# STAT 542: Homework 1

*Spring 2020, by Yifan Shi (yifans16)*

*Due: Monday, Feb 3 by 11:59 PM*

## Contents



## Question 1 [70 Points] KNN

For this question, you **cannot** use (load) any additional `R` package. Complete the following steps.

### a. [15 points]

Generate the 1000 training data from the following multivaraite Normal distribution:

$$X \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where $\boldsymbol{\mu} = (0, 1, 2)^T$ and $\boldsymbol{\Sigma}$ is a covariance matrix with diagnal elements 1 and off-diagnal elements 0.5. Show with a rigorous derivation that your code will indeed generate observations from this distribution. Then, $Y$ is generated from

$$Y = 0.25 \times X_1 + 0.5 \times X_2 + X_3 + \epsilon,$$

with i.i.d. standard normal $\epsilon$. Set a random seed to 1. For the covariates, output the estimated mean and covariance matrix your data. Plot the outcome against your third variable $X^{(3)}$ and center the figure.

**Answer:**

```
set.seed(1)
n <- 1000
mu <- c(0,1,2)
sig <- matrix(c(1,.5,.5,.5,1,.5,.5,.5,1),3,3)
p <- length(mu)
#eigenvalues and eigenvectors of covariance matrix
e.val <- eigen(sig,symmetric = T)$values
e.vec <- eigen(sig,symmetric = T)$vectors
#data matrix and response vector
X <- matrix(rnorm(p*n),n)
X <- t(drop(mu) + e.vec %*% diag(sqrt(pmax(e.val,0)),p) %*% t(X))
Y <- .25*X[,1] + .5*X[,2] + X[,3] + rnorm(n,0,1)
```

The estimated mean of covariates is:

```
apply(X,2,mean)
```

```
## [1] 0.01834934 1.01322229 1.99696037
```

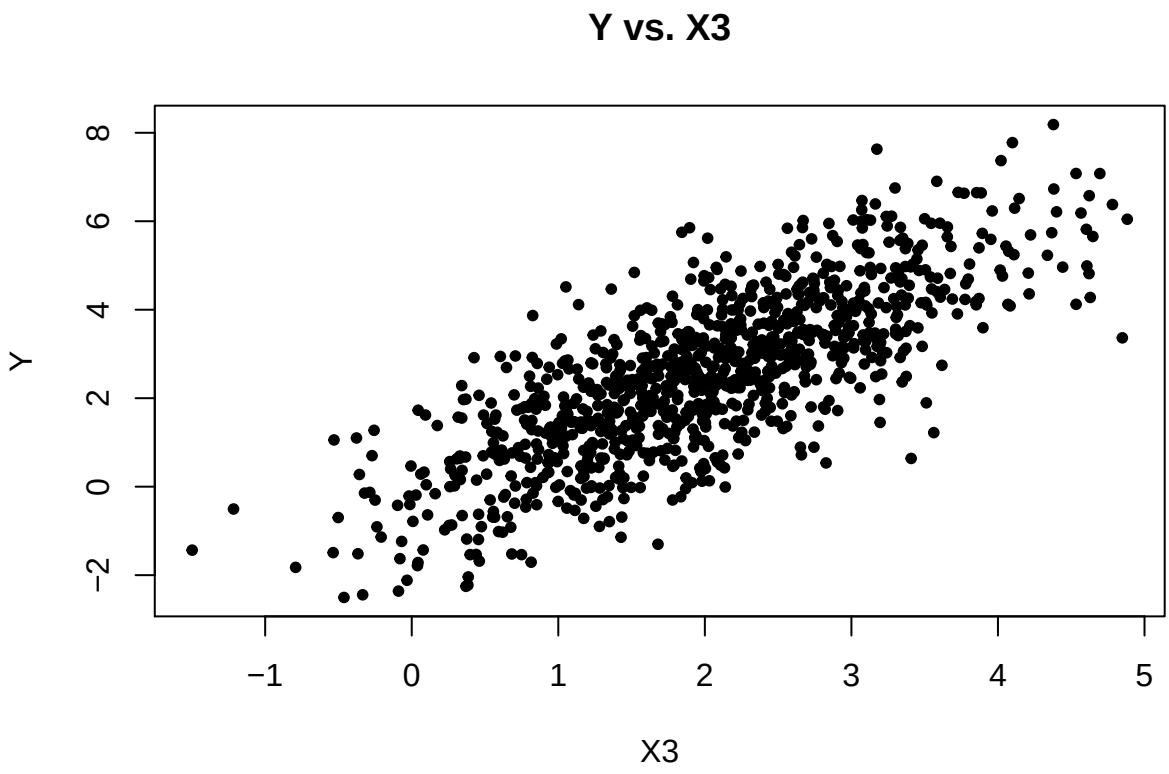
The covairance matrix of covariates is:

```
cov(X)
```

```
##           [,1]      [,2]      [,3]
## [1,] 1.0187554 0.5202598 0.5285779
## [2,] 0.5202598 1.1104559 0.5570781
## [3,] 0.5285779 0.5570781 1.0852619
```

