Assignment 1

Data Mining

Nilava Metya

October 05, 2023

Problem 1

Compare the classification performance of linear regression and k-nearest neighbor classification on the `zipcode` data. In particular, consider only the 2's and 3's, and$ k=1,3, 5, 7, and 15$. Write your own function for the k-nearest neighbor classification. Show both the training and test error for each choice of k. The zipcode data are available from the book website: https://hastie.su.domains/ElemStatLearn/data.html

My own k-Nearest Neighbor function

Two auxilliary functions: `L2_distance` and `nearest_nb()`

```r
#find L2 distance between two vectors v and u
L2distance = function(v,u){
  if (length(v) != length(u)){
    stop('Vectors are be of the different length')
  }
  else{
    return(sum(abs(u-v)))
  }
}

#find k nearest points neighbours from inp among x[1],...,x[n]
nearest_nb = function(x, inp, k, d=L2distance){
  if(ncol(x) != length(inp)){
    stop('Data have different number of variables')
  }
  dist = 1:nrow(x)
  for(i in 1:nrow(x)){
    dist[i] = d(x[i,],inp)
  }
  dist.sort = sort(dist)
  closest_k = dist.sort[1:k]
  knearest_ind = which(dist %in% closest_k)
  return(knearest_ind)
}
```

Finally defining the function

```r
#self-defined knn function
knn_own = function(trainX, testX, trainY, k){
  all_pred = c()
  for(i in 1:nrow(testX)){
    nbs = nearest_nb(trainX, testX[i,], k)
    prediction = mean(trainY[nbs])
    all_pred[i] = round(prediction,0)
  }
  return(all_pred)
}
```

Loading Data

We start by loading training data. Look at the columns which correspond only to 2 or 3 — these are the features. We extract out the 3's which is the output we want:

```r
X <- as.matrix(read.table(gzfile("./zip.train.gz")))
y2or3 <- which(X[, 1] == 2 | X[, 1] == 3)
train.X <- X[y2or3, -1]
train.Y <- X[y2or3, 1] == 3
```

We do the same thing as above for test data:

```r
X <- as.matrix(read.table(gzfile("./zip.test.gz")))
y2or3 <- which(X[, 1] == 2 | X[, 1] == 3)
test.X <- X[y2or3, -1]
test.Y <- X[y2or3, 1] == 3
```

Linear Regression

Next we fit a linear model based on the training data (X, Y):

```r
lr.fit <- lm(train.Y ~ train.X)
```

Now that we have a model, we look at predictions that the model makes when the input features are taken from testing data and training data. This will be a number between 0 and 1, but since we are dealing with classification, we round it off to the nearest integer.

```r
prediction_test <- round(cbind(1, test.X) %*% lr.fit$coef,0)
prediction_train <- round(cbind(1, train.X) %*% lr.fit$coef,0)
```

Obtain the errors for linear regression for testing data and training data:

```r
errortest.lr <- mean(prediction_test != test.Y)*100
errortrain.lr <- mean(prediction_train != train.Y)*100
```

k-Nearest Neighbours

We want to run kNN for $k \in \{1, 2, \cdots, 20\}$.

```r
k <- 1:20
```

First we start with the inbuilt kNN. We basically do the same as linear regression - fir the model based on training data, test it on the test data and train data to get the testing error and training error respectively.

```r
library(class)
errortest.knn <- numeric(length(k))
errortrain.knn <- numeric(length(k))
for (i in 1:length(k)) {
  prediction_test <- knn(train.X, test.X, train.Y, k[i])
  prediction_train <- knn(train.X, train.X, train.Y, k[i])
  errortest.knn[i] <- mean(prediction_test != test.Y)*100
  errortrain.knn[i] <- mean(prediction_train != train.Y)*100
}
```

Next we execute the *k*NN function that I wrote (including error calculation).

```
errortest.knn_own <- numeric(length(k))
errortrain.knn_own <- numeric(length(k))
for (i in 1:length(k)) {
  prediction_test <- knn_own(train.X, test.X, train.Y, k[i])
  prediction_train <- knn_own(train.X, train.X, train.Y, k[i])
  errortest.knn_own[i] <- mean(prediction_test != test.Y)*100
  errortrain.knn_own[i] <- mean(prediction_train != train.Y)*100
}
```

