**Jinghui Yuan 4177204**

**Problem 1**

We want to select the best values of the tuning parameters C and γ for the LS-SVM problem in R or Python or both using the dataset “pb2.txt”. Choose randomly 40 observations for training. You can use your favorite search engine to find details. Make sure to report the links or details.

1. Use ‘K-fold’ cross validation to find the best values of the tuning parameters C and γ for the LS-SVM problem – Choose an appropriate value of K.

In this problem, we will first develop a LS-SVM function to calculate the accuracy of test dataset based on the training dataset. The parameters of this function would be train.X, train.Y, test.X, test.Y, maximum iteration, gamma, and C.

1. Import the data;

> data <- read.table("C://Onedrive/OneDrive - Knights - University of Central Florida/UCF/Courses/Statistical Computing/Project1/pb2.txt")

1. Convert the data to matrix format and then change Y from (1, 2) to (1, -1);

> #Convert the data to matrix form

> Data <- as.matrix(data, ncol=5)

> N <- length(Y)

> for (i in 1:N){

+ if (Data[i,1] > 1){

+ Data[i,1] <--1

+ }

+ }

1. Define the Gaussian kernel based on the formula of ;

> rbf\_kernel <- function(x1,x2,gamma){

+ K<-exp(-(1/gamma^2)\*t(x1-x2)%\*%(x1-x2))

+ return(K)

+ }

1. Conjugate Gradient Method to solving the problem Ax=B;

Define the function of conjugate gradient method with the input of matrix A and B, together with a predefined large value, which represents the maximum iterations.

> ## Conjugate Gradient Method to solving the problem Ax=B

> conjugate\_gradient\_method<-function (A,B,N){

+ M<-length(B)

+ i<-1

+ x<-matrix(0,N,M)

+ r<-matrix(0,N,M)

+ p<-matrix(0,N,M)

+ beta<-numeric(N)

+ lamda<-numeric(N)

+ r[1,]<-B

+ if (t(r[i,])%\*%r[i,]>1e-5){

+ i<-i+1

+ if (i==2){

+ p[i,]<-r[1,]

+ }

+ else {

+ beta[i]<-t(r[i-1,])%\*%r[i-1,]/t(r[i-2,])%\*%r[i-2,]

+ p[i,]<-r[i-1,]+beta[i]\*p[i-1,]

+ }

+ lamda[i]<-t(r[i-1,])%\*%r[i-1,]/t(p[i,])%\*%A%\*%p[i,]

+ x[i,]<-x[i-1,]+lamda[i]\*p[i,]

+ r[i,]<-r[i-1,]-(lamda[i]\*A)%\*%p[i,]

+ }

+ return(x[i,])

+ }

1. Training the LSSVM and predict the y for test dataset, then calculate the accuracy;

> lssvmmodel<-function(X, Y, Xt, Yt, step, gamma, C){

Based on the given training dataset X and Y, calculate the H matrix and d2 matrix.

+ N <- length(Y)

+ Dm<-matrix(0,N,N)

+ for(i in 1:N){

+ for(j in 1:N){

+ Dm[i,j]<-Y[i]\*Y[j]\*rbf\_kernel(X[i,1:4],X[j,1:4],gamma)

+ }

+ }

+ H<-Dm+diag(N)\*(1/C)+diag(N)\*1e-12 # adding a very small number to the diag, some trick

+ d2<-as.vector(rep(1,N))

Call the Conjugate Gradient Method function to solve the equation of H\*X=Y, H\*X=d2.

+ # Call the Conjugate Gradient Method function to get the alpha and b

+ nta<-conjugate\_gradient\_method(H,Y,step)

+ vta<-conjugate\_gradient\_method(H,d2,step)

+ s<-t(Y)%\*%nta

+ b<-(t(nta)%\*%d2)/s

+ alpha<-vta-nta\*b

+ alpha<-as.vector(alpha)

Predict the Y value for the test dataset based on the trained model.