The plot of Y against X3:

```
plot(X[,3],Y,xlab="X3",ylab="Y",main="Y vs. X3",pch=20)
```




**b. [15 Points]**

Write a function `myknn(xtest, xtrain, ytrain, k)` that fits a KNN model and predict 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 $\ell_1$ distance to evaluate the closeness between any two points. Please note that for this question, you can only use the base R (hence no additional package).

**Answer:**

```
myknn <- function(xtest,xtrain,ytrain,k){
  trn.len <- nrow(xtrain)
  tst.len <- nrow(xtest)
  #distance between each observation
  dist <- as.matrix(dist(rbind(xtest,xtrain),method="manhattan"))
  #select distance only between train and test
  dist <- dist[1:tst.len,(tst.len+1):(tst.len+trn.len)]
```

```
  #initialize ytest
  ytest <- rep(0,tst.len)
  for (i in 1:tst.len){
    #find the indexes of k cloest obs
    index <- as.data.frame(sort(dist[i,],index.return=T))[1:k,2]
    #define the ytest obs by its k neighbors
    ytest[i] <- mean(ytrain[index])
  }
  return(ytest)
}
```

### c. [10 Points]

Use the first 400 observations 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

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

**Answer:**

```
trn.x <- X[1:400,]
trn.y <- Y[1:400]
tst.x <- X[-1:-400,]
tst.y <- Y[-1:-400]
tst.yhat <- myknn(tst.x,trn.x,trn.y,5)
```

The mean squared error is:

```
mean((tst.y-tst.yhat)^2)
```

```
## [1] 1.343127
```

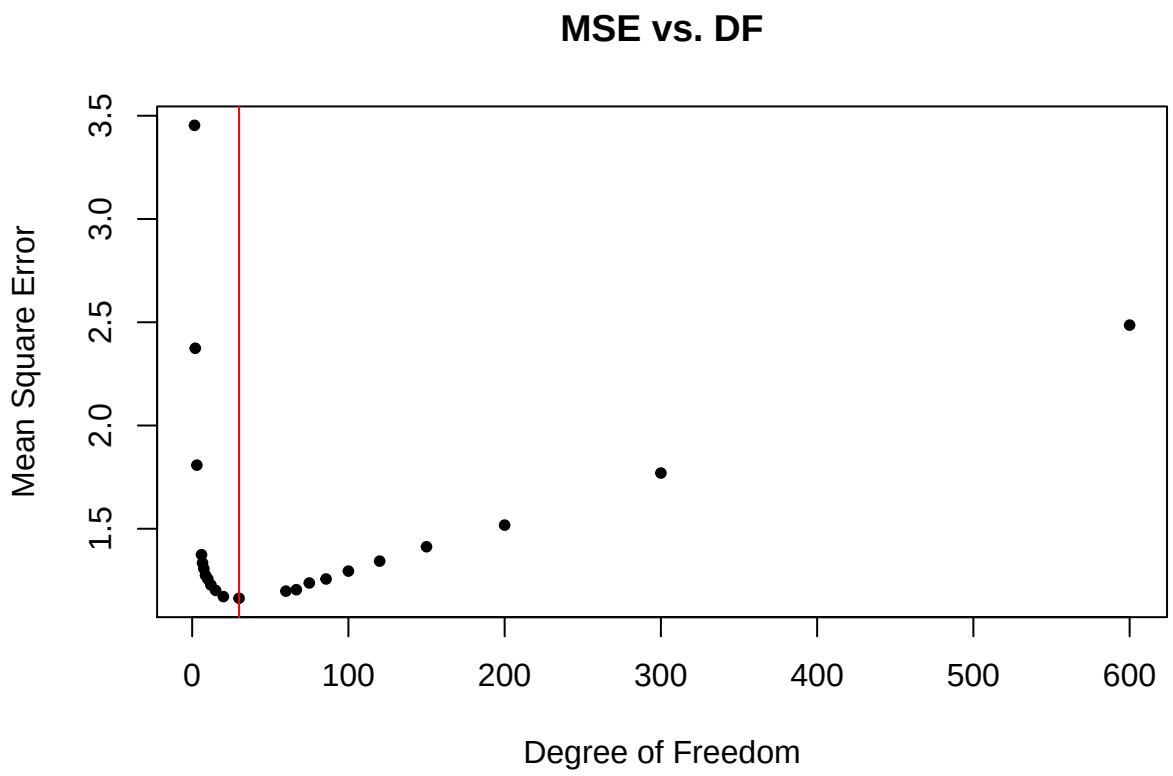
### d. [10 Points]

