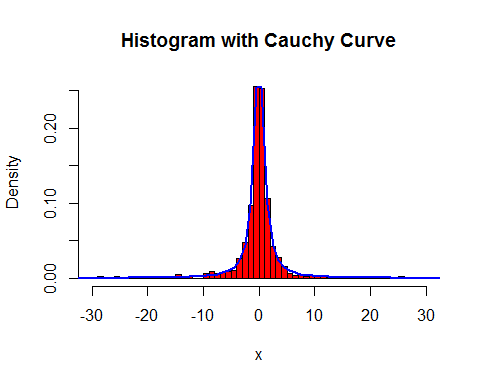
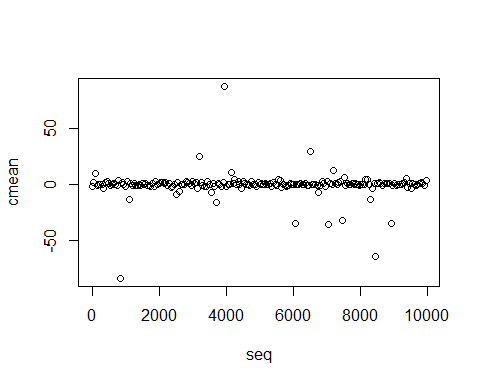
## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

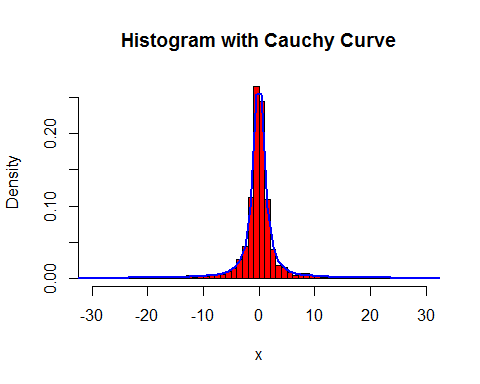
#########################################################################  
# Problem 1  
#########################################################################  
  
rnormalBX <- function(n=1, mean = 0, var = 1, seed = NULL){  
   
 if(!is.null(seed)) set.seed(seed) # set seed for the random number  
   
 # generate two uniform and then calculate r and theta and finally generate normal random number  
   
 u = runif(n, 0, 1)  
 v = runif(n, 0, 1)  
 r = sqrt(-2\*log(v))  
 theta = 2\*pi\*u  
 norm = cbind(r\*cos(theta), r\*sin(theta))  
 if(n != 1) {norm = (norm - colMeans(norm))/sqrt(diag(var(norm)))}  
   
 # transform normal random number with specified mean and variance   
   
 if (mean != 0 || var != 1){  
 norm = sqrt(var)\*norm + mean  
   
 }  
 return(unname(norm, force = TRUE))  
}  
  
# # calling rnormalBX function to generate normal random number using Box-Muller algorithm  
# x = rnormalBX(n = 1000, mean = 0, var = 1, seed = 123)  
# summary(x)  
  
  
rcauchydist <- function(n=1, seed = NULL){  
 x = rnormalBX(n = n, mean = 0, var = 1, seed = seed)  
 return(x[,1]/x[,2])  
}  
  
x = rcauchydist(1000, seed = 4578)  
#Part 1 Independence of two normal random variable generated using Box-Muller algorithm  
  
  
  
# Density Curve for the normal sample. Function densityCurve has two parameters, data = data set for the density curve, dist = distribution.  
  
  
densityCurve <- function(data , nbreaks = NULL){  
   
 if(is.null(nbreaks)){ nbreaks = pretty(data)}  
 title = "Histogram with Cauchy Curve"  
 his <- hist(data, breaks = nbreaks, freq = FALSE, col="red", xlab="x", main=title, xlim = c(-30,30))  
 xfit <- his$mids   
 yfit <- dcauchy(xfit, location = 0, scale = 1)  
 lines(xfit, yfit, col="blue", lwd=2)  
   