+ # Predict the y for test dataset X\_t

+ M <- nrow(Xt)

+ y\_pred<-numeric(M)

+ for(k in 1:M){

+ ayK<-numeric(N)

+ for (l in 1:N){

+ ayK[l]<-alpha[l]\*Y[l]\*rbf\_kernel(Xt[k,1:4],X[l,1:4],gamma)

+ }

+ y\_pred[k] <- sign(sum(ayK)+b)

+ }

Calculate the predicted accuracy.

+ ### Evaluate the performance

+ correct<-0

+ for (m in 1:M){

+ correct<-correct+(1-0.5\*abs(Yt[m]-y\_pred[m]))

+ }

+ accuracy<-correct/M

+ return(accuracy)

+ }

1. Resample 60 rows from the raw dataset

> #Resample 60 rows from the raw dataset

> Data<-Data[sample(1:62, 60, replace=FALSE), ]

Cut the dataset to 5 folds.

> #Choose 5-fold datasets split (5\*12)- take the row number as the index of 5 fold datasets

> folds <- cut(seq(1,nrow(Data)),breaks=5,labels=FALSE)

1. Calculate the accuracy for different gamma and C value based on 5-fold CV

> #Calculate the accuracy for different gamma and C value based 5-fold CV

Define a function with the input of Data, C.range, and gamma.range to calculate the predicted accuracy.

> optimal<-function(Data, C.range, gamma.range){

+ N<-length(C.range)

+ M<-length(gamma.range)

+ tot.accuracy<-matrix(0,N,M)

+ pb = txtProgressBar(min = 0, max = N, initial = 0)

+ for (i in 1:N){

+ C<-C.range[i]

+ setTxtProgressBar(pb,i)

+ for (j in 1:M){

+ gamma<-gamma.range[j]

+ accuracy<-numeric(5)

For different C and gamma value, conduct 5-fold cross validation to calculate the average predicted accuracy.

+ for (k in 1:5){

+ #Segement the data by fold using the which() function

+ testIndexes <- which(folds==i,arr.ind=TRUE)

+ testData <- Data[testIndexes, ]

+ trainData <- Data[-testIndexes, ]

Subset the X and Y from the training and test dataset

+ train.X<-trainData[,2:5]

+ train.Y<-trainData[,1]

+ test.X<-testData[,2:5]

+ test.Y<-testData[,1]

Call the LS-SVM function to calculate the predicted accuracy for each fold.

+ accuracy[k]<-lssvmmodel(train.X, train.Y, test.X, test.Y, 500000, gamma, C)

+ }

+ tot.accuracy[i,j]<-sum(accuracy)/5

+ }

+ }

+ return(tot.accuracy)

+ }

1. Define the range of C value and gamma value

Define the range of C value and gamma value, we defined the range for C value is (1, 10), and the range for gamma is (0.1, 4)

> C.range<-seq(1,10, by=1)

> gamma.range<-seq(0.1,4, by=0.1)

1. Call the above function to generate the accuracy according to different C and gamma

> test<-optimal(Data, C.range, gamma.range)

> test

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]

[1,] 0.5500000 0.5666667 0.5000000 0.5666667 0.5833333 0.5666667 0.6166667 0.60 0.6166667

[2,] 0.4500000 0.5416667 0.5500000 0.6166667 0.5833333 0.5833333 0.6166667 0.60 0.6166667

[3,] 0.4500000 0.4666667 0.5666667 0.5833333 0.5833333 0.6166667 0.5833333 0.60 0.5833333

[4,] 0.5166667 0.5166667 0.6166667 0.5333333 0.5833333 0.5500000 0.6166667 0.60 0.6000000

[5,] 0.5500000 0.5666667 0.5166667 0.5666667 0.5833333 0.5833333 0.6000000 0.55 0.6166667

[6,] 0.5500000 0.5666667 0.5000000 0.5666667 0.5666667 0.5833333 0.5833333 0.60 0.6166667