Consider $k$ being 1, 2, ..., 10, 20, 30, ..., 100, 200, 300, 400, 500. Use the degrees of freedom as the horizontal axis. Demonstrate your results in a single, easily interpretable figure with proper legends. What is your optimal tuning parameter?

**Answer:**

```
k <- c(c(1:9),c(1:9)*10,c(1:4)*100)
mse <- rep(0,length(k))
for (i in 1:length(k)){
  mse[i] <- mean((tst.y-myknn(tst.x,trn.x,trn.y,k[i]))^2)
}
```

```
plot(600/k,mse,xlab="Degree of Freedom",
     ylab="Mean Square Error",pch=20,main="MSE vs. DF")
abline(v=600/k[which.min(mse)],col="red")
```




```
k[which.min(mse)]
```

```
## [1] 20
```

```
mse[which.min(mse)]
```

```
## [1] 1.16343
```

Thus, my optimal tuning parameter is k=20

**e. [10 Points]**

Fit a linear regression to this data (with the same train/test split). What is the degree of freedom of this model? Is it better or worse than your optimal KNN? Why?

**Answer:**

```
df <- as.data.frame(cbind(X,Y))
trn.df <- df[1:400,]
tst.df <- df[-1:-400,]
m1 <- lm(Y~.,data=trn.df)
mean((tst.df[,4]-predict(m1,tst.df[,1:3]))^2)
```

```
## [1] 1.046746
```

The degree of freedom of this model is p = 4; It is better than my optimal kNN with smaller mean squared error; I think the reason is that the response Y here has linear relationship with the covariates.

**f. [10 Points]**

Try a new model and re-generate just your Y values:

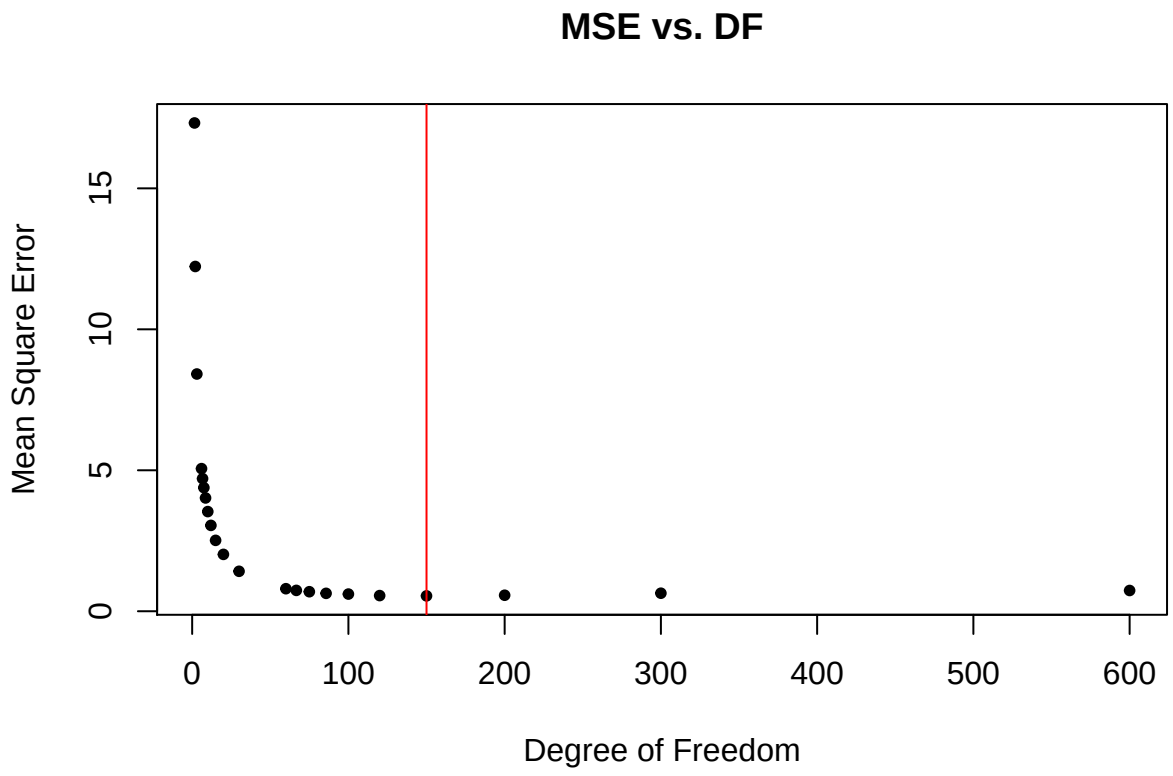
$$Y = 0.25 \times X_1 + 0.5 \times X_2 + X_3^2 + \epsilon$$

