**Jinghui Yuan 4177204**

**Problem 1**

We want to solve the following LS-SVM problem in R or Python or both using the dataset “pb2.txt”:

1. Use the large scale algorithm (Hestenes-Stiefel algorithm) discussed in Suykens et al. (1999), available on class webcourse, to solve the LS- SVM problem. Provide the training and prediction codes.

In this problem, there are six steps to finish the training process of LS-SVM, and then the result could be used to predict any vectors.

1. Import the data and convert the data to matrix format;

> 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 to get matrix X and vector Y;

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

> Y <- Data[,1]

> Y<-as.vector(Y)

> X <- as.matrix(Data[,2:5],ncol=4)

> N <- length(Y)

> for (i in 1:N){

+ if (Y[i] > 1){

+ Y[i]<--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. Generate the matrix of H and d2;

> gamma<-1.5

> 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,],X[j,],gamma)

+ }

+ }

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

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

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 a large space to store the temporary vector.

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

+ i<-1

+ x<-matrix(0,62,N)

+ r<-matrix(0,62,N)

+ p<-matrix(0,62,N)

+ beta<-numeric(N)

+ lamda<-numeric(N)

Define the initial value of r [,1] is vector B.

+ r[,1]<-B

Do while the inner product of r [, i] is greater than zero.

+ while (t(r[,i])%\*%r[,i]>0){

+ i<-i+1

+ if (i==2){

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

Define the function of LS-SVM training process with the input of matrix H, vector Y and d2, and a predefined large value, which represents the maximum steps of the iterations.

> lssvmtrain<-function(H, Y, d2, step){

Call the function of conjugate gradient method to solve the formula of .

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

Call the function of conjugate gradient method to solve the formula of .

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

Compute .

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

Find the solution .

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

Find the solution .

+ alpha<-vta-nta\*b

+ alpha<-as.vector(alpha)

Return and b.

+ list(alpha=alpha, b=b)

+ }

Call the function of LSSVM

> model<-lssvmtrain(H, Y, d2, 5000)

Show the training results of LSSVM given the training dataset of pb2.txt

> model

$alpha

[1] 0.6209580 0.5821270 0.5108039 0.6209724 0.6209354 0.6208125 0.4937116 0.6209636 0.6209636 0.5422552 0.6203481 0.6248321 0.3158152 0.5025541

[15] 0.4841169 0.4947374 0.6210115 0.5934147 0.6181902 0.5987957 0.6231194 0.6257878 0.6209559 0.6165021 0.5702146 0.5827981 0.4389975 0.5926431

[29] 0.5913158 0.6209667 0.5143790 0.5790331 0.5789065 0.5727048 0.5790331 0.5790366 0.5790321 0.5789829 0.5727427 0.5789879 0.5723189 0.5790056

[43] 0.5790364 0.5790451 0.5788412 0.5565510 0.5834415 0.5801823 0.5790004 0.5039796 0.5790364 0.5270967 0.5782717 0.5801856 0.5731555 0.5790364

[57] 0.5241321 0.5790840 0.5790301 0.5790364 0.5790364 0.5790364

$b

[,1]

[1,] -0.03493935

1. Predict the class of an object X

Define the function of LSSVM-Predict to predict the class of an object based on the training dataset and the training model results.

> lssvmpredict <- function(Xv,Yv,x,model,N){

+ alpha<-model$alpha

+ b<-model$b

+ gamma<-1.5

+ ayK<-numeric(N)

Calculate the .

+ for(i in 1:N){

+ ayK[i]<-alpha[i]\*Yv[i]\*rbf\_kernel(Xv[i,],x,gamma)

+ }

Predict the sign of .

+ result <- sign(sum(ayK)+b)

+ return(result)

+ }

Given the test vector of (16, 13, 16, 14).

> z <- c(16,13,16,14)

Call the function of LSSVM-Predict to predict the sign of given vector, the result indicates that the predicted sign is consistent with the sample sign.

> lssvmpredict(X,Y,z,model,62)

[,1]

[1,] -1

Given another test vector of (18, 17, 33, 26).

> z <- c(18,17,33,26)

Call the function of LSSVM-Predict to predict the sign of given vector, the result indicates that the predicted sign is consistent with the sample sign.