Tabulating

Now that we have found the errors, we construct a table and print it:

```
error <- matrix(c(errortest.lr, errortest.knn, errortest.knn_own, errortrain.lr, errortrain.knn, errort
colnames(error) <- c("Test Error(%)", "Train Error(%)")
rownames(error) <- c("Linear Regression", paste("inbuilt k-NN with k =", k),
                     paste("own k-NN with k =", k))
print(error)
```

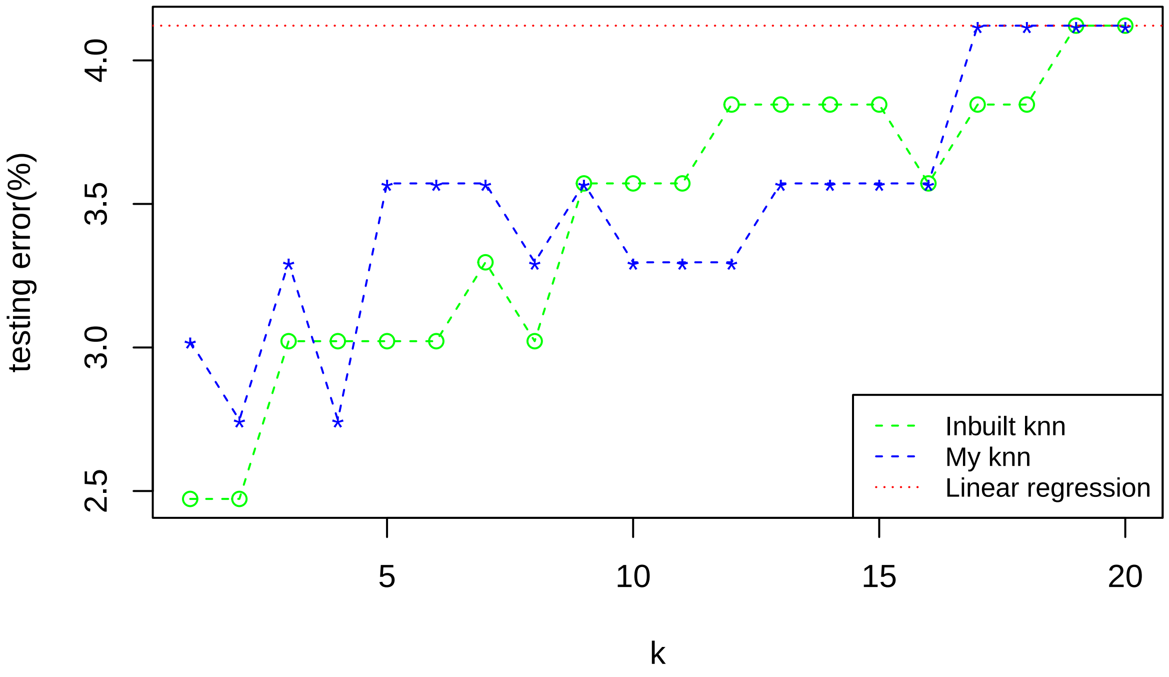
```
##                            Test Error(%) Train Error(%)
## Linear Regression               4.120879      0.5759539
## inbuilt k-NN with k = 1         2.472527      0.0000000
## inbuilt k-NN with k = 2         2.472527      0.6479482
## inbuilt k-NN with k = 3         3.021978      0.5039597
## inbuilt k-NN with k = 4         3.021978      0.7199424
## inbuilt k-NN with k = 5         3.021978      0.5759539
## inbuilt k-NN with k = 6         3.021978      0.6479482
## inbuilt k-NN with k = 7         3.296703      0.6479482
## inbuilt k-NN with k = 8         3.021978      0.7919366
## inbuilt k-NN with k = 9         3.571429      0.9359251
## inbuilt k-NN with k = 10        3.571429      0.7919366
## inbuilt k-NN with k = 11        3.571429      0.8639309
## inbuilt k-NN with k = 12        3.846154      0.7919366
## inbuilt k-NN with k = 13        3.846154      0.8639309
## inbuilt k-NN with k = 14        3.846154      0.8639309
## inbuilt k-NN with k = 15        3.846154      0.9359251
## inbuilt k-NN with k = 16        3.571429      1.0799136
## inbuilt k-NN with k = 17        3.846154      1.0799136
## inbuilt k-NN with k = 18        3.846154      1.0799136
## inbuilt k-NN with k = 19        4.120879      1.0799136
## inbuilt k-NN with k = 20        4.120879      1.2958963
## own k-NN with k = 1             3.021978      0.0000000
## own k-NN with k = 2             2.747253      0.5039597
## own k-NN with k = 3             3.296703      0.6479482
## own k-NN with k = 4             2.747253      0.5039597
## own k-NN with k = 5             3.571429      0.5759539
## own k-NN with k = 6             3.571429      0.5759539
## own k-NN with k = 7             3.571429      0.7199424
## own k-NN with k = 8          3.296703      0.7199424
## own k-NN with k = 9          3.571429      0.7919366
## own k-NN with k = 10         3.296703      0.7199424
## own k-NN with k = 11         3.296703      0.7199424
## own k-NN with k = 12         3.296703      0.6479482
## own k-NN with k = 13         3.571429      0.7199424
## own k-NN with k = 14         3.571429      0.7919366
## own k-NN with k = 15         3.571429      1.0079194
## own k-NN with k = 16         3.571429      1.0079194
## own k-NN with k = 17         4.120879      1.2239021
## own k-NN with k = 18         4.120879      1.0079194
## own k-NN with k = 19         4.120879      1.1519078
## own k-NN with k = 20         4.120879      1.1519078
```