and redo your analysis of the KNN and linear regression. Which one is better? Why?

**Answer:**

```
set.seed(1)
Yn <- .25*X[,1] + .5*X[,2] + (X[,3])^2 + rnorm(n,0,1)
trn.yn <- Yn[1:400]
tst.yn <- Yn[-1:-400]
mse.n <- rep(0,length(k))
for (i in 1:length(k)){
  mse.n[i] <- mean((tst.yn-myknn(tst.x,trn.x,trn.yn,k[i]))^2)
}
```

```
plot(600/k,mse.n,xlab="Degree of Freedom",
     ylab="Mean Square Error",pch=20,main="MSE vs. DF")
abline(v=600/k[which.min(mse.n)],col="red")
```




```
k[which.min(mse.n)]
```

```
## [1] 4
```

```
mse.n[which.min(mse.n)]
```

```
## [1] 0.5447382
```

The optimal tuning parameter is k=4 here with the mean squared error 0.5447382

```
dfn <- as.data.frame(cbind(X,Yn))
trn.dfn <- dfn[1:400,]
tst.dfn <- dfn[-1:-400,]
m2 <- lm(Yn~.,data=trn.dfn)
mean((tst.dfn[,4]-predict(m2,tst.dfn[,1:3]))^2)
```

```
## [1] 2.304036
```

Here kNN model is better with smaller mean squared error because the new response Y does not have linear relationship with all the covariates.

## Question 2 [15 Points] Curse of Dimensionality

Following the previous analysis of the nonlinear model, let's consider a high-dimensional setting. For this problem, use the standard Euclidean distance. Also, keep the $Y$ values the same as part f. in the previous question. We consider two cases that both generate an additional set of 97 covariates:

- Generate another 97-dimensional covariate with all independent Gaussian entries
- Generate another 97-dimensional covariate using the formula $X^T A$, where $X$ is the original 3-dimensional vector, and $A$ is a $3 \times 97$ dimensional matrix that remains the same for all observations. Generate $A$ using i.i.d. uniform entries.

Fit KNN in both settings (with the total of 100 covariates) by repeating part d) of Question 1. Intuitively present your results and comment on your findings.