> lssvmpredict(X,Y,z,model,62)

[,1]

[1,] 1

1. Write a code for updating the QR decomposition after adding one row to the training set. (qr.update on class webcourse)
2. Define the QR composition function to generate the Q and R given data X and Y

> QR\_Com<-function(X,Y,gamma){

Calculate the matrix , while

+ A\_up<-append(0,Y)

+ N<-length(Y)

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

+ for(i in 1:N){

+ for(j in 1:N){

+ omega[i,j]<-Y[i]\*Y[j]\*rbf\_kernel(X[i,],X[j,],gamma)

+ }

+ }

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

+ A\_down<-cbind(Y,H)

+ A<-rbind(A\_up,A\_down)

+ A<-as.matrix(A)

Generate Q and R for matrix A

+ qr<-qr(A)

+ Q<-qr.Q(qr)

+ R<-qr.R(qr)

+ list(Q=Q,R=R)

+ }

1. Define the Givens function based on the algorithm 1.1 in the material of “qr\_updata” to return c and s when given a and b.

> Givens<-function(a,b){

+ if (b==0){

+ c<-1

+ s<-0

+ }

+ else {

+ if (abs(b)>=abs(a)){

+ t<-(-a/b)

+ s<-1/sqrt(1+t^2)

+ c<-s\*t

+ }

+ else {

+ t<-(-b/a)

+ c<-1/sqrt(1+t^2)

+ s<-c\*t

+ }

+ }

+ list(c=c,s=s)

+ }

1. Define QR updata function to generate the updated Q and R when given the original data and adding data.

> qrupdate<-function(X,Y,X\_add,Y\_add,gamma){

Get the new row (*u*) of matrix A, after adding one row to the sample size (X and Y).

+ n<-nrow(X)

+ u\_t<-matrix(0,1,n)

+ for (i in 1:n) {

+ u\_t[i]<-Y\_add\*Y[i]\*rbf\_kernel(X\_add,X[i,],gamma)

+ }

+ u<-append(Y\_add,u\_t)

Generate the Q and R for the original dataset of X and Y

+ Q<-QR\_Com(X,Y,gamma)$Q

+ R<-QR\_Com(X,Y,gamma)$R

Generate the matrix

+ n\_q<-n+1

+ Q\_1<-rbind(Q,matrix(0,1,n\_q))

+ Q\_1<-cbind(Q\_1,matrix(0,n\_q+1,1))

+ Q\_1[n\_q+1,n\_q+1]<-1

**Update Q and R by adding a new row *u***

+ ##Update Q and R by adding a new row

Predefine list c and s

+ c<-numeric(n\_q)

+ s<-numeric(n\_q)

+ for (j in 1:n\_q) {

Call the Givens function to generate c and s for given value of R and *u*

+ c[j]<-Givens(R[j,j],u[j])$c

+ s[j]<-Givens(R[j,j],u[j])$s

Update the diagonal value of R

+ R[j,j]<-c[j]\*R[j,j]-s[j]\*u[j]

Update *j*th row of R and *u*

+ if (j<n\_q){

+ t1<-R[j,(j+1):n\_q]

+ t2<-u[(j+1):n\_q]

+ R[j,(j+1):n\_q]=c[j]\*t1-s[j]\*t2

+ u[(j+1):n\_q]=s[j]\*t1+c[j]\*t2

+ }

Update Q

+ t1\_q<-Q\_1[1:(n\_q+1),j]

+ t2\_q<-Q\_1[1:(n\_q+1),(n\_q+1)]

+ Q\_1[1:(n\_q+1),j]<-c[j]\*t1\_q-s[j]\*t2\_q

+ Q\_1[1:(n\_q+1),(n\_q+1)]<-s[j]\*t1\_q+c[j]\*t2\_q

+ }

Update

+ R\_1<-rbind(R,matrix(0,1,n\_q))

**Update Q and R by adding a new column**

Get the new column (*u2*) of matrix A, after adding one row to the sample size (X and Y).