[7,] 0.5500000 0.5666667 0.5000000 0.5666667 0.6166667 0.5500000 0.6000000 0.60 0.6000000

[8,] 0.4500000 0.5416667 0.6166667 0.5333333 0.5833333 0.5833333 0.5833333 0.60 0.5833333

[9,] 0.5500000 0.5666667 0.5000000 0.5833333 0.5333333 0.5666667 0.5833333 0.60 0.6166667

[10,] 0.4833333 0.5583333 0.5000000 0.5333333 0.5666667 0.5833333 0.6000000 0.60 0.6000000

[,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18]

[1,] 0.6333333 0.6333333 0.6000000 0.6166667 0.6166667 0.6166667 0.6333333 0.65 0.65

[2,] 0.6000000 0.6333333 0.6166667 0.6166667 0.6166667 0.6333333 0.6500000 0.65 0.65

[3,] 0.6333333 0.5833333 0.6333333 0.6166667 0.6166667 0.6333333 0.6500000 0.65 0.70

[4,] 0.6333333 0.6000000 0.6166667 0.6166667 0.6166667 0.6333333 0.6500000 0.65 0.70

[5,] 0.6166667 0.6166667 0.6333333 0.6166667 0.6166667 0.6333333 0.6500000 0.65 0.70

[6,] 0.6166667 0.6000000 0.6333333 0.6166667 0.6166667 0.6333333 0.6500000 0.65 0.70

[7,] 0.6166667 0.6166667 0.6333333 0.6166667 0.6166667 0.6333333 0.6500000 0.65 0.70

[8,] 0.6000000 0.5833333 0.6333333 0.6166667 0.6166667 0.6333333 0.6500000 0.65 0.70

[9,] 0.5833333 0.6000000 0.6333333 0.6166667 0.6333333 0.6333333 0.6500000 0.65 0.70

[10,] 0.6000000 0.6333333 0.6166667 0.6166667 0.6333333 0.6333333 0.6500000 0.65 0.70

[,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]

[1,] 0.6666667 0.7000000 0.7000000 0.7166667 0.7166667 0.7166667 0.7166667 0.7333333

[2,] 0.7000000 0.7000000 0.7166667 0.7166667 0.7166667 0.7166667 0.7166667 0.7333333

[3,] 0.7000000 0.7000000 0.7166667 0.7166667 0.7166667 0.7166667 0.7333333 0.7333333

[4,] 0.7000000 0.7000000 0.7166667 0.7166667 0.7166667 0.7166667 0.7333333 0.7333333

[5,] 0.7000000 0.7000000 0.7166667 0.7166667 0.7166667 0.7166667 0.7333333 0.7333333

[6,] 0.7000000 0.7166667 0.7166667 0.7166667 0.7166667 0.7333333 0.7500000 0.7333333

[7,] 0.7000000 0.7166667 0.7166667 0.7166667 0.7166667 0.7333333 0.7500000 0.7500000

[8,] 0.7000000 0.7166667 0.7166667 0.7166667 0.7166667 0.7333333 0.7500000 0.7500000

[9,] 0.7000000 0.7166667 0.7166667 0.7166667 0.7166667 0.7333333 0.7500000 0.7500000

[10,] 0.7000000 0.7166667 0.7166667 0.7166667 0.7333333 0.7333333 0.7500000 0.7500000

[,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34]

[1,] 0.7500000 0.7333333 0.7166667 0.7166667 0.7333333 0.7500000 0.7833333 0.7833333

[2,] 0.7333333 0.7166667 0.7166667 0.7166667 0.7500000 0.7833333 0.7833333 0.7833333

[3,] 0.7333333 0.7166667 0.7166667 0.7333333 0.7833333 0.7833333 0.7833333 0.7833333

[4,] 0.7166667 0.7166667 0.7166667 0.7333333 0.7833333 0.7833333 0.7833333 0.7833333

[5,] 0.7166667 0.7166667 0.7166667 0.7333333 0.7833333 0.7833333 0.7833333 0.7833333

