Question 1

Author: Bin Dong

Email: bindong2@iillinois.edu

a.

```
# Clear entire workspace
rm(list=ls())

# l2 norm distance
mydist <- function(x1, x2) {
    #data <- t(apply(x2, 1, function(x) x-x1))
    data <- sweep(x2, 2, x1);
    return (sqrt(apply((data)^2, 1, sum)))
}

# Get most frequent
getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}

# KNN
myknn <- function(xtest, xtrain, ytrain, k) {
    # Initialte a vector for storing results.
    result <- rep(0, times = nrow(xtest))
    for( c in (1:nrow(xtest)))
    {
      dimMatrix <- mydist(xtest[c,],xtrain)
      # Sort by indices and take the order of them (top k)
      top_indices = order(dimMatrix, decreasing=FALSE)[1:k]
      result[c] <- mean(ytrain[top_indices])
    }
    return <- result
}
```

```
# This section is only a test of functionality of myknn.
# xtest is 2 by 5 matrix
xtest <- matrix(1:10, ncol=5, byrow=TRUE)
# xtrain is 20 by 5 matrix
xtrain <- matrix(1:100, ncol=5, byrow=TRUE)
# ytrain is 20 length vector
ytrain <- c(1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0)

# a is expected as a 2 length vector
a<- myknn(xtest, xtrain, ytrain, 3)
print(a)
```

```
## [1] 0.6666667 0.6666667
```

b.

```
# Initialize random seed
set.seed(1)

# Use library MASS for generating multi dimension normal distribution
library(MASS)

# Mean
my_mean <- c(1,2,3,4,5)
my_var <- matrix(rep(0, 25), nrow=5, ncol=5)
for( i in c(1:5))
{
  for(j in c(1:5))
  {
    my_var[i,j] = 0.5^(abs(i-j))
  }
}

x <- mvrnorm(1000,my_mean,my_var)

y <- x[,1]+x[,2]+(x[,3]-2.5)^2+rnorm(1,0,1)
```

The first three data are:

```
print(x[1:3,])
```

```
##            [,1]     [,2]     [,3]     [,4]     [,5]
## [1,]  2.0770490 3.555163 2.641969 3.902436 5.108741
## [2,]  2.4780195 2.161175 2.188487 2.796376 5.330744
## [3,] -0.1413538 2.630428 4.666608 4.493909 5.698190
```

```
print(y[1:3])
```

```
## [1] 4.135994 3.219862 5.666890
```

c.

```
split_count <- 400
x_train <- x[1:split_count,]
x_test <- x[(split_count+1):1000,]
y_train <- y[1:split_count]
y_test <- y[(split_count+1):1000]

predictions <- myknn(x_test, x_train, y_train, 5)
```

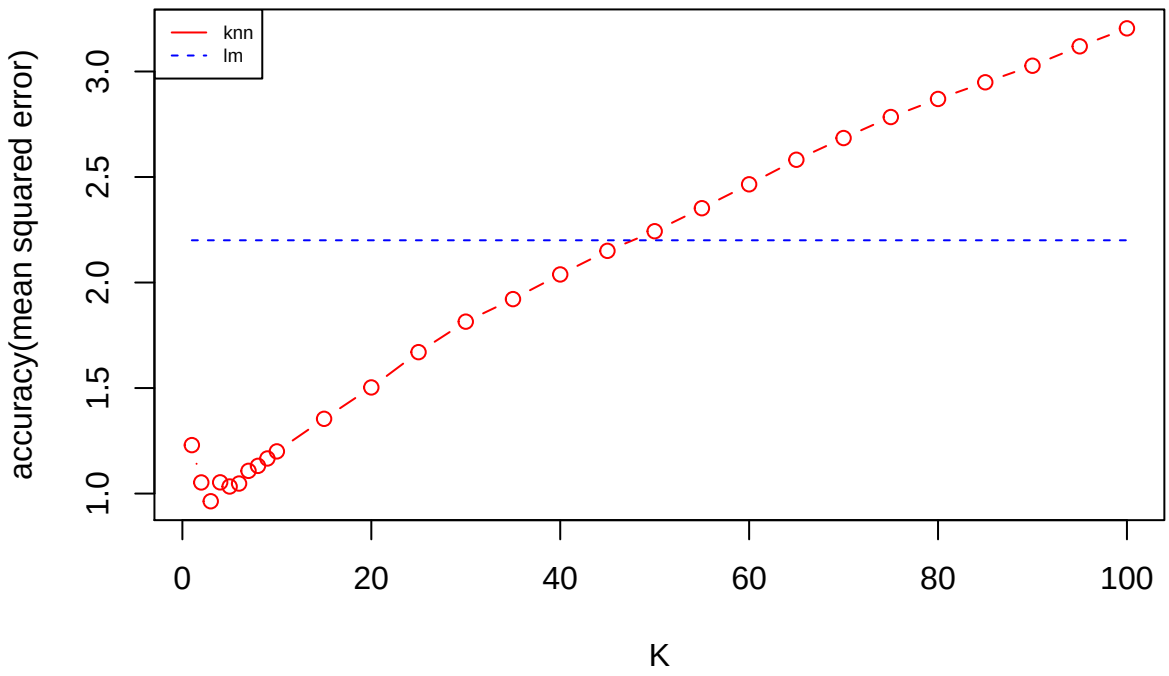
The MSE when k=5 is:

```r
sum((predictions - y_test)^2)/600
```

```
## [1] 1.033578
```