}  
  
  
# Calling densityCurve function to create density curve.  
densityCurve(x, nbreaks = 500)



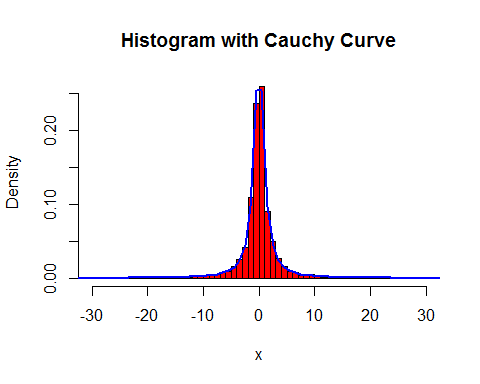
# problem 1 part 2  
  
cauchyMeanForEachSample <- function(samSize = 1:100, seed = NULL){  
 if(!is.null(seed)) set.seed(seed)  
 return(apply(as.array(samSize), 1, function(x) mean(rcauchydist(x))))  
}  
  
seq = seq.int(1,10^4, 50)  
cmean = cauchyMeanForEachSample(seq, seed = 245)  
plot(seq, cmean)



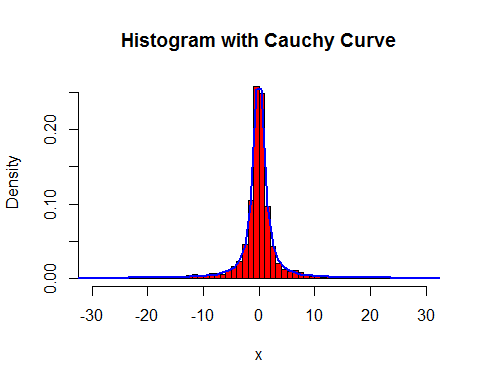
# problem 1 part 3  
  
cauchyMeanSim <- function(samSize = 100, distSize = 1000 , seed = NULL){  
 if(!is.null(seed)) set.seed(seed)  
 meandist = replicate(distSize, mean(rcauchydist(samSize)))  
   
 return(meandist)  
}  
  
cmean = cauchyMeanSim(40, distSize = 2000, seed = 365)  
densityCurve(cmean, nbreaks = 800)



cmean = cauchyMeanSim(100, distSize = 2000, seed = 365)  
densityCurve(cmean, nbreaks = 1300)



cmean = cauchyMeanSim(500, distSize = 2000, seed = 4125)  
densityCurve(cmean, nbreaks = 1500)



#########################################################################  
# Problem 2  
#########################################################################  
  
  
require('quadprog')

## Loading required package: quadprog

data <- read.table("C:/Users/ka746940/Desktop/UCF/STA 6106 - Statistical Computing/Assignments/Midterm/pb2.txt")  
#data <- read.table("D:/UCF/STA 6106 Statistical Computing/Assignments/Midterm/pb2.txt")  
data[,1][data[,1]==2] <- -1  
X = data[,2:5]  
mmean = colMeans(X)  
cvar = diag(var(X))  
Y = data[,1]  
X1 = scale(X)  
## Defining the Gaussian kernel  
rbf\_kernel <- function(x1,x2,ker\_par){  
 x1 = as.matrix(x1)  
 x2 = as.matrix(x2)  
 K<-exp(-(1/(ker\_par^2))\*t(x1-x2)%\*%(x1-x2))  
 return(K)  
}  
  
  
poli\_kernel <- function(x1, x2, c, d){  
 K<- (t(as.matrix(x1)) %\*% as.matrix(x2) + c)^d  
 return(K)  
}  
  
  
kcalculator <- function(X, kernel, ker\_par){  
 X=as.matrix(X)  
 N<-dim(X)[1]  
 K<-matrix(0,N,N)  
 if (toupper(kernel)== "GAUSSIAN"){  
 for(i in 1:N){  
 for(j in 1:N){  
 K[i,j]<-rbf\_kernel(X[i,],X[j,],ker\_par)  
 }  
 }  
 }  
 if (toupper(kernel)== "POLYNOMIAL"){  
 for(k in 1:N){  
 for(l in 1:N){  
 K[k,l]<-poli\_kernel(X[k,],X[l,],ker\_par[1], ker\_par[2])  
 }  
 }  
 }  
 return(K)  
}  
  
  
bcalculator <- function(Y, X, alpha, kernel, ker\_par){  
 N<-length(Y)  
 K = kcalculator(X, kernel, ker\_par)  
 w01=rowSums((alpha\*Y)\*K)  
 w0 = mean(Y-w01)  
   