+ u2<-append(u\_t,Y\_add\*Y\_add\*rbf\_kernel(X\_add,X\_add,gamma))

+ u2<-append(Y\_add,u2)

+ u2<-t(Q\_1)%\*%u2

Predefine list c1 and s1

+ m<-nrow(R\_1)

+ k<-ncol(R\_1)+1

+ c1<-numeric(k-m+1)

+ s1<-numeric(k-m+1)

+ for (i in m:k) {

Update *u2*

+ c1[i]<-Givens(u2[i-1],u2[i])$c

+ s1[i]<-Givens(u2[i-1],u2[i])$s

+ u2[i]<-c1[i]\*u2[i-1]-s1[i]\*R\_1[(i-1),(i-1)]

Update

+ R\_1<-cbind(R\_1,matrix(u2,m,1))

Update

+ Q\_1[1:m,(i-1):i]<-Q\_1[1:m,(i-1):i]%\*%matrix(c(c[j],-s[j],s[j],c[j]),2,2)

+ }

+ list(Q\_1=Q\_1,R\_1=R\_1)

+ }

1. Testing the QR update algorithm to generate the updated Q and R, after adding one row.

Select the original dataset, we select the first 5 row from the raw data.

> X1<-head(X,5)

> Y1<-head(Y,5)

Select the added sample row, we select the 6th row in the raw data.

> X\_add<-X[6,]

> Y\_add<-Y[6]

Call the QR update function to generate the updated Q and R, after adding the 6th row.

> Q\_1<-qrupdate(X1,Y1,X\_add,Y\_add,1.5)$Q\_1

> R\_1<-qrupdate(X1,Y1,X\_add,Y\_add,1.5)$R\_1

Call the QR composition function to generate the Q and R for the first 6 rows data.

> X2<-head(X,6)

> Y2<-head(Y,6)

> Q<-QR\_Com(X2,Y2,1.5)$Q

> R<-QR\_Com(X2,Y2,1.5)$R

Calculate the difference in the results between QR update function and QR composition function.

> Q\_diff<-Q\_1-Q

> R\_diff<-R\_1-R

> Q\_diff

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

[1,] 0.000000e+00 3.330669e-16 1.110223e-16 -2.220446e-16 0.7169065 -0.03125935 0.03115415

[2,] 5.551115e-17 -1.110223e-16 -5.551115e-17 -8.326673e-17 -0.4301439 0.01875561 -0.01869249

[3,] 5.551115e-17 2.775558e-17 0.000000e+00 8.326673e-17 -0.4301345 0.01875520 -0.01869208

[4,] 5.551115e-17 2.775558e-17 1.110223e-16 0.000000e+00 -0.4301345 0.01875520 -0.01869208

[5,] 5.551115e-17 2.775558e-17 1.665335e-16 1.665335e-16 1.6886910 0.01875561 -0.01869249

[6,] 5.551115e-17 2.775558e-17 2.220446e-16 5.551115e-17 -0.1991390 0.01329963 0.08836657

[7,] 5.551115e-17 1.110223e-16 1.665335e-16 1.249001e-16 -0.1991390 -0.08832124 -0.01359742

> R\_diff

Y

A\_up -4.440892e-16 2.220446e-16 2.220446e-16 4.440892e-16 3.330669e-16 1.110223e-16 0.27216553

0.000000e+00 0.000000e+00 7.771561e-16 4.996004e-16 9.992007e-16 7.216450e-16 -0.10171303

0.000000e+00 0.000000e+00 4.440892e-16 5.551115e-17 4.996004e-16 5.551115e-16 -0.08637829

0.000000e+00 0.000000e+00 0.000000e+00 4.440892e-16 8.326673e-17 -1.110223e-16 -0.07506583

0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 3.531391e+00 3.850081e-01 0.45138781

0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 -2.220446e-16 0.05949732

0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 -3.51334350

As we can see from the above comparison results, there still exist difference, while the differences are very small.

1. Our next challenge is to solve the LSSVM using matrix inversion and apply it to incremental/decremental LSSVM. Use the code from ques- tion 2 on QR update to solve the incremental LSSVM problem. (incre- ment.lssvm on class webcourse.)
2. Define the solution function with given X, Y, X\_add and Y\_add.

> Solution<-function(X1,Y1,X\_add,Y\_add,gamma){

Call the QR update function to generate the updated Q and R after adding one row

+ Q\_1<-qrupdate(X1,Y1,X\_add,Y\_add,gamma)$Q\_1

+ R\_1<-qrupdate(X1,Y1,X\_add,Y\_add,gamma)$R\_1

+ Y\_new<-append(Y1,Y\_add)

+ n<-length(Y\_new)

Generate the

+ B<-t(Q\_1)%\*%append(0,rep(1,n))

Call the function “solve” to solve the function of

+ x<-solve(R\_1,B)

+ x<-as.vector(x)

+ b<-x[1]

+ alpha<-x[2:length(x)]

+ list(b=b,alpha=alpha)

+ }

1. Use the first 61 rows of data as the original data and add the 62th row to generate the solution based on the above functions.

