum number of iterations to prevent excessively long runing time.

a. [35 Points] Based on this description, write your own `R` function `mylm_g(x, y, delta, epsilon, maxitr)` to implement this optimization version of linear regression. The output of this function should be a vector of the estimated beta value.

```
library(mlbench)

mylm_g <- function(x, y, delta, epsilon, maxiter) {

  beta_old <- matrix(1,nrow=ncol(x), ncol=1)
  beta_new <- matrix(1,nrow=ncol(x), ncol=1)
  iter <- 1
  beta_diff <- 1

  while (beta_diff > epsilon && iter <= maxiter) {

    beta_new <- beta_old - (delta / nrow(x)) * (t(x) %*% (x %*% beta_old - y))
    beta_diff <- euc.dist(beta_old, beta_new)
    beta_old <- beta_new

    iter <- iter + 1
  }

  print(sprintf("Delta: %f Epsilon: %.9f took %d iterations", delta, epsilon,iter))

  return(beta_new)
}
```

b. [15 Points] Test this function on the Boston Housing data from the `mlbench` package. Documentation is provided here if you need a description of the data. We will remove `medv`, `town` and `tract` from the data and use `cmedv` as the outcome. We will use a scaled and centered version of the data for estimation. Please also note that in this case, you do not need the intercept term. And you should compare your result to the `lm()` function on the same data. Experiment on different `maxitr` values to obtain a good solution. However your function should not run more than a few seconds.

```
library(mlbench)
library(knitr)
data(BostonHousing2)
X = BostonHousing2[, !(colnames(BostonHousing2) %in% c("medv", "town", "tract", "cmedv"))]
X = data.matrix(X)
X = scale(X)
Y = as.vector(scale(BostonHousing2$cmedv))

delta <- 0.1
epsilon <- 10^-6
maxiter <- 5000

beta <- mylm_g(X, Y, delta, epsilon, maxiter)
```

```
## [1] "Delta: 0.100000 Epsilon: 0.000001000 took 1178 iterations"
```

```
beta
```

```
##                 [,1]
## lon     -0.032319596
## lat      0.030241889
## crim    -0.097928372
## zn       0.118259599
## indus    0.011349842
## chas     0.071317218
## nox     -0.199695261
## rm       0.287239327
## age      0.007560419
## dis     -0.321040938
## rad      0.290749939
## tax     -0.236413977
## ptratio -0.206799837
## b        0.091234909
## lstat   -0.417969129
```

```
model.lm = lm(Y ~ X)
lm.coeffs = model.lm$coefficients[2:16]
lm.coeffs
```

```
##         Xlon         Xlat        Xcrim          Xzn       Xindus        Xchas
## -0.032316441  0.030245087 -0.097935969  0.118273098  0.011390378  0.071312253
##         Xnox          Xrm         Xage         Xdis         Xrad         Xtax
## -0.199703772  0.287232811  0.007564852 -0.321039342  0.290850755 -0.236526155
##     Xptratio           Xb       Xlstat
## -0.206804965  0.091235409 -0.417972819
```

```
modeldiff <- beta - lm.coeffs
modeldiff
```

```
##                  [,1]
## lon     -3.154922e-06
## lat     -3.198031e-06
## crim     7.597723e-06
## zn      -1.349918e-05
## indus   -4.053649e-05
## chas     4.964327e-06
## nox      8.510655e-06
## rm       6.515886e-06
## age     -4.432409e-06
## dis     -1.596258e-06
## rad     -1.008163e-04
## tax      1.121778e-04
## ptratio  5.128383e-06
## b       -4.999005e-07
## lstat    3.689583e-06
```

```
kable(cbind(beta, as.data.frame(lm.coeffs), modeldiff), digits=9, caption="Result Table for Beta and LM
```