}  
  
  
  
svmtrain <- function(X, Y, C=Inf, kernel = "Gaussian", ker\_par =1.5, esp=1e-2){  
 N<-length(Y)  
 X<-as.matrix(X)  
 Y<-as.vector(Y)  
   
 K = kcalculator(X, kernel, ker\_par)  
 Dm = (Y %\*% t(Y))\*K  
 Dm<-Dm+diag(N)\*1e-8 # adding a very small number to the diag, some trick  
 dv<-t(rep(1,N))  
 meq<-1  
 Am<-cbind(matrix(Y,N),diag(N))  
 bv<-rep(0,1+N) # the 1 is for the sum(alpha)==0, others for each alpha\_i >= 0  
 if(C!=Inf){  
 # an upper bound is given  
 Am<-cbind(Am,-1\*diag(N))  
 bv<-c(cbind(matrix(bv,1),matrix(rep(-C,N),1)))  
 }  
 alpha\_org<-solve.QP(Dm,dv,Am,bvec=bv, meq=meq)$solution  
 indx<-which(alpha\_org>esp,arr.ind=TRUE)  
 alpha<-alpha\_org[indx]  
 nSV<-length(indx)  
 if(length(indx)==0){  
 throw("QP is not able to give a solution for these data points")  
 }  
 Xv<-X[indx,]  
 Yv<-as.vector(Y[indx])  
 w<-unname(t(Xv)%\*%(alpha\*Yv), force = TRUE)  
 # choose one of the support vector to compute b. for safety reason,  
 # select the one with max alpha  
   
 b = bcalculator(Yv, Xv, alpha, kernel, ker\_par)  
   
 return(list(alpha=alpha, wstar=w, b=b, nSV=nSV, Xv=Xv, Yv=Yv, kernel = kernel ,ker\_par=ker\_par))  
}  
  
### Predict the class of an object X  
  
  
  
svmpredict <- function(x,model){  
 x = as.matrix(x)  
 kernel = model$kernel  
 ker\_per = model$ker\_par  
 alpha<-model$alpha  
 b<-model$b  
 Yv<-model$Yv  
 Xv<-model$Xv  
 ker\_par<-model$ker\_par  
 # wstar<-model$wstar  
 result = as.vector(rep(0,dim(x)[1]))  
 for (k in 1:dim(x)[1]){  
 sum = 0  
 if (toupper(kernel)== "GAUSSIAN"){  
 for (i in 1 : length(alpha)){  
 s1 = alpha[i] \* Yv[i] \* rbf\_kernel(Xv[i,],x[k,],ker\_per)  
 sum = sum + s1  
 }  
 result[k]<-sign(sum + b)  
 }  
   
 if (toupper(kernel)== "POLYNOMIAL"){  
 for (i in 1 : length(alpha)){  
 s1 = alpha[i] \* Yv[i] \* poli\_kernel(Xv[i,],x[k,],ker\_per[1], ker\_per[2])  
 sum = sum + s1  
 }  
 result[k]<-sign(sum + b)  
 }  
 }  
   