[6,] 0.7166667 0.7166667 0.7166667 0.7333333 0.7833333 0.7833333 0.7833333 0.7833333

[7,] 0.7166667 0.7166667 0.7166667 0.7500000 0.7833333 0.7833333 0.7833333 0.7833333

[8,] 0.7166667 0.7166667 0.7166667 0.7500000 0.7833333 0.7833333 0.7833333 0.7833333

[9,] 0.7166667 0.7166667 0.7166667 0.7666667 0.7833333 0.7833333 0.7833333 0.7833333

[10,] 0.7166667 0.7166667 0.7166667 0.7666667 0.7833333 0.7833333 0.7833333 0.7833333

[,35] [,36] [,37] [,38] [,39] [,40]

[1,] 0.7833333 0.7833333 0.8000000 0.7833333 0.7833333 0.7833333

[2,] 0.7833333 0.8000000 0.7833333 0.7833333 0.7833333 0.7833333

[3,] 0.7833333 0.8000000 0.7833333 0.7833333 0.7833333 0.7833333

[4,] 0.8000000 0.7833333 0.7833333 0.7833333 0.7833333 0.7833333

[5,] 0.8000000 0.7833333 0.7833333 0.7833333 0.7833333 0.7833333

[6,] 0.8000000 0.7833333 0.7833333 0.7833333 0.7833333 0.7833333

[7,] 0.8000000 0.7833333 0.7833333 0.7833333 0.7833333 0.7833333

[8,] 0.8000000 0.7833333 0.7833333 0.7833333 0.7833333 0.7833333

[9,] 0.8000000 0.7833333 0.7833333 0.7833333 0.7833333 0.7833333

[10,] 0.8000000 0.7833333 0.7833333 0.7833333 0.7833333 0.7833333

1. Index the best accuracy value in the result matrix, and then pick up the corresponding C and gamma value

Index the maximum value in the accuracy matrix

> valueindex<-which(test == max(test), arr.ind = TRUE)

> valueindex

row col

[1,] 4 35

[2,] 5 35

[3,] 6 35

[4,] 7 35

[5,] 8 35

[6,] 9 35

[7,] 10 35

[8,] 2 36

[9,] 3 36

[10,] 1 37

>

Pick up the corresponding C value

> C.list<-C.range[valueindex[,1]]

> C.list

[1] 4 5 6 7 8 9 10 2 3 1

>

Pick up the corresponding gamma value

> gamma.list<-gamma.range[valueindex[,2]]

> gamma.list

[1] 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.6 3.6 3.7

The above results indicate that the value of C does not make much sense to the predicted accuracy, however the gamma value around 3.5-3.7 could have highest prediction performance.

1. Use ‘Leave-one-out cross validation’ (LOOCV) to find the best values of the tuning parameters C and γ for the LS-SVM problem

Since the LOOCV is a special case of k-fold CV with k equals to 60, thus we did some modification based on the above function.

> ## Training the LSSVM and predict the y for test dataset, then determine the predict is true or not

> lssvmmodel\_LOOCV<-function(X, Y, Xt, Yt, step, gamma, C){

+ N <- length(Y)

+ Dm<-matrix(0,N,N)

+ for(i in 1:N){

+ for(j in 1:N){

+ Dm[i,j]<-Y[i]\*Y[j]\*rbf\_kernel(X[i,1:4],X[j,1:4],gamma)

+ }

+ }

+ H<-Dm+diag(N)\*(1/C)+diag(N)\*1e-12 # adding a very small number to the diag, some trick

+ d2<-as.vector(rep(1,N))

+ # Call the Conjugate Gradient Method function to get the alpha and b

+ nta<-conjugate\_gradient\_method(H,Y,step)

+ vta<-conjugate\_gradient\_method(H,d2,step)

+ s<-t(Y)%\*%nta

+ b<-(t(nta)%\*%d2)/s

+ alpha<-vta-nta\*b

+ alpha<-as.vector(alpha)