| | beta | lm.coeffs | modeldiff |
|---|---|---|---|

Table 1: Result Table for Beta and LM

| | beta | lm.coeffs | modeldiff |
|---|---|---|---|
| lon | -0.032319596 | -0.032316441 | -0.000003155 |
| lat | 0.030241889 | 0.030245087 | -0.000003198 |
| crim | -0.097928372 | -0.097935969 | 0.000007598 |
| zn | 0.118259599 | 0.118273098 | -0.000013499 |
| indus | 0.011349842 | 0.011390378 | -0.000040536 |
| chas | 0.071317218 | 0.071312253 | 0.000004964 |
| nox | -0.199695261 | -0.199703772 | 0.000008511 |
| rm | 0.287239327 | 0.287232811 | 0.000006516 |
| age | 0.007560419 | 0.007564852 | -0.000004432 |
| dis | -0.321040938 | -0.321039342 | -0.000001596 |
| rad | 0.290749939 | 0.290850755 | -0.000100816 |
| tax | -0.236413977 | -0.236526155 | 0.000112178 |
| ptratio | -0.206799837 | -0.206804965 | 0.000005128 |
| b | 0.091234909 | 0.091235409 | -0.000000500 |
| lstat | -0.417969129 | -0.417972819 | 0.000003690 |

We set a maxiter of 5000 to limit run time, but with epsilon = 10^-6, the mylm_g function finishes in 1178 iterations.

The table shows the difference in our Beta values and the values found by running the default lm function provided by R.

Next test with a tighter epsilon of 10^-8. This converges in only 1909 iterations.

```
epsilon <- 10^-8
beta <- mylm_g(X, Y, delta, epsilon, maxiter)
```

```
## [1] "Delta: 0.100000 Epsilon: 0.000000010 took 1909 iterations"
```

```
modeldiff <- beta - lm.coeffs
kable(cbind(beta, as.data.frame(lm.coeffs), modeldiff), digits=9, caption="Result Table for Beta and LM
```

Table 2: Result Table for Beta and LM

| | beta | lm.coeffs | modeldiff |
|---|---|---|---|
| lon | -0.032316472 | -0.032316441 | -3.100e-08 |
| lat | 0.030245055 | 0.030245087 | -3.200e-08 |
| crim | -0.097935893 | -0.097935969 | 7.600e-08 |
| zn | 0.118272962 | 0.118273098 | -1.350e-07 |
| indus | 0.011389971 | 0.011390378 | -4.070e-07 |
| chas | 0.071312303 | 0.071312253 | 5.000e-08 |
| nox | -0.199703687 | -0.199703772 | 8.500e-08 |
| rm | 0.287232876 | 0.287232811 | 6.500e-08 |
| age | 0.007564807 | 0.007564852 | -4.500e-08 |
| dis | -0.321039359 | -0.321039342 | -1.700e-08 |
| rad | 0.290849743 | 0.290850755 | -1.012e-06 |
| tax | -0.236525028 | -0.236526155 | 1.127e-06 |
| ptratio | -0.206804914 | -0.206804965 | 5.100e-08 |
| b | 0.091235404 | 0.091235409 | -5.000e-09 |

| | beta | lm.coeffs | modeldiff |
|---|---|---|---|
| lstat | -0.417972782 | -0.417972819 | 3.700e-08 |

## Bonus Question [5 Points] The Non-scaled Version

When we do not scale and center the data matrix (both X and Y), it could be challenging to obtain a good solution. Try this with your code, and comment on what you observed and explain why. Can you think of a way to calculate the beta parameters on the original scale using the solution from the previous question? To earn a full 5 point bonus, you must provide a rigors mathematical derivation and also validate that by comparing it to the `lm()` function on the original data.

Answer:

When running the mylm_g function on unscaled data, the beta coefficients quickly diverge and become unstable, resulting in near infinite coeffecients after only running a few dozen iterations.

The problem is we are not calculating an intercept (B-zero) and the data scale varies between parameters by over 4 orders of magnitude, so the gradient coefficient calculation becomes unstable and diverges.

We can use the beta parameters from the scaled solution to calculate the proper beta parameters on unscaled data by "unwinding" the operations performed by scaling and calculating an intercept. The code below calculates the the unscaled beta coefficients directly from the scaled beta coefficients and compares them to the coefficients calculated from the default R lm library.