 return(result)  
}  
  
  
model1 = svmtrain(X1, Y, kernel = "Polynomial", ker\_par = c(23,2))  
model1

## $alpha  
## [1] 0.9442694 12.7598160 2.9771319 0.4951243 6.5785068 4.6269065  
## [7] 17.0508919 13.1673609 1.6616235 9.1511914 1.9643374 14.0507365  
## [13] 29.1091677 5.4996860 0.4865120  
##   
## $wstar  
## [,1]  
## [1,] 0.13176131  
## [2,] -0.05720521  
## [3,] 0.23166994  
## [4,] -0.19237407  
##   
## $b  
## [1] -2.113031  
##   
## $nSV  
## [1] 15  
##   
## $Xv  
## V2 V3 V4 V5  
## [1,] 0.26995486 0.50746227 0.279433139 -1.78899530  
## [2,] -0.34994149 -0.70264007 -1.615472836 -1.36238873  
## [3,] -0.34994149 -2.39678334 0.008732286 -0.08256901  
## [4,] -0.03999331 -1.91274240 1.091535700 -1.14908544  
## [5,] -1.27978602 -0.21859913 -0.397318995 -1.14908544  
## [6,] 1.19979939 1.47554414 1.091535700 1.41055399  
## [7,] 0.57990304 0.26544180 -0.803370275 -1.36238873  
## [8,] -0.96983784 0.02342134 0.414783566 0.13073427  
## [9,] 0.88985122 1.23352367 -0.126618141 -0.29587230  
## [10,] -0.34994149 0.26544180 -1.886173690 -1.78899530  
## [11,] -0.03999331 -1.67072194 -1.209421556 0.55734084  
## [12,] 0.57990304 -0.46061960 -0.803370275 -1.78899530  
## [13,] -0.03999331 0.50746227 0.279433139 0.13073427  
## [14,] -2.20963055 -1.91274240 -0.397318995 -0.93578216  
## [15,] -2.51957873 -2.39678334 -2.156874544 -2.00229859  
##   
## $Yv  
## [1] 1 1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1  
##   
## $kernel  
## [1] "Polynomial"  
##   
## $ker\_par  
## [1] 23 2

model =svmtrain(X, Y, kernel = "Gaussian", ker\_par = 1.5)  
model

## $alpha  
## [1] 1.0501540 0.9502877 0.8921877 1.0501938 1.0500985 1.0497410 0.7327245  
## [8] 1.0501697 1.0501697 0.8370291 1.0486323 1.0607960 0.1564406 0.7565102  
## [15] 0.7049907 0.7364052 1.0503008 0.9806550 1.0430088 0.9920919 1.0560719  
## [22] 1.0624388 1.0501487 1.0391176 0.9111579 0.9539508 0.7780606 0.9744232  
## [29] 0.9707955 1.0501780 0.7806911 0.9498212 0.9494753 0.9326510 0.9498211  
## [36] 0.9498309 0.9498184 0.9496841 0.9327482 0.9496976 0.9329458 0.9497462  
## [43] 0.9498303 0.9498547 0.9492968 0.8919099 0.9622947 0.9530680 0.9497318  
## [50] 0.7567436 0.9498302 0.8143531 0.9478474 0.9530772 0.9346812 0.9498302  
## [57] 0.8117641 0.9499648 0.9498131 0.9498302 0.9498302 0.9498302  
##   
## $wstar  
## [,1]  
## [1,] 101.601256  
## [2,] 56.720330  
## [3,] 286.630550  
## [4,] 8.467797  
##   
## $b  
## [1] -0.05016975  
##   
## $nSV  
## [1] 62  
##   
## $Xv  
## V2 V3 V4 V5  
## [1,] 15 17 24 14  
## [2,] 17 15 32 26  
## [3,] 15 14 29 23  
## [4,] 13 12 10 16  
## [5,] 20 17 26 28  
## [6,] 15 21 26 21  
## [7,] 15 13 26 22  
## [8,] 13 5 22 22  
## [9,] 14 7 30 17  
## [10,] 17 15 30 27  
## [11,] 17 17 26 20  
## [12,] 17 20 28 24  
## [13,] 15 15 29 24  
## [14,] 18 19 32 28  
## [15,] 18 18 31 27  
## [16,] 15 14 26 21  
## [17,] 10 14 19 17  
## [18,] 18 21 30 29  
## [19,] 18 21 34 26  
## [20,] 13 17 30 24  
## [21,] 16 16 16 16  
## [22,] 11 15 25 23  
## [23,] 16 13 26 16  
## [24,] 16 13 23 21  
## [25,] 18 18 34 24  
## [26,] 16 15 28 27  
## [27,] 15 16 29 24  
## [28,] 18 19 32 23  
## [29,] 18 16 33 23  
## [30,] 17 20 21 21  
## [31,] 19 19 30 28  
## [32,] 13 14 12 21  
## [33,] 14 12 14 26  
## [34,] 12 19 21 21  
## [35,] 11 20 16 16  
## [36,] 12 9 14 18  
## [37,] 10 13 18 24  
## [38,] 10 8 13 23  
## [39,] 12 20 19 23  
## [40,] 11 10 11 27  
## [41,] 12 18 25 25  
## [42,] 14 18 13 26  
## [43,] 14 10 25 28  
## [44,] 13 16 8 14  
## [45,] 14 8 13 25  
## [46,] 13 16 23 28  
## [47,] 16 21 26 26  
## [48,] 14 17 14 14  
## [49,] 16 16 15 23  
## [50,] 13 16 23 24  
## [51,] 2 6 16 21  
## [52,] 14 16 22 26  
## [53,] 17 17 22 28  
## [54,] 16 13 16 14  
## [55,] 15 14 20 26  
## [56,] 12 10 12 9  
## [57,] 14 17 24 23  
## [58,] 13 15 18 20  
## [59,] 11 16 18 28  
## [60,] 7 7 19 18  
## [61,] 12 15 7 28  
## [62,] 6 5 6 13  
##   
## $Yv  
## [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [24] 1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
## [47] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
##   
## $kernel  
## [1] "Gaussian"  
##   
## $ker\_par  
## [1] 1.5