+

Predict the y for test dataset X\_t

+ ayK<-numeric(N)

+ for (l in 1:N){

+ ayK[l]<-alpha[l]\*Y[l]\*rbf\_kernel(Xt,X[l,1:4],gamma)

+ }

+ y\_pred <- sign(sum(ayK)+b)

+

Determine whether the predict is true or faulse

+ correct<-1-0.5\*abs(Yt-y\_pred)

+ return(correct)

+ }

1. Cut the dataset to 60 folds.

> #Choose 60-fold datasets split (5\*12)- take the row number as the index of 60 fold datasets

> folds <- cut(seq(1,nrow(Data)),breaks=60,labels=FALSE)

1. Calculate the accuracy for different gamma and C value based on LOOCV

Define a function with the input of Data, C.range, and gamma.range to calculate the predicted accuracy.

> optimal\_LOOV<-function(Data, C.range, gamma.range){

+ N<-length(C.range)

+ M<-length(gamma.range)

+ tot.accuracy<-matrix(0,N,M)

+ pb = txtProgressBar(min = 0, max = N, initial = 0)

+ for (i in 1:N){

+ C<-C.range[i]

+ setTxtProgressBar(pb,i)

+ for (j in 1:M){

+ gamma<-gamma.range[j]

+ correct<-numeric(60)

For different C and gamma value, conduct LOOCV to calculate the average predicted accuracy.

+ for (k in 1:60){

+ #Segement the data by fold using the which() function

+ testIndexes <- which(folds==k,arr.ind=TRUE)

+ testData <- Data[testIndexes, ]

+ trainData <- Data[-testIndexes, ]

Subset the X and Y from the training and test dataset

+ train.X<-trainData[,2:5]

+ train.Y<-trainData[,1]

+ test.X<-testData[2:5]

+ test.Y<-testData[1]

Call the LS-SVM function to calculate the predicted accuracy for each fold.

+ correct[k]<-lssvmmodel\_LOOCV(train.X, train.Y, test.X, test.Y, 500000, gamma, C)

+ }

+ tot.accuracy[i,j]<-sum(correct)/60

+ }

+ }

+ return(tot.accuracy)

+ }

1. Define the range of C value and gamma value

Define the range of C value and gamma value, we defined the range for C value is (1, 10), and the range for gamma is (0.1, 5)

> C.range<-seq(1,10, by=1)

> gamma.range<-seq(0.1,4, by=0.1)

1. Call the above function to generate the accuracy according to different C and gamma

> test\_LOOCV<-optimal\_LOOCV(Data, C.range, gamma.range)

> test\_LOOV

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]

[1,] 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667

[2,] 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667

[3,] 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667

[4,] 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667

[5,] 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667

[6,] 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667

[7,] 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667

[8,] 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667

[9,] 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667

[10,] 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667 0.5166667

[,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16]

[1,] 0.5166667 0.5166667 0.5166667 0.5166667 0.5333333 0.5666667 0.5666667 0.5666667

[2,] 0.5166667 0.5166667 0.5166667 0.5333333 0.5500000 0.5666667 0.5666667 0.5666667

[3,] 0.5166667 0.5166667 0.5166667 0.5333333 0.5500000 0.5666667 0.5833333 0.5666667

[4,] 0.5166667 0.5166667 0.5166667 0.5500000 0.5500000 0.5666667 0.5833333 0.5666667

[5,] 0.5166667 0.5166667 0.5166667 0.5500000 0.5666667 0.5666667 0.5833333 0.5666667

[6,] 0.5166667 0.5166667 0.5166667 0.5500000 0.5666667 0.5666667 0.5833333 0.5666667

[7,] 0.5166667 0.5166667 0.5166667 0.5500000 0.5666667 0.5666667 0.5833333 0.5666667

[8,] 0.5166667 0.5166667 0.5166667 0.5500000 0.5666667 0.5666667 0.5833333 0.5666667