Plotting

I also plotted the errors for my kNN function, the inbuilt kNN function and the linear regression.

Here's the plot for the testing error:

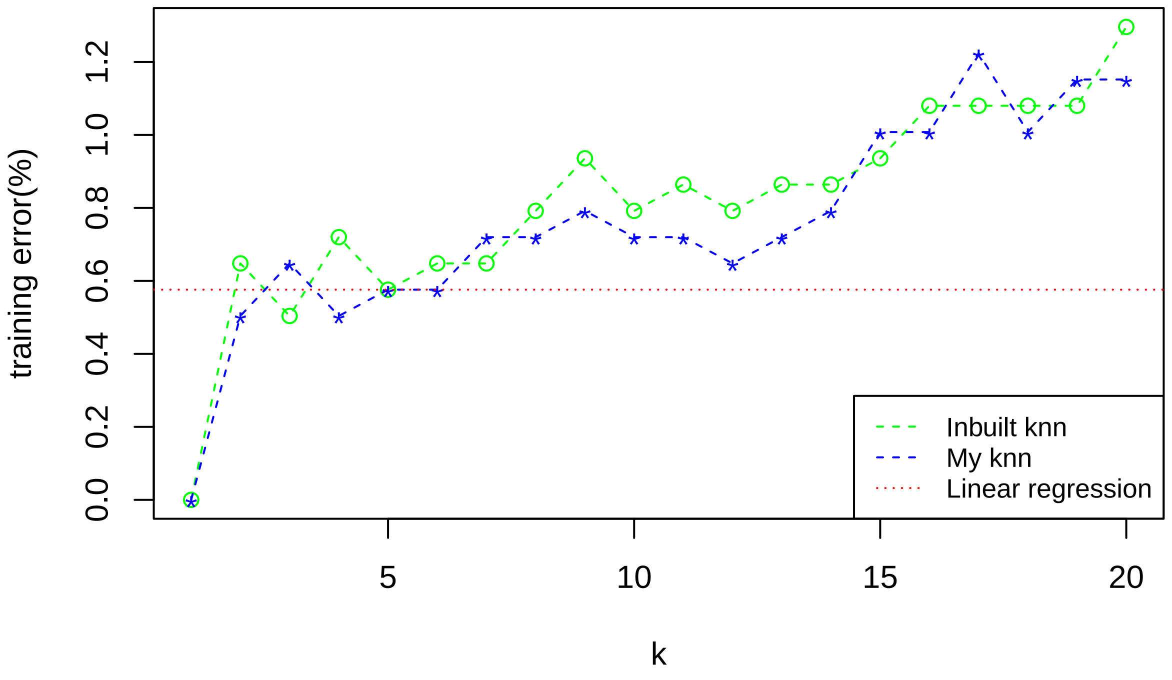
```r
y.limit = c(min(c(errortest.knn_own, errortest.knn, errortest.lr)),
            max(c(errortest.knn_own, errortest.knn,errortest.lr)))
plot(k,errortest.knn, type='o', lty = 2, ylim=y.limit, col="green", ylab="testing error(%)")
points(k, errortest.knn_own, type="o", lty=2, pch = "*", col="blue")
abline(h = errortest.lr, col="red", lty = 3)
legend("bottomright", legend=c("Inbuilt knn", "My knn", "Linear regression"),
       col=c("green", "blue","red"), lty=c(2,2,3), cex=0.8)
```