p1 = svmpredict(X1, model1)  
cat("Training Erro", (length(Y)-sum(Y==p1))/length(Y)\*100, "%")

## Training Erro 0 %

p2 = svmpredict(X, model)  
cat("Training Erro", (length(Y)-sum(Y==p2))/length(Y)\*100, "%")

## Training Erro 0 %

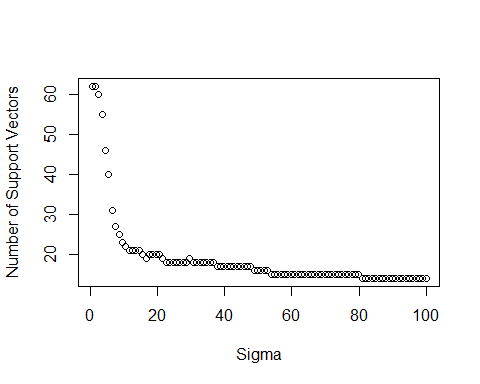
a = matrix(c(18,17,33,26), 1,4,byrow = TRUE)  
a1 = (a-mmean)/cvar  
p1 = svmpredict(a1, model1)  
cat("Prediction", p1)

## Prediction 1

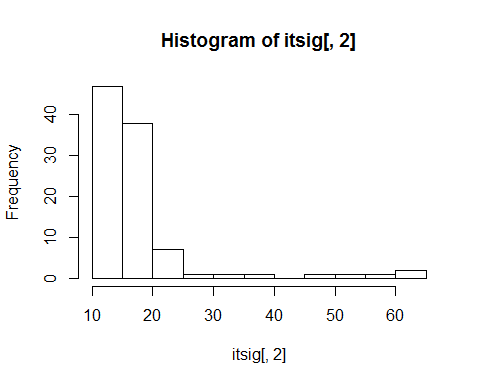
p2 = svmpredict(a, model)  
cat("Prediction", p2)

## Prediction 1

# part b problem 2  
  
sigm = seq(.5, 100, length = 100)  
itsig = matrix(0, length(sigm), 3)  
  
for (i in 1:length(sigm)){  
 fit = svmtrain(X, Y, kernel = "Gaussian", ker\_par = sigm[i])  
 pred = svmpredict(X, fit)  
 error = (length(Y)-sum(Y==pred))/length(Y)\*100  
 nSV = fit$nSV  
 itsig[i,] = c(error, nSV, sigm[i])  
}  
  
  
plot(itsig[,3], itsig[,2], xlab = "Sigma", ylab = "Number of Support Vectors")



hist(itsig[,2])