The R code is presented in its entirety below since the bonus question was calculated and tested in a separate R file in my RStudio project:

```
library(mlbench)

euc.dist <- function(x1, x2) {
  sqrt(sum((x1 - x2) ^ 2))
}

mylm_g <- function(x, y, delta, epsilon, maxiter) {
  x <- cbind(rep(1,nrow(x)), x)

  beta_old <- matrix(1,nrow=ncol(x), ncol=1)
  beta_new <- matrix(1,nrow=ncol(x), ncol=1)
  iter <- 1
  beta_diff <- 1

  while (beta_diff > epsilon && iter <= maxiter) {

    beta_new <- beta_old - (delta / nrow(x)) * (t(x) %*% (x %*% beta_old - y))
    beta_diff <- euc.dist(beta_old, beta_new)
    beta_old <- beta_new

    iter <- iter + 1
  }

  return(beta_new)
}

data(BostonHousing2)
X = BostonHousing2[, !(colnames(BostonHousing2) %in% c("medv", "town", "tract", "cmedv"))]
```

```r
X_orig = data.matrix(X)
X_scaled = scale(data.matrix(X))
Y_orig = as.vector(BostonHousing2$cmedv)
Y_scaled = as.vector(scale(BostonHousing2$cmedv))

delta <- 0.1
epsilon <- 10^-9
maxiter <- 10000

beta <- mylm_g(X_scaled, Y_scaled, delta, epsilon, maxiter)

lm.model <- lm(Y_orig ~ X_orig)

#-------------------------------------------------------------------------------
# Convert beta back to original coords by reversing operations and calculating
# beta-zero, the intercept
#-------------------------------------------------------------------------------
beta_convert <- beta[c(2:nrow(beta)),] * as.vector(apply(as.matrix(Y_orig),2,sd)) /
                                        as.matrix(apply(X_orig,2,sd))

beta0_convert <- beta[1,1] + mean(Y_orig) - sum(beta[c(2:nrow(beta)),] *
                              (as.matrix(apply(X_orig,2,mean)) *
                               as.vector(apply(as.matrix(Y_orig),2,sd)) /
                               as.matrix(apply(X_orig,2,sd))))

convertedBeta <- matrix(c(beta0_convert, beta_convert))

difference <- abs(lm.model$coefficients - convertedBeta)
kable(cbind(convertedBeta, as.data.frame(lm.model$coefficients), difference), digits=9, caption="Result
```

Table 3: Result Table for Calculated Beta from scaled values and LM

| | convertedBeta | lm.model$coefficients | difference |
|---|---|---|---|
| (Intercept) | -4.375739e+02 | -4.375739e+02 | 9.214e-06 |
| X_origlon | -3.935202e+00 | -3.935202e+00 | 3.820e-07 |
| X_origlat | 4.495441e+00 | 4.495441e+00 | 4.780e-07 |
| X_origcrim | -1.045469e-01 | -1.045469e-01 | 8.000e-09 |
| X_origzn | 4.656475e-02 | 4.656476e-02 | 5.000e-09 |
| X_origindus | 1.524529e-02 | 1.524535e-02 | 5.400e-08 |
| X_origchas | 2.578020e+00 | 2.578020e+00 | 1.800e-07 |
| X_orignox | -1.582458e+01 | -1.582458e+01 | 6.690e-07 |
| X_origrm | 3.753712e+00 | 3.753712e+00 | 8.500e-08 |
| X_origage | 2.467658e-03 | 2.467659e-03 | 1.000e-09 |
| X_origdis | -1.399927e+00 | -1.399927e+00 | 7.000e-09 |
| X_origrad | 3.067144e-01 | 3.067145e-01 | 1.070e-07 |
| X_origtax | -1.288632e-02 | -1.288633e-02 | 6.000e-09 |
| X_origptratio | -8.771212e-01 | -8.771212e-01 | 2.200e-08 |
| X_origb | 9.176196e-03 | 9.176196e-03 | 0.000e+00 |
| X_origlstat | -5.374411e-01 | -5.374411e-01 | 5.000e-09 |

Mathematically, since we scaled both the X and Y variables to calculate the original Beta coefficients, to

recover the Beta for the unscaled data, we simply multiple each X Beta coefficient by the standard deviation of Y by the standard deviation of X, i.e.

Beta_convert = Beta_scaled * SD_origY / SD_origX

To calculate the Beta_zero intercept for the unscaled data is a little more complicated, but effectively we are just reversing the normalization operations on the scaled beta coefficients. We have to add the original intercept from scaled data (which is effectively zero) to the mean of the original Y unscaled data, i.e.:

intercept = scaled_intercept (i.e. 0) + y_orig_mean

and then subtract the summation of the scaled beta values multiplied by x_orig_mean * SD_origY/SD_origX

itercept = scaled_intercept (i.e. 0) + y_orig_mean - sum(orig_beta$x_origmean$SD_origY/SD_origX)