CS598 : HW1_apoorva6

Apoorva

9/8/2019

Contents

Question 1 - KNN

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

Part a.

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 ‘2 norm to evaluate the distance between two points. Please note that you cannot use any additional R package within this function.

```
#Function to find Euclidean distance between test and train variables
euclideanDist <- function(a, b){
  eu_dist = 0
  for(k in c(1:(length(a))))
  {
    eu_dist = eu_dist + (a[[k]]-b[[k]])^2
  }
  eu_dist = sqrt(eu_dist)
  return(eu_dist)
}

# My Knn function
myknn  <- function(xtest, xtrain, ytrain, k)
{
pred =c()

for(i in c(1:nrow(xtest)))
{
  dist = c()
  for(j in c(1:nrow(xtrain)))
  {
    dist <- c(dist, euclideanDist(xtest[i,], xtrain[j,]))

  }
  dist_df <- data.frame(dist = dist,y = ytrain)
  dist_eu <- dist_df[order(dist_df$dist),]
  dist_eu_k <- dist_eu[1:k,]
  avg_dist <- mean(dist_eu_k[,"y"])
  pred <- c(pred,avg_dist)
}
```

```
mean_Sq_error <-  mean((pred - test[,6])^2)
return(mean_Sq_error)
}
```

Part b.

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 ϵ 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
mu <- c(1,2,3,4,5)
# Creating the matrix for sigma
calc2 <- vector()
for (i in 1:5)
{
  for (j in 1:5)
  {
    calc <- 0.5 ^ abs(i-j)
    calc2 <- c(calc2, calc)
  }
}
Sigma <- matrix(calc2,5,5)

X <- mvrnorm(n = 1000, mu, Sigma)
colnames(X)<- c("X1","X2","X3","X4","X5")

# Generating outcome values 'Y'
eps <- rnorm(1000,mean=0,sd=1)
Y <- X[,"X1"] + X[,"X2"] +((X[,"X3"] - 2.5)^2) + eps

data <- cbind(X,Y)
data[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
```

Part c.

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

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

```
train <- data[1:400,]
#nrow(train)
test <- data[401:1000,]
#nrow(test)

# The function Myknn returns MSE value
myknn(test[,-6], train[,-6],train[,6],5)
```

```
## [1] 2.191387
```

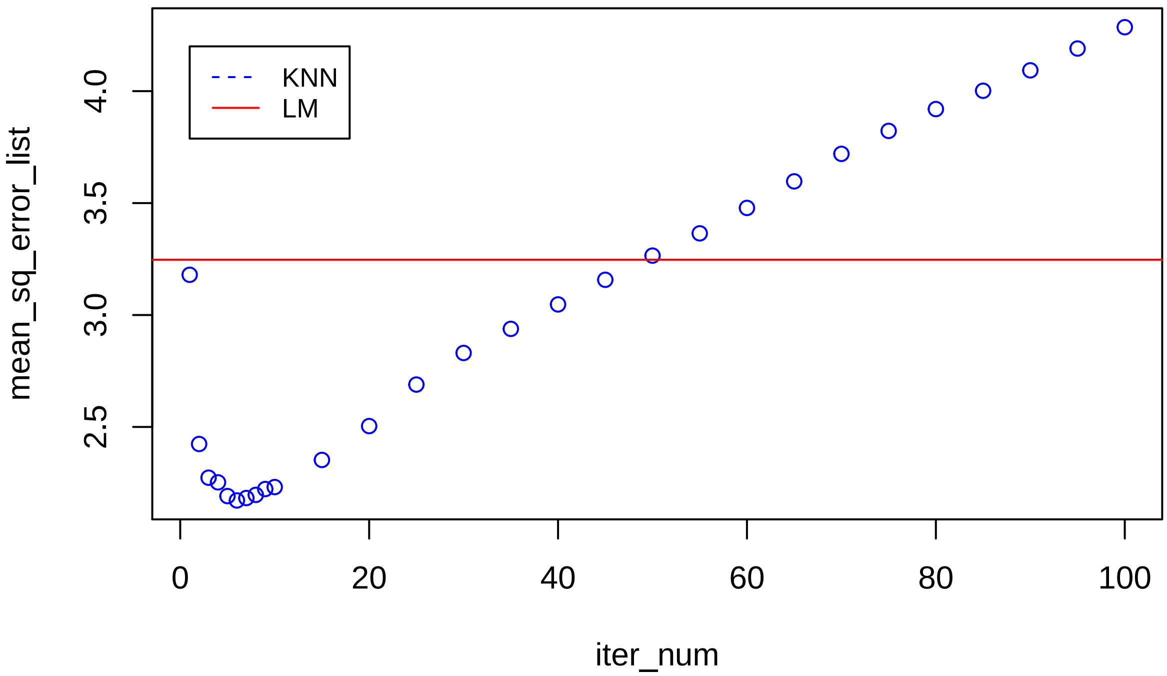
Part d.

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.