[9,] 0.5166667 0.5166667 0.5166667 0.5500000 0.5666667 0.5666667 0.5833333 0.5666667

[10,] 0.5166667 0.5166667 0.5166667 0.5500000 0.5666667 0.5666667 0.5833333 0.5666667

[,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]

[1,] 0.5666667 0.5666667 0.5666667 0.6000000 0.6166667 0.6166667 0.6333333 0.6333333

[2,] 0.5666667 0.5666667 0.6166667 0.6166667 0.6166667 0.6333333 0.6333333 0.6333333

[3,] 0.5666667 0.5666667 0.6166667 0.6166667 0.6166667 0.6333333 0.6333333 0.6166667

[4,] 0.5666667 0.5666667 0.6166667 0.6166667 0.6166667 0.6333333 0.6333333 0.6166667

[5,] 0.5666667 0.5833333 0.6166667 0.6166667 0.6166667 0.6333333 0.6333333 0.6166667

[6,] 0.5500000 0.5833333 0.6166667 0.6166667 0.6166667 0.6333333 0.6333333 0.6166667

[7,] 0.5500000 0.6000000 0.6166667 0.6166667 0.6166667 0.6333333 0.6333333 0.6333333

[8,] 0.5500000 0.6000000 0.6166667 0.6166667 0.6166667 0.6333333 0.6333333 0.6333333

[9,] 0.5500000 0.6000000 0.6166667 0.6166667 0.6166667 0.6333333 0.6333333 0.6333333

[10,] 0.5500000 0.6166667 0.6166667 0.6166667 0.6166667 0.6333333 0.6333333 0.6333333

[,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32]

[1,] 0.6166667 0.6166667 0.6500000 0.6500000 0.6666667 0.7000000 0.7000000 0.7166667

[2,] 0.6166667 0.6500000 0.6500000 0.6666667 0.7000000 0.7000000 0.7166667 0.7166667

[3,] 0.6166667 0.6500000 0.6500000 0.6666667 0.7000000 0.7000000 0.7166667 0.7333333

[4,] 0.6166667 0.6500000 0.6500000 0.6833333 0.7000000 0.7166667 0.7166667 0.7333333

[5,] 0.6166667 0.6500000 0.6500000 0.7000000 0.7000000 0.7166667 0.7166667 0.7333333

[6,] 0.6166667 0.6500000 0.6666667 0.7000000 0.7000000 0.7166667 0.7166667 0.7333333

[7,] 0.6333333 0.6500000 0.6666667 0.7000000 0.7000000 0.7166667 0.7166667 0.7333333

[8,] 0.6333333 0.6500000 0.6666667 0.7000000 0.7000000 0.7166667 0.7166667 0.7333333

[9,] 0.6333333 0.6500000 0.6666667 0.7000000 0.7000000 0.7166667 0.7166667 0.7333333

[10,] 0.6333333 0.6500000 0.6666667 0.7000000 0.7000000 0.7166667 0.7166667 0.7333333

[,33] [,34] [,35] [,36] [,37] [,38] [,39] [,40]

[1,] 0.7333333 0.7333333 0.7333333 0.7333333 0.7333333 0.7500000 0.7666667 0.7500000

[2,] 0.7333333 0.7333333 0.7333333 0.7333333 0.7500000 0.7666667 0.7666667 0.7666667

[3,] 0.7333333 0.7333333 0.7333333 0.7500000 0.7666667 0.7666667 0.7666667 0.7666667

[4,] 0.7333333 0.7333333 0.7333333 0.7500000 0.7666667 0.7666667 0.7666667 0.7666667

[5,] 0.7333333 0.7333333 0.7500000 0.7500000 0.7666667 0.7666667 0.7666667 0.7666667

[6,] 0.7333333 0.7333333 0.7500000 0.7500000 0.7666667 0.7666667 0.7666667 0.7666667

[7,] 0.7333333 0.7333333 0.7500000 0.7500000 0.7666667 0.7666667 0.7666667 0.7666667