**Answer:**

```
#new function with l_2 norm
myknn_1 <- function(xtest,xtrain,ytrain,k){
  trn.len <- nrow(xtrain)
  tst.len <- nrow(xtest)
  #distance between each obs
  dist <- as.matrix(dist(rbind(xtest,xtrain),method="euclidean"))
  #select distance only between train and test
  dist <- dist[1:tst.len,(tst.len+1):(tst.len+trn.len)]
  #initialize ytest
  ytest <- rep(0,tst.len)
  for (i in 1:tst.len){
    #find the indexes of k cloest obs
    index <- as.data.frame(sort(dist[i,],index.return=T))[1:k,2]
    #define the ytest obs by its k neighbors
    ytest[i] <- mean(ytrain[index])
  }
  return(ytest)
}
```

Setting 1:

```
set.seed(1)
Xn1 <- cbind(X,matrix(rnorm(1000*97,0,1),1000,97))
trn.xn1 <- Xn1[1:400,]
tst.xn1 <- Xn1[-1:-400,]
mse.n1 <- rep(0,length(k))
```

```
for (i in 1:length(k)){
  mse.n1[i] <- mean((tst.yn-myknn_1(tst.xn1,trn.xn1,trn.yn,k[i]))^2)
}
```

The parameter k and the lowest mean squared error are:

```
k[which.min(mse.n1)]
```

```
## [1] 10
```

```
mse.n1[which.min(mse.n1)]
```

```
## [1] 9.72154
```

Setting 2:

```
set.seed(1)
Xn2 <- cbind(X,(X %*% matrix(runif(3*97,0,1),3,97)))
trn.xn2 <- Xn2[1:400,]
tst.xn2 <- Xn2[-1:-400,]
mse.n2 <- rep(0,length(k))
for (i in 1:length(k)){
  mse.n2[i] <- mean((tst.yn-myknn_1(tst.xn2,trn.xn2,trn.yn,k[i]))^2)
}
```

The parameter k and the lowest mean squared error are:

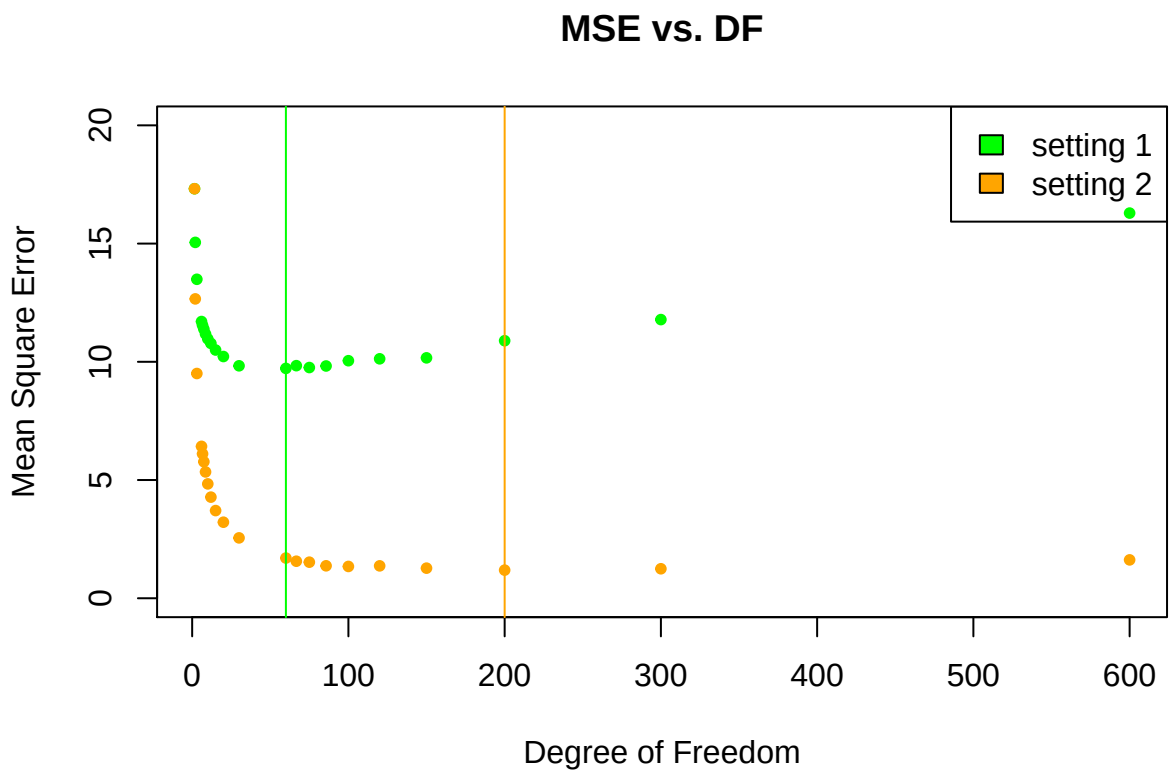
```
k[which.min(mse.n2)]
```

```
## [1] 3
```

```
mse.n2[which.min(mse.n2)]
```

```
## [1] 1.189917
```

```
plot(600/k,mse.n1,xlab="Degree of Freedom",ylab="Mean Square Error",
     pch=20,col="green",ylim=c(0,20),main="MSE vs. DF")
points(600/k,mse.n2,pch=20,col="orange")
abline(v=600/k[which.min(mse.n1)],col="green")
abline(v=600/k[which.min(mse.n2)],col="orange")
legend("topright",c("setting 1","setting 2"),fil=c("green","orange"))
```




