Albert Osom
STA 6107
Report

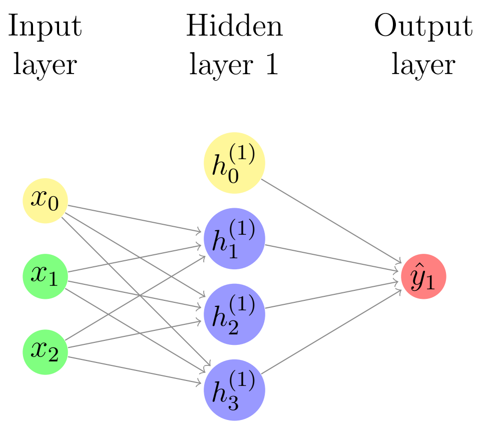
Problem 1

The goal is to estimate the price of new housing in any given city at the design phase or beginning of the construction. Our response or dependent variable is 'Sale Prices'. Use only the first two continuous variables without missing values. We will build our models based on the first 50 observations (training set).
1. Obtain a one layer neural network for example 1 (example using Win and Wout with the sigmoid activation function)
2. Obtain a one layer neural network for example 2 (example using tanh activation function)

1.1

Obtain a one layer neural network for example 1 (example using Win and Wout with the sigmoid activation function)

Solution



Obtaining a one layer neural network is reduced to solving an optimization problem to obtain $W_{in}, W_{out}, b_1, b_2$. To solve this problem, we will be using the squared error loss i.e

$$loss(l) = \frac{(Y - \hat{Y})'(Y - \hat{Y})}{N}$$

Lets first make the following denotations

$$\begin{aligned} Z_1 &= XW_{in} + b_1 & A_1 &= \sigma(Z_1) \quad (\text{ activation function}) \\ \hat{Y} = Z_2 &= A_1 W_{out} + b_2 \end{aligned}$$

We obtain partial derivatives wrt to our parameters of interest as follows

$$\begin{aligned}
\frac{\partial l}{\partial W_{out}} &= \frac{-2}{N} A_1'(Y - Z_2) \\
\frac{\partial l}{\partial W_{in}} &= \frac{-2}{N} X'(Y - Z_2) W_{out}' A_1(\mathbf{1} - A_1) \\
\frac{\partial l}{\partial b_1} &= \frac{-2}{N} \mathbf{1}_N'(Y - Z_2) W_{out}' A_1(\mathbf{1} - A_1) \\
\frac{\partial l}{\partial b_2} &= \frac{-2}{N} (Y - Z_2)' \mathbf{1}_N
\end{aligned}$$

The following gradient descent algorithm obtains these values

1. Set $t = 0$ and initialize $W_{out}^{(0)}, W_{in}^{(0)}, b_1^{(0)}, b_2^{(0)}$. Choose ϵ to check for convergence
2. For $t = 1, ..., T$
   - $W_{out}^{(t+1)} \leftarrow W_{out}^{(t)} - \eta \frac{\partial l}{\partial W_{out}}$ $\quad \eta$ is fixed learning rate
   - $b_2^{(t+1)} \leftarrow b_2^{(t)} - \eta \frac{\partial l}{\partial b_2}$
   - $W_{in}^{(t+1)} \leftarrow W_{in}^{(t)} - \eta \frac{\partial l}{\partial W_{in}}$
   - $b_1^{(t+1)} \leftarrow b_1^{(t)} - \eta \frac{\partial l}{\partial b_1}$
3. if $(loss(t+1) - loss(t)) \leq \epsilon$ stop;
   else set $t \leftarrow t + 1$ and return to step 2

We now Implement this algorithm in R and output the MSEs to choose the efficient number of neurons in the hidden layer. We initialize weights by randomly sampling from the uniform(0,1).