#########################################################################  
# Problem 3  
#########################################################################  
  
  
require('quadprog')  
X = as.list(numeric())  
for (j in 1:15){  
 x2 = rnorm(5, 2, 1)  
 for (i in 1:4){  
 x1 = rnorm(5, 2, 1)  
 x2= cbind(x2, x1)  
 }  
 x2=unname(as.matrix(x2), force = TRUE)  
 X[j]=list(x2)  
}  
for (j in 16:25){  
 x2 = rnorm(5, 0, 1)  
 for (i in 1:4){  
 x1 = rnorm(5, 0, 1)  
 x2= cbind(x2, x1)  
 }  
 x2=unname(as.matrix(x2), force = TRUE)  
 X[j]=list(x2)  
}  
  
  
Y = c(rep(1,15), rep(-1,10))  
  
## Defining the Gaussian kernel  
rbf\_kernel <- function(x1,x2,gamma){  
 x1 = as.matrix(x1)  
 x2 = as.matrix(x2)  
 return(exp(-(1/gamma^2)\*sum(diag(t(x1 - x2) %\*% (x1 - x2)))))  
}  
  
  
  
  
  
kcalculator <- function(X, ker\_par){  
 X=as.matrix(X)  
 N<-dim(X)[1]  
 K<-matrix(0,N,N)  
 for(i in 1:N){  
 for(j in 1:N){  
 K[i,j]<-rbf\_kernel(X[i,][[1]],X[j,][[1]],ker\_par)  
 }  
 }  
 return(K)  
}  
  
  
  
bcalculator <- function(Y, X, alpha, ker\_par){  
 N<-length(Y)  
 K = kcalculator(X, ker\_par)  
 w01=rowSums((alpha\*Y)\*K)  
 w0 = mean(Y-w01)  
   
}  
  
  
  
svmtrain <- function(X, Y, C=Inf, ker\_par =1.5, esp=1e-2){  
 N<-length(Y)  
 X<-as.matrix(X)  
 Y<-as.vector(Y)  
   
 K = kcalculator(X, ker\_par)  
 Dm = (Y %\*% t(Y))\*K  
 Dm<-Dm+diag(N)\*1e-8 # adding a very small number to the diag, some trick  
 dv<-t(rep(1,N))  
 meq<-1  
 Am<-cbind(matrix(Y,N),diag(N))  
 bv<-rep(0,1+N) # the 1 is for the sum(alpha)==0, others for each alpha\_i >= 0  
 if(C!=Inf){  
 # an upper bound is given  
 Am<-cbind(Am,-1\*diag(N))  
 bv<-c(cbind(matrix(bv,1),matrix(rep(-C,N),1)))  
 }  
 alpha\_org <- solve.QP(Dm,dv,Am,bvec=bv, meq=meq)$solution  
 indx<-which(alpha\_org>esp,arr.ind=TRUE)  
 alpha<-alpha\_org[indx]  
 nSV<-length(indx)  
 if(length(indx)==0){  
 throw("QP is not able to give a solution for these data points")  
 }  
 Xv<-X[indx,]  
 Yv<-as.vector(Y[indx])  
 # choose one of the support vector to compute b. for safety reason,  
 # select the one with max alpha  
   
 b = bcalculator(Yv, Xv, alpha, ker\_par)  
   