[8,] 0.7333333 0.7333333 0.7500000 0.7500000 0.7666667 0.7666667 0.7666667 0.7666667

[9,] 0.7333333 0.7333333 0.7500000 0.7500000 0.7666667 0.7666667 0.7666667 0.7666667

[10,] 0.7333333 0.7333333 0.7500000 0.7500000 0.7666667 0.7666667 0.7666667 0.7666667

1. Index the best accuracy value in the result matrix, and then pick up the corresponding C and gamma value

Index the maximum value in the accuracy matrix

> valueindex\_LOOCV <-which(test\_LOOCV == max(test\_LOOCV), arr.ind = TRUE)

> valueindex\_LOOCV

row col

[1,] 3 37

[2,] 4 37

[3,] 5 37

[4,] 6 37

[5,] 7 37

[6,] 8 37

[7,] 9 37

[8,] 10 37

[9,] 2 38

[10,] 3 38

[11,] 4 38

[12,] 5 38

[13,] 6 38

[14,] 7 38

[15,] 8 38

[16,] 9 38

[17,] 10 38

[18,] 1 39

[19,] 2 39

[20,] 3 39

[21,] 4 39

[22,] 5 39

[23,] 6 39

[24,] 7 39

[25,] 8 39

[26,] 9 39

[27,] 10 39

[28,] 2 40

[29,] 3 40

[30,] 4 40

[31,] 5 40

[32,] 6 40

[33,] 7 40

[34,] 8 40

[35,] 9 40

[36,] 10 40

Pick up the corresponding C value

> C.list<-C.range[valueindex\_LOOCV [,1]]

> C.list

> C.list

[1] 4 5 6 7 8 9 10 2 3 1

Pick up the corresponding gamma value

> gamma.list<-gamma.range[valueindex\_LOOCV [,2]]

> gamma.list

[1] 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.6 3.6 3.7

The above results indicate that the value of C does not make much sense to the predicted accuracy, however the gamma value around 3.5-3.7 could have highest prediction performance.

**Problem 2**

The second part of this project will deal with “tensors”. Tensor rather than vector is becoming an important tool in Machine Learning or Deep Learning. For this problem, you have to use tensor packages from R or Python. 2 tensor packages from “rTensor” and “TensorA” are available on the class webcourse.

**Part I**

In this part, we will generate i × j × k numbers and rearrange these numbers as one entry of a tensor T (i, j, k). More details can be found in the file ‘tensor1’ on the class webcourse.

1. Let U be a matrix of ones. You will need to find out the dimension of U yourself. Our goal is to find the “Mode-n product: tensor matrix multiplication” of the tensor T by the matrix U. Find A1 = T ×1 U and A2 = T ×2 U
2. Generate 1000 numbers from normal distribution, then arrange these numbers to a tensor of (5, 5, 40);

> Tensor <- new("Tensor",3L,c(5L,5L,40L),data=rnorm(1000,0,1))

>

1. Generate a matrix U with the column number equals to the 1th model of Tensor;

> U <- matrix(rep(1,100),ncol=5)

>

> ##Calculate teh Mode-1 product

> A1<-ttm(Tensor,U,m=1)

> A1

Numeric Tensor of 3 Modes

Modes: 20 5 40

Data:

[1] 5.828748 5.828748 5.828748 5.828748 5.828748 5.828748

1. Calculate the Mode-2 product, in this case, the 1th mode should 2th mode;

> A2<-ttm(Tensor,U,m=2)

> A2

Numeric Tensor of 3 Modes

Modes: 5 20 40

Data:

[1] 1.4504970 1.6016129 2.3474211 4.4324680 0.7555424 1.4504970

1. Next, Let u be a vector of ones. You will need to fi out the dimen- sion of u yourself. Our goal is to fi the “Mode-n product: tensor vec- tor multiplication” of the tensor T by the vector u. Find B1 = T ×1 u and B2 = T ×2 u
2. Generate a vector u with the column number equals to the 1th model of Tensor