```
#Regression function
reg <- lm(Y~., data=as.data.frame(train))
prediction_lm <- predict(reg,as.data.frame(test))
mean_Sq_error_lm <-  mean((prediction_lm - test[,6])^2)

# Note down MSE values for values of K ranging from 1 to 100 in steps of 5
mean_sq_error_list <- c()
iter_num <- 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)
for (i in iter_num)
{
  mse <- myknn(test[,-6], train[,-6],train[,6],i)
  mean_sq_error_list <- c(mean_sq_error_list,mse)
}

#Plot of MSE of the linear regression model and the knn model for different values of k
plot(mean_sq_error_list~iter_num, col = "blue",xlim = c(1,100))
abline(h=mean_Sq_error_lm, col="red")
legend(1, 4.2,legend = c("KNN", "LM"),
       col= c("blue", "red"),lty=2:1, cex=0.8)
```



Question 2 - 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 δ 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 ϵ, e.g., $\epsilon = 10^{-6}$. You should also set a maximum number of iterations to prevent excessively long runing time.

We will implement our function on the Boston Housing data from the mlbench package. 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.

```
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))
```

Part a.

Based on the above 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.

```
mylm_g <- function(x, y, delta, epsilon, maxitr)
{
  iter <- 1
  diff <- 10
  beta_old <- matrix(0, nrow=15)
  while((iter < maxitr) & (diff > epsilon))
  {
    doh_beta <- t(x) %*% ((x) %*% beta_old - y) / length(y)
    beta_new <- beta_old - (doh_beta * delta)
    diff <- sqrt(sum((beta_old-beta_new)^2))
    beta_old <- beta_new
    iter <- iter + 1
  }
  print(beta_old)
}

my_lm <- mylm_g(X,Y,0.1,(10^(-6)),400)
```

```
##                 [,1]
## lon     -0.033218658
## lat      0.029718413
## crim    -0.096323627
## zn       0.115461478
## indus    0.003275932
## chas     0.072240723
## nox     -0.197124711
## rm       0.288555314
## age      0.006776135
## dis     -0.320624437
## rad      0.270922202
## tax     -0.214677567
## ptratio -0.205533203
## b        0.091161291
## lstat   -0.417368861
```

Part b.

Compare your results with the lm() function on the same data. Experiment on different maxitr values to obtain a good solution

```
########################## Linear regression using library #################################
reg <- lm(Y~X)
lm_coef <- summary(reg)$coefficients[2:16,1]

########################## Comparision #################################
beta_coef <- data.frame(my_lm,lm_coef,my_lm-lm_coef)
colnames(beta_coef) <- c("GradientDescent","Lm","Difference")
beta_coef
```

```
##         GradientDescent           Lm    Difference
## lon        -0.033218658 -0.032316441 -9.022174e-04
## lat         0.029718413  0.030245087 -5.266747e-04
## crim       -0.096323627 -0.097935969  1.612342e-03
## zn          0.115461478  0.118273098 -2.811619e-03
## indus       0.003275932  0.011390378 -8.114446e-03
## chas        0.072240723  0.071312253  9.284695e-04
## nox        -0.197124711 -0.199703772  2.579061e-03
## rm          0.288555314  0.287232811  1.322503e-03
## age         0.006776135  0.007564852 -7.887173e-04
## dis        -0.320624437 -0.321039342  4.149052e-04
## rad         0.270922202  0.290850755 -1.992855e-02
## tax        -0.214677567 -0.236526155  2.184859e-02
## ptratio    -0.205533203 -0.206804965  1.271762e-03
## b           0.091161291  0.091235409 -7.411765e-05
## lstat      -0.417368861 -0.417972819  6.039576e-04
```