d.

```r
k_collection <- c(1,2,3,4,5,6,7,8,9,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100)

y_collection <- rep(0, times = length(k_collection))


# Collecting MSE for different k
y_index <- 1

for(k in k_collection)
{
  predictions <- myknn(x_test, x_train, y_train, k)
  y_collection[y_index] <- sum((predictions - y_test)^2)/600
  y_index <- y_index+1
}

# Fit data with a linear model.
myfit <- lm(y_train~x_train)

xtest.df <- data.frame(x_train=I(x_test))
lmprediction <- predict(myfit, newdata=xtest.df)

lm.mse <- sum((y_test- lmprediction)^2)/nrow(x_test)

lm_benchmark = rep(lm.mse, times = length(k_collection))

# Plot data. y_collection is mse of knn and lm_benchmark is mse of linear model.
plot(k_collection, y_collection,type="b", xlab = "K", ylab="accuracy(mean squared error)", col ="red",

lines(k_collection, lm_benchmark, type="l", col="blue", lty=2)
legend("topleft", cex=.6, c("knn", "lm"), col=c("red", "blue"), lty=1:2)
```



Question 2

a.

```
rm(list=ls())

# Calculate gradient
gradient <- function (x, y, beta)
{
  return <- as.vector(t(x%*%beta-y)%*%x/nrow(x))
}

# optimization function
mylm_g <- function(X, Y, delta, epsilon, maxiter)
{
  beta <- rep(0, times=ncol(X))
  epsilon <- rep(epsilon, times=ncol(X))
  iter <- 0
  for(i in c(1:maxiter)){
    oldBeta <- beta;

    newBeta <- oldBeta - gradient(X,Y,oldBeta)*delta
```

```
    if(all(newBeta-oldBeta < epsilon))
    {

      cat("stopped after", i, "iterations", "\n")
      break;
    }
    beta<-newBeta

  }
  return <- beta;
}
```

b.

```
library(mlbench)
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))
beta = rep(0, times=15)

result <- mylm_g(X, Y, 0.1, 0.000001, 10000)
```

```
## stopped after 1167 iterations
```

```
print(result)
```

```
##  [1] -0.032321375  0.030240037 -0.097924015  0.118251851  0.011326542
##  [6]  0.071320080 -0.199690479  0.287243072  0.007557858 -0.321041946
## [11]  0.290691954 -0.236349417 -0.206796921  0.091234619 -0.417966989
```

```
# Compare with build-in lm() function
myfit <- lm(Y~X)
print(myfit)
```

```
##
## Call:
## lm(formula = Y ~ X)
##
## Coefficients:
## (Intercept)         Xlon         Xlat        Xcrim          Xzn
##  -7.634e-16   -3.232e-02    3.025e-02   -9.794e-02    1.183e-01
##      Xindus        Xchas         Xnox          Xrm         Xage
##   1.139e-02    7.131e-02   -1.997e-01    2.872e-01    7.565e-03
##        Xdis         Xrad         Xtax     Xptratio           Xb
##  -3.210e-01    2.909e-01   -2.365e-01   -2.068e-01    9.124e-02
##      Xlstat
##  -4.180e-01
```

After comparing parameters, they are very close to each other between built-in lm() function and my implementation.

Bonus Question

```
library(mlbench)
data(BostonHousing2)
X = BostonHousing2[, !(colnames(BostonHousing2) %in% c("medv", "town", "tract", "cmedv"))]
X = data.matrix(X)
Y = as.vector(BostonHousing2$cmedv)
beta = rep(0, times=15)

result <- mylm_g(X, Y, 0.000001, 0.000001, 100000)

print(result)
```

```
##  [1] -0.362815164  0.207952011 -0.079764050  0.053784944 -0.048726057
##  [6]  0.034544162  0.001922380  0.223111205  0.055873592 -0.206926867
## [11]  0.231593673 -0.013901235 -0.267899056  0.007062945 -0.823286719
```

```
# Compare with build-in lm() function
myfit <- lm(Y~X)
print(myfit)
```

```
##
## Call:
## lm(formula = Y ~ X)
##
## Coefficients:
## (Intercept)         Xlon         Xlat        Xcrim          Xzn
##  -4.376e+02   -3.935e+00    4.495e+00   -1.045e-01    4.656e-02
##       Xindus        Xchas         Xnox          Xrm         Xage
##   1.525e-02    2.578e+00   -1.582e+01    3.754e+00    2.468e-03
##         Xdis         Xrad         Xtax     Xptratio           Xb
##  -1.400e+00    3.067e-01   -1.289e-02   -8.771e-01    9.176e-03
##       Xlstat
##  -5.374e-01
```

The first observation is that it cannot converge when delta is large.

THe second observation is that it is not converging to the same result as lm.

The reasons behind this behavior are:

1. The way i used to converge gradient is to substract a uniform vector from beta. When data are not scaled, their parameters are in different orders of magnitude. Substracting a uniform vector from such parameters can really slow down the converging of large parameters.
2. Since large parameters are dominating the gradient, so small parameters will not converge if the step is set too large.