In setting 1, I got k = 10 with mse = 9.72154 and in setting 2, I got k = 3 with mse = 1.189917; kNN perform better in the second scenario because the additonal columns in the second data matrix are the affine transformations of the original three column. Thus the principle components of the new data matrix in setting 2 are still the original three columns.

## Question 3 [15 Points] Classification of Handwritten Digit Data

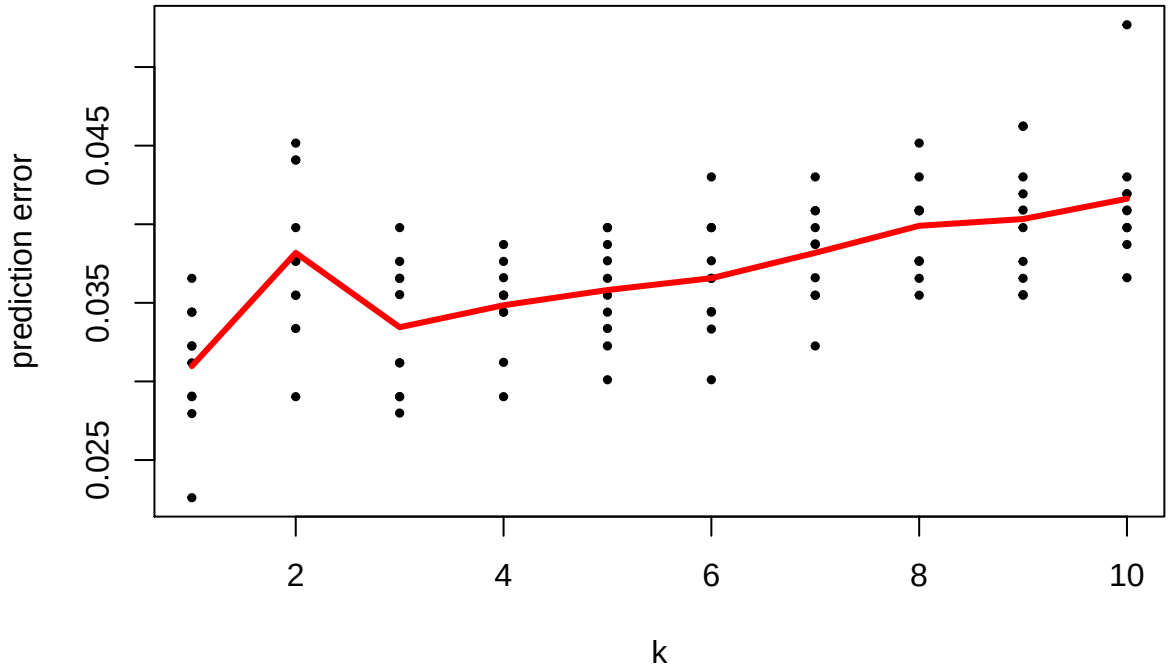
Use the Handwritten Digit Data from the `ElemStatLearn` package. Combine the train and test data defined in the package. For this question, you are asked to perform 10-fold cross-validation. Choose a grid of 10 different $k$ values reasonably and select the best tuning of $k$. Select the first observation, and find and plot the closest $k$ neighbors (in the rest of the dataset) based on your optimal $k$. Is this observation correctly classified? Comment on the overall performence of KNN on this dataset.