And here's the plot for the training error:

```r
y.limit = c(min(c(errortrain.knn_own, errortrain.knn,errortrain.lr)),
            max(c(errortrain.knn_own, errortrain.knn,errortrain.lr)))
plot(k,errortrain.knn, type='o', lty = 2, ylim=y.limit, col="green", ylab="training error(%)")
points(k, errortrain.knn_own, type="o", lty=2, pch = "*", col="blue")
```

```r
abline(h = errortrain.lr, col="red", lty = 3)
legend("bottomright", legend=c("Inbuilt knn", "My knn", "Linear regression"),
       col=c("green", "blue","red"), lty=c(2,2,3), cex=0.8)
```



As we can see, the *k*NN classifier produces less errorsome results on the testing dataset as compared to when we fit a linear model. Also, there's small result difference between my function and the inbuilt function, but the overall trend is the same. Additionally, my function performs much slower than the inbuilt function.

Problem 2

Consider the linear regression model with p parameters, fit by least squares to a set of training data $(x_1, y_1), \cdots, (x_N, y_N)$ draw from the population distribution where $x_1 \sim N_p(0, \Sigma)$ and $y_i = x_i^T \beta + \epsilon_i$ with $\epsilon_i \sim N(0, \sigma^2)$. Please use one method to construct confidence sets for β. Illustrate your method by simulating the data with $N = 100, p = 1, \Sigma = 2$ and $\sigma^2 = 1$. If you have more than one practical methods of constructing confidence sets for β and you can illustrate them by your simulations, you will obtain bonus points.

Getting the confidence interval

This code takes input the parameter β, samples the data points x_i, ϵ_i as mentioned in the problem, obtains the y_i, and gets an estimate $\hat{\beta} = \dfrac{\sum(x_i - \overline{x})(y_i - \overline{y})}{\sum(x_i - \overline{x})^2}$ for β. The interval is gotten by considering the error $\frac{\hat{\sigma} \cdot z_{1-\frac{\alpha}{2}}}{||\boldsymbol{x}||}$. So, our function outputs these two values because the confidence interval is $\left(\hat{\beta} - \dfrac{\hat{\sigma} \cdot z_{1-\frac{\alpha}{2}}}{||\boldsymbol{x}||}, \hat{\beta} + \dfrac{\hat{\sigma} \cdot z_{1-\frac{\alpha}{2}}}{||\boldsymbol{x}||}\right)$.

```r
get_confidence <- function(b, std_x, std_e, alpha){
  z <- qnorm(1-alpha/2)
  x <- rnorm(N,0,std_x)
  x.mean <- mean(x)
  e <- rnorm(N,0,std_e)
  y <-  b * x+ e
  y.mean <- mean(y)
  b_hat <- sum((x-x.mean) %*% (y-y.mean))/sum((x-x.mean)^2)
  error <- z * sqrt(var(e))/sqrt(sum((x-x.mean)^2))
  return(c(b_hat,error))
}
```

We run the above code for various values of β and see whether it belongs to the confidence interval or not. At the end, we report the fraction of cases where it did belong to the interval.

First we initialize our values for the simulation:

```r
beta <- seq(-1000, 1000, by=0.1)
N <- 100
s_x <- 2
s_error <- 1
alpha <- 0.05
```

Now we call the function to run the simulation and keep track of which β's are in the interval.

```r
total <- 0
for(i in beta){
  c <- get_confidence(i, s_x, s_error, alpha)
  if(i > c[1]-c[2] & i < c[1]+c[2])
    total <- total + 1
}
```

Finally we report this ratio.

```r
print(paste("Percentage of cases where beta landed in the interval:",
            (100*total/length(beta))))
```

```
## [1] "Percentage of cases where beta landed in the interval: 95.0852457377131"
```

I did this many times with different significance levels α's. Every time, this above ratio was close to $1 - \alpha$.