> X1<-head(X,61)

> Y1<-head(Y,61)

> X\_add<-X[62,]

> Y\_add<-Y[62]

> solution<-Solution(X1,Y1,X\_add,Y\_add,1.5)

> solution$b

[1] -0.02862919

> solution$alpha

[1] 0.6011549 0.5644270 0.4955194 0.6011695 0.6011341 0.6010186 0.4805972 0.6011602 0.6011602

[10] 0.5268593 0.6005791 0.6052922 0.3125963 0.4889678 0.4715789 0.4815760 0.6012111 0.5750756

[19] 0.5985368 0.5801130 0.6034487 0.6062662 0.6011529 0.5969331 0.5532803 0.5650358 0.4269678

[28] 0.5744156 0.5731631 0.6011636 0.5002466 0.5988363 0.5987019 0.5921181 0.5988363 0.5988400

[37] 0.5988352 0.5987831 0.5921587 0.5987883 0.5916470 0.5988072 0.5988398 0.5988480 0.5986326

[46] 0.5750185 0.6029948 0.5999188 0.5988016 0.5188124 0.5988398 0.5438237 0.5980288 0.5999224

[55] 0.5926094 0.5988398 0.5407042 0.5988841 0.5988331 0.5988398 0.5987536 -0.5789889

1. Compare the difference in the results between this method and the first part of Problem 1.

> alpha\_diff<-solution$alpha-model$alpha

> alpha\_diff

[1] -0.019803101 -0.017700050 -0.015284566 -0.019802877 -0.019801304 -0.019793869 -0.013114426

[8] -0.019803422 -0.019803422 -0.015395892 -0.019768999 -0.019539860 -0.003218862 -0.013586243

[15] -0.012537988 -0.013161415 -0.019800458 -0.018339134 -0.019653441 -0.018682735 -0.019670682

[22] -0.019521648 -0.019802989 -0.019569020 -0.016934379 -0.017762254 -0.012029619 -0.018227556

[29] -0.018152632 -0.019803152 -0.014132437 0.019803199 0.019795390 0.019413325 0.019803216

[36] 0.019803364 0.019803154 0.019800116 0.019415969 0.019800420 0.019328016 0.019801520

[43] 0.019803421 0.019802920 0.019791350 0.018467557 0.019553307 0.019736540 0.019801195

[50] 0.014832794 0.019803422 0.016726995 0.019757109 0.019736747 0.019453960 0.019803422

[57] 0.016572107 0.019800150 0.019803033 0.019803422 0.019717199 -1.158025336

> b\_diff

[,1]

[1,] 0.006310165

As can be seen from the above comparison, there exist minor difference between the parameter estimation results, which may not influence the accuracy of the classifier.

**Problem 2**

Use your favorite search engine (google, yahoo, bing, ...) to fi details about support vector regression (SVR). Write a code similar to the SVM discussed in class (“svmtrain.r” and “svmpredict.r”) to solve the SVR problem. The dataset to be used is the Boston Housing Dataset. This data can be obtained from R:

<https://stat.ethz.ch/R-manual/R-devel/library/MASS/html/Boston.html>

Also, this dataset can be downloaded from UCI Machine Learning Repos- itory. This dataset contains 506 cases concerning housing in Boston Mass Area. There are 14 attributes, in which housing values is output response, y, and the other 13 factors affecting the housing values are treated as input variables. To construct the SVR and evaluate its performance, randomly selected 300 cases from the dataset are used as training data, and the others are used as testing data.

In the SVR problem, the linear function could be expressed as:

The optimization problem could be expressed as:

According to the paper “Smola, A.J. and Schölkopf, B., 2004. A tutorial on support vector regression. Statistics and computing, 14(3): 199-222.” The dual optimization problem could be derived as:

This maximization is equal to the following minimization:

Since the is a small value, and may not have much influence on the results of optimization results, thus this could be deleted from the above objective function.