 return(list(alpha=alpha, b=b, nSV=nSV, Xv=Xv, Yv=Yv, ker\_par=ker\_par))  
}  
  
### Predict the class of an object X  
  
model1 = svmtrain(X, Y, ker\_par = 34)  
model1

## $alpha  
## [1] 17.435224 4.636095 1.387078 7.398206 14.387654 1.672537  
##   
## $b  
## [1] 0.04111517  
##   
## $nSV  
## [1] 6  
##   
## $Xv  
## $Xv[[1]]  
## [,1] [,2] [,3] [,4] [,5]  
## [1,] 1.791005 0.9373987 1.5704581 1.7116151 0.8715314  
## [2,] 1.935961 0.7035555 1.8409616 1.6568003 3.7042145  
## [3,] 1.518782 1.5514231 1.4039887 1.7383544 -0.1136536  
## [4,] 2.590826 2.1883605 0.3589047 0.9670999 -0.1341516  
## [5,] 2.213851 1.2717987 1.1242464 2.4561336 3.0036224  
##   
## $Xv[[2]]  
## [,1] [,2] [,3] [,4] [,5]  
## [1,] 2.193985 1.8601529 0.7735266 2.5817856 1.427708  
## [2,] 1.085144 0.6706914 1.4580711 -0.4305902 1.296278  
## [3,] 2.152642 2.8609584 2.3579620 -0.5820291 2.665084  
## [4,] 1.977562 1.5517392 3.6612383 1.1882341 1.439582  
## [5,] 1.748261 2.7771260 3.3405637 1.0744940 1.990053  
##   
## $Xv[[3]]  
## [,1] [,2] [,3] [,4] [,5]  
## [1,] 1.61138123 1.489698 1.6553262 1.1443286 1.783337  
## [2,] 2.97046801 3.152641 2.7846135 1.3769139 1.651357  
## [3,] 0.01290151 1.496316 3.0440754 2.4604586 1.822243  
## [4,] 1.62229301 1.228470 0.8353059 2.2957132 3.401451  
## [5,] 2.29624371 1.256652 4.6783608 0.7917519 0.949735  
##   
## $Xv[[4]]  
## [,1] [,2] [,3] [,4] [,5]  
## [1,] 0.4422517 -0.4147885 1.93637905 0.5486326 1.36412203  
## [2,] 0.3031537 -0.2312506 -0.06097989 -1.2274083 1.19712888  
## [3,] -0.5985895 0.1965434 0.79569075 1.6857144 0.36783316  
## [4,] -0.8235111 -0.2344804 -1.18259827 1.6403477 -0.02473491  
## [5,] 0.8244089 0.5836166 -0.66056424 0.6839662 1.27212903  
##   
## $Xv[[5]]  
## [,1] [,2] [,3] [,4] [,5]  
## [1,] 0.9681756 0.7562062 0.3102077 -0.3132653 1.0043097  
## [2,] 1.9554157 -1.0739419 0.6826529 0.3173759 0.2363517  
## [3,] 0.6348683 0.3340240 0.7388280 0.7089062 -0.3998155  
## [4,] -0.1040241 0.0567613 1.3390056 1.3473598 -1.1768608  
## [5,] -0.3897819 -0.7907710 0.5882989 1.5989761 1.2699914  
##   
## $Xv[[6]]  
## [,1] [,2] [,3] [,4] [,5]  
## [1,] -1.0422518 0.4358865 -1.03951011 -1.35935770 -0.13912231  
## [2,] -0.7468983 -1.0385101 0.77130590 1.84850157 1.85099530  
## [3,] 0.6263233 -0.8088807 -0.07453035 -0.10244283 -0.27370867  
## [4,] -1.6065223 1.9305588 -0.36347512 -0.06387201 -0.34498374  
## [5,] -0.1411257 0.7083914 2.84838971 0.64791087 -0.08064947  
##   
##   
## $Yv  
## [1] 1 1 1 -1 -1 -1  
##   
## $ker\_par  
## [1] 34

svmpredict <- function(x,model){  
 x = as.matrix(x)  
 kernel = model$kernel  
 ker\_per = model$ker\_par  
 alpha<-model$alpha  
 b<-model$b  
 Yv<-model$Yv  
 Xv<-as.matrix(model$Xv)  
 ker\_par<-model$ker\_par  
 # wstar<-model$wstar  
 result = as.vector(rep(0,dim(x)[1]))  
 for (k in 1:dim(x)[1]){  
 sum = 0  
 for (i in 1 : length(alpha)){  
 sum = sum + alpha[i] \* Yv[i] \* rbf\_kernel(Xv[i,][[1]],x[k,][[1]],ker\_per)  
 }  
 result[k]<-sign(sum + b)  
   
 }  
 margin = sum(alpha)  
 return(list(result = result, margin = sqrt(1/margin), primal = margin))  
}  
  
  
  
svmpredict(model1$Xv, model1)

## $result  
## [1] 1 1 1 -1 -1 -1  
##   
## $margin  
## [1] 0.1459943  
##   
## $primal  
## [1] 46.91679

svmpredict(X, model1)

## $result  
## [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1  
## [24] -1 -1  
##   
## $margin  
## [1] 0.1459943  
##   
## $primal  
## [1] 46.91679