```
library(kknn)

# Returns a vector of the Prob that a given observation xi is Spam or Not
knearest = function(data, k, newdata) {

  n1=dim(data)[1] #number of observations in training set
  n2=dim(newdata)[1] #number of observations in test set
  p=dim(data)[2] #number of variables observed in both sets (including whether it is spam or not)
  Prob=numeric(n2) #vector as long as test data, to be filled with probabilities for spam or not spam
  X=as.matrix(data[,-p]) # matrix of the training data excluding the spam/not spam
  Y=as.matrix(newdata[,-p]) # matrix is the test data excludign the spam/not spam
  X=X/matrix(sqrt(rowSums(X^2)), nrow=n1, ncol=p-1) #divide each Xi on row j by sqrt of the sum of the square of all Xij
from current row j
  Y=Y/matrix(sqrt(rowSums(Y^2)), nrow=n2, ncol=p-1) #same but for Y
  C=X%*%t(Y) # Calulate the cosine simularity of data compared to newdata
  D=1-C # Calculate the distance between data and newdata
  for (i in 1:n2 ){
    nearest_neighbors_indexes = order(D[,i])[1:k] # get indexes of the 5 nearest
    Prob[i] = sum(data[nearest_neighbors_indexes,p])/k # calculate the probability of it being spam or not by Ki/K
  }
  return(Prob)
}

# Returns a list with the TPR and FPR values of each probability p
ROC = function(Y, Yfit, p){
  m=length(p) # number of p values to test
  TPR=numeric(m) # a vector of TPR values to be filled with one value for each p value
  FPR=numeric(m) # same as TPR but for TPR
  for(i in 1:m){
    Ypred=ifelse(Yfit>p[i],1,0) # calculate the predictions
    t=table('true'=Ypred, 'pred'=Y) # calculate the confusion matrix
    print(t)
    TPR[i]=t[2,2]/sum(t[,2]) # calculate the TPR value
    FPR[i]=t[2,1]/sum(t[,1]) # calculate the FPR value
  }
  return (list(TPR=TPR,FPR=FPR))
}

# Main program
dataframe = read.csv("spambase.csv", dec=',')
n = dim(dataframe)[1]
set.seed(12345)
id=sample(1:n, floor(n/2))
train = dataframe[id,]
test = dataframe[-id,]

# Calculate the probability of an observation being spam using K nearest neighbors
Prob_k5 = knearest(train, 5, test) # k = 5
Prob_k1 = knearest(train, 1, test) # k = 1
Prob_k5_kknn = kknn(Spam ~ .,train=train, test=test, k=5) # k = 5 using kknn

# Confusion matrixes for the different k values/methods of calculation
cm_k5 = table(Prob_k5>0.5, test[,49])
cm_k1 = table(Prob_k1>0.5, test[,49])
cm_k5_kknn = table(Prob_k5_kknn$fitted.values>0.5, test[,49])

# Missclassification rate for the different k values/methods of calculation
mcr_k5 = 1-sum(diag(cm_k5)/sum(cm_k5))
mcr_k1 = 1-sum(diag(cm_k1)/sum(cm_k1))
mcr_k5_kknn = 1-sum(diag(cm_k5_kknn)/sum(cm_k5_kknn))

# ROC and sensitivty for k = 5
p_seq = seq(from=0.05, to=0.95, by=0.05) # probabilty values to test
list_result = ROC(test[,49], Prob_k5, p_seq) # list of TPR and FPR values for each probabilty p
list_result_kknn = ROC(test[,49], Prob_k5_kknn$fitted.values, p_seq) # list of TPR and FPR values for each probabilty p

# Roc Curves
plot(x=list_result$FPR, y=list_result$TPR, xlab="FPR", ylab="TPR", main="ROC Curves", xlim=c(0,1), ylim=c(0,1),
type="l", col="blue") # ROC curve for knearest
legend("topright", legend=c("(blue) knearest, k=5", "(green) kknn, k=5"))
lines(x=list_result_kknn$FPR, y=list_result_kknn$TPR, type="l", col="green") # ROC curve for kknn

# Specificity
specificity = 1-list_result$FPR
specificity_kknn = 1-list_result_kknn$FPR

plot(p_seq, specificity , xlab="p", ylab="Specifity (1-FPR)", main="Specificity", xlim=c(0,1), ylim=c(0,1), type="l",
col="blue")
legend("bottomright", legend=c("(blue) knearest, k=5", "(green) kknn, k=5"))
lines(p_seq, specificity_kknn, col="green")
```

```
# Sensitivity
sensitivity = list_result$TPR
sensitivity_kknn = list_result_kknn$TPR

plot(p_seq, sensitivity, xlab="p", ylab="Sensitivity (TPR)", main="Sensitivity", xlim=c(0,1), ylim=c(0,1), type="l",
col="blue")
legend("topright", legend=c("(blue) knearest, k=5", "(green) kknn, k=5"))
lines(p_seq, sensitivity_kknn, col="green")
```