In the “quadprog” package, it could solve the following problems:

In order to apply this package, we should match the following items:

Since , then , this could be split to 2 conditions:

Therefore,

Based on the above derivation, we can use the package “quadprog” to solve the problem:

1. Randomly select 300 cases as the training dataset, and then leave the other 206 cases as the testing dataset.

> data2 <- read.table("C://Onedrive/OneDrive - Knights - University of Central Florida/UCF/Courses/Statistical Computing/Project1/Boston.txt")

> Data2<-data.frame(data2)

Sample 300 rows from the original dataset

> train<-Data2[sample(nrow(Data2),300),]

> Y<-train[,14]

> X <- as.matrix(train[,1:13],ncol=13)

1. Import the package “quadprog” and define the Gaussian kernel function

> require('quadprog')

> ## Defining the Gaussian kernel

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

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

+ return(K)

+ }

1. Train the SVR model based on the training dataset

> svrtrain <- function(X,Y,C=Inf, gamma=1.5,esp=1e-10){

+ N<-length(Y)

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

+ X<-as.matrix(X)

+ Y<-as.vector(Y)

Calculate the kernel matrix ***D***

+ for(i in 1:N){

+ for(j in 1:N){

+ Dm[i,j]<-rbf\_kernel(X[i,],X[j,],gamma)

+ }

+ }

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

Calculate the vector and matrix ***A*** to apply the quadprog package

+ dv<-t(Y)

+ meq<-1

+ Am<-cbind(matrix(rep(1,N),N),diag(N))

+ if(C!=Inf){

+ Am<-cbind(Am,-1\*diag(N))

+ bv<-append(0,rep(-C,2\*N))

+ }

+ alpha\_org<-solve.QP(Dm,dv,Am,meq=meq,bvec=bv)$solution

Based on the results of quadprog, select all the support vectors and the corresponding

+ indx<-which(alpha\_org>esp,arr.ind=TRUE)

+ alpha<-alpha\_org[indx]

+ nSV<-length(indx)

+ if(nSV==0){

+ throw("QP is not able to give a solution for these data points")

+ }

+ Xv<-X[indx,]

+ Yv<-Y[indx]

+ Yv<-as.vector(Yv)

+ ## choose one of the support vector to compute b. Instead of using an arbitrary Support

+ ##Vector xs, it is better to take an average over all of the Support Vectors in S

Based on the selected support vectors to calculate the b

+ b <- numeric(nSV)

+ ayK <- numeric(nSV)

+ for (i in 1:nSV){

+ for (m in 1:nSV){

+ ayK[m] <- alpha[m]%\*%rbf\_kernel(Xv[m,],Xv[i,],gamma)

+ }

+ b[i]<-Yv[i]-sum(ayK)

+ }

+ w0 <- mean(b)

+ #list(alpha=alpha, wstar=w, b=w0, nSV=nSV, Xv=Xv, Yv=Yv, gamma=gamma)

+ list(alpha=alpha, b=w0, nSV=nSV, Xv=Xv, Yv=Yv, gamma=gamma)

+ }

Call the training function to train the sample dataset

> model <-svrtrain(X,Y,C=12,gamma=24,esp=1e-10)

1. Define a SVR prediction function

> svrpredict <- function(x,model){

Call the support vectors estimation results from the training model (**, b, X, Y**)

+ alpha<-model$alpha

+ b<-model$b

+ Yv<-model$Yv

+ Xv<-model$Xv

+ nSV<-model$nSV

+ gamma<-model$gamma

For each test sample, calculate the predict value based on all support vectors.

+ N<-nrow(x)

+ ayK <- numeric(nSV)

+ result<-numeric(N)

+ for(k in 1:N){

+ for(i in 1:nSV){

+ ayK[i]<-alpha[i]\*rbf\_kernel(Xv[i,],x[k,],gamma)

+ }

+ result[k] <- sum(ayK)+b

+ }

+ return(result)

+ }

1. Predict the Y value for the testing dataset

> test<-Data2[-train.rows,]

> Yt<-test[,14]

> Xt<- as.matrix(test[,1:13],ncol=13)

> Predict<-svrpredict(Xt,model)

1. Evaluate the model performance based on MAE and RMSE

> N<-length(Yt)

> Tot\_MAE<-0

> Tot\_RMSE<-0

> for (i in 1:N){

+ Tot\_MAE<-Tot\_MAE+abs(Yt[i]-Predict[i])

+ Tot\_RMSE<-Tot\_RMSE+(Yt[i]-Predict[i])^2

+ }

> MAE<-Tot\_MAE/N

> RMSE<-sqrt(Tot\_RMSE/N)

> MAE

[1] 31.28341

> RMSE

[1] 37.26328

The evaluation results show that the MAE value is 31.28341 and the RMSE is 37.26328, these values indicate that the model performance is not good.