> u <- matrix(rep(1,5),ncol=5,nrow=1)

>

> B1<-ttm(Tensor,u,m=1)

> B1

Numeric Tensor of 3 Modes

Modes: 1 5 40

Data:

[1] 5.82874794 0.79365585 0.03015745 0.36139230 3.57358794 0.49856296

>

> B2<-ttm(Tensor,u,m=2)

> B2

Numeric Tensor of 3 Modes

Modes: 5 1 40

Data:

[1] 1.4504970 1.6016129 2.3474211 4.4324680 0.7555424 1.8183275

1. Write a function to compute B = T ×1 u1 ×2 u2 . . . ×k uk , where the ui are vectors of ones.
2. Develop a function to reduce the dimension of tensor, with the input of a tensor

> tensor\_reduce<-function(Tensor){

+ N<-dim(Tensor)

+ B<-Tensor

+ N<-as.vector(N)

+ M<-length(N)

+ for (k in 1:M){

+ u<-matrix(rep(1,N[k]),ncol=N[k],nrow=1)

+ B<-ttm(B,u,m=k)

+ }

+ return(B)

+ }

1. Randomly generate a 3-mode tensor and then call the above function

> A<-rand\_tensor(modes = c(3, 4, 5), drop = FALSE)

> tensor\_reduce(A)

Numeric Tensor of 3 Modes

Modes: 1 1 1

Data:

[1] -11.29692

**Part II**

In this part, we will represent a social web as a tensor. More details can be found in the file ‘social web tensor’ on the class webcourse. Read the details in section 3.2 on page 48. Our challenge now is to perform data collection step only. Apply the techniques described in section 3.3 on page 51.

In this problem, the data from the following table was utilized.

Table 1. Simple strategies for to-follow recommendations, h = #semanticweb

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| POS | WeFollow recommends | h-set recommends | followers | | H-set recommends | followers | |
| total | in h-set | total | in H-set |
| 1 | tommyh | PaulMiller | 2215 | 82 | timberners\_lee | 20694 | 252 |
| 2 | jahendler | jahendler | 909 | 76 | timoreilly | 1428425 | 200 |
| 3 | ivanherman | tommyh | 738 | 76 | jahendler | 910 | 185 |
| 4 | PaulMiller | ivan\_herman | 680 | 69 | LeeFeigenbaum | 273 | 176 |
| 5 | opencalais | opencalais | 1797 | 68 | danbri | 1781 | 160 |
| 6 | danja | danja | 1313 | 59 | kidehen | 1806 | 159 |
| 7 | CaptSolo | juansequeda | 988 | 57 | ivan\_herman | 680 | 153 |
| 8 | juansequeda | CaptSolo | 1224 | 52 | PaulMiller | 2216 | 150 |
| 9 | sclopit | gothwin | 685 | 50 | tommyh | 737 | 142 |
| 10 | gothwin | robocrunch | 3679 | 49 | novaspivack | 7896 | 139 |
| 11 | robocrunch | alexiskold | 4321 | 48 | johnbreslin | 1933 | 128 |
| 12 | kristathomas | kristathomas | 1499 | 48 | w3c | 10672 | 128 |
| 13 | kendall | kendall | 1694 | 45 | mimasnews | 198 | 127 |
| 14 | bobdc | andraz | 2065 | 44 | iand | 1094 | 123 |
| 15 | phclouin | sclopit | 513 | 42 | rww | 1045511 | 123 |
| 16 | brown2020 | cjmconnors | 466 | 39 | terraces | 632 | 119 |
| 17 | alexiskold | gkob | 399 | 37 | mhausenblas | 449 | 118 |
| 18 | andraz | dorait | 2541 | 35 | opencalais | 1799 | 112 |
| 19 | cjmconnors | phclouin | 266 | 35 | ldodds | 621 | 110 |
| 20 | ontoligent | openamplify | 1387 | 34 | semanticnews | 843 | 102 |