###Answer:

```
library(ElemStatLearn)
library(class)
#define data matrix and response vector
dat <- rbind(zip.train,zip.test)
dat.x <- dat[,2:257]
dat.y <- dat[,1]
#10 different k and 10 folds
#I found that the error increases monotonely when k>10
#Thus I changed this k sequence to be 1:10
newk <- c(1:10)
nfold <- 10
infold = sample(rep(1:nfold, length.out=nrow(dat)))
errorMatrix = matrix(0,10,10)
```

```
for (j in 1:nfold){
  for (k in 1:length(newk)){
        knn.fit=knn(dat.x[infold!=j,],dat.x[infold==j,],dat.y[infold!=j],newk[k])
        errorMatrix[k,j]=mean(knn.fit!=dat.y[infold==j])
  }
}
```

```
plot(rep(newk,nfold),as.vector(errorMatrix),pch=19,
     cex=0.5,xlab="k",ylab="prediction error")
points(newk,apply(errorMatrix,1,mean),col="red",
       pch=19,type="l",lwd=3)
```




The parameter k with best prediction is:

```
newk[which.min(apply(errorMatrix, 1, mean))]
```

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

Find the index of the nearest neighbor of the first obs with k=1:

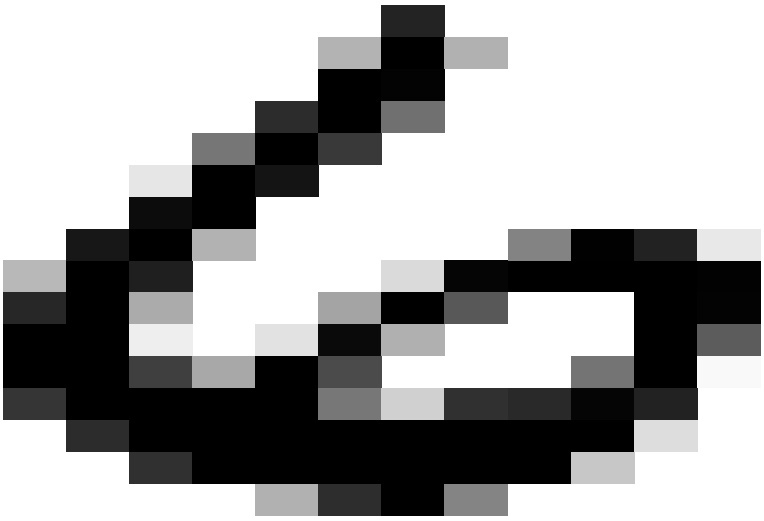
```
library(FNN)
get.knn(dat,k=1)$nn.index[1]
```

```
## [1] 1357
```

Print the nearest neighbor of the first obs and itself:

```
image(zip2image(dat,1), col=gray(256:0/256), zlim=c(0,1),
      xlab="1st obs", ylab="", axes = FALSE)
```

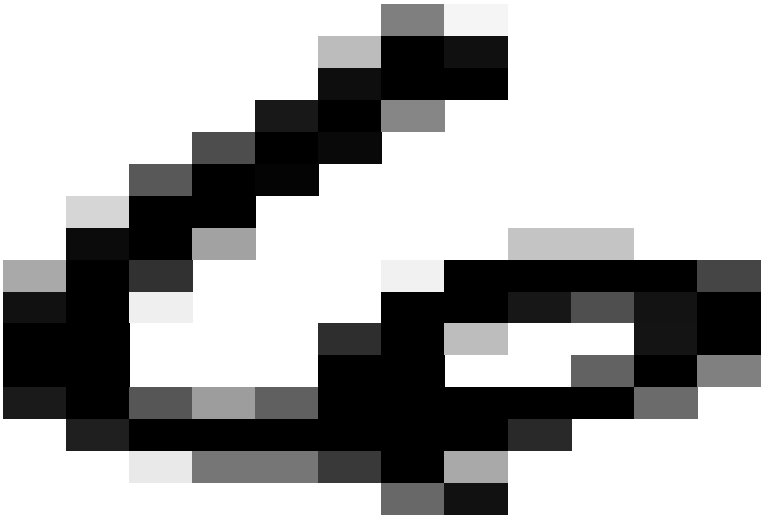
```
## [1] "digit  6  taken"
```



1st obs

```
image(zip2image(dat,1357), col=gray(256:0/256), zlim=c(0,1),
      xlab="nearest neighbor", ylab="", axes = FALSE)
```

```
## [1] "digit  6  taken"
```



nearest neighbor

kNN performs well based on this dataset and its prediction is fairly accurate.