Before we implement this algorithm, we first define a function that computes the MSEs using chosen weights. This function will help us monitor our MSEs when our model is learning

```
NN1.error<-function(X,y,w1,b1,w2,b2){

    Z1<-X%*%W1 + matrix(b1,nrow(X),ncol=ncol(W1),byrow=TRUE)

    ### activation function layer 1

    A1<-matrix(0,ncol=ncol(W1),nrow=nrow(X))

    for (j in 1:ncol(A1)){A1[,j]<-sigmoid(Z1[,j])}

    ## Score function

    Z2<-A1%*%W2 + rep(b2,nrow(A1))

    mse<-mean((y.train-Z2)^2)
    return(mse)
}
```

Lets now implement the algorithm to learn our Neural Network. Choosing our fixed learning rate ($\eta = 0.2$)

```
NN1.sigmoid<-function(units=3,iter=5000,eta=0.1){

     mse.model<-numeric(iter)

     W1<-matrix(runif(units*ncol(X.train)),ncol=units)
     W2<-matrix(runif(units),ncol=1)

     b1<-runif(units)
     W1.old<-W1
     b1.old<-b1
     b2<-runif(1)
     W2.old<-W2
     b2.old<-b2

     N<-nrow(X.train)

for (i in 1:iter) {

    Z1<-X.train%*%W1.old + matrix(b1.old,nrow(X.train),ncol=ncol(W1.old),byrow=TRUE)

    ### activation function layer 1

    sigmoid<-function(x){ 1/(1+exp(-x))}

    A1<-matrix(0,ncol=ncol(W1.old),nrow=nrow(X.train))

    for (j in 1:ncol(A1)){A1[,j]<-sigmoid(Z1[,j])}

    Z2<- A1%*%W2.old + rep(b2.old,nrow(A1))

    dldw2<--(2/N)*t(A1)%*%(y.train-Z2)
    dldb2<- crossprod(-(2/N)*(y.train-Z2),rep(1,length(y.train)))

    dldw1<--(2/N)*t(X.train)%*% ((y.train-Z2)%*%t(W2.old)*((A1)*(1-A1)))

    ones<-matrix(1,nrow=nrow(A1),ncol=1)

    dldb1<--(2/N)*t(ones)%*%(((y.train-Z2)%*%t(W2.old))*((A1)*(1-A1)))

    W1.new<-W1.old-eta*(dldw1)
    b1.new<-b1.old-eta*(dldb1)
    W2.new<-W2.old-eta*(dldw2)
    b2.new<-b2.old-eta*(dldb2)

    mse.new<-NN.error(X.train,Y.train,W1.new,b1.new,W2.new,b2.new)
    mse.model[i]<- mse.new

    #print(mse.new)

    #if ((mse.mew-mse.old)< 4){
      #  break
   # }
    W1.old<-W1.new
    W2.old<-W2.new
    b1.old<-b1.new
    b2.old<-b2.new
    mse.old<-mse.new
}
    return(mse.model)
}
```

We know observe the plot of the MSEs for number of neurons (3,5,10,15) on our training set. These plots can also inform us on the number of iterations to stop.

```
units<-c(3,5,10,15)
for(i in units) {
    model3<-NN1.sigmoid(i,eta=0.2,iter=5000)
    plot(model3,type="l",main=paste(i,"Neurons in Hidden layer"),ylab="MSE Error")
}
```

We realize from the plots that, we can choose the model with 5 neurons and have an early stopping before 1000 iterations.

Now lets compare the MSEs before and after the model has learnt. Before we do that,we will add a stopping rule to our Neural Network model above. That is, we stop after 500 iterations and the difference in MSE is less than 0.001

```
NN1.sigmoid<-function(units=3,iter=5000,eta=0.1){

    mse.model<-numeric(iter)

    W1<-matrix(runif(units*ncol(X.train)),ncol=units)
    W2<-matrix(runif(units),ncol=1)

    b1<-runif(units)
    W1.old<-W1
    b1.old<-b1
    b2<-runif(1)
    W2.old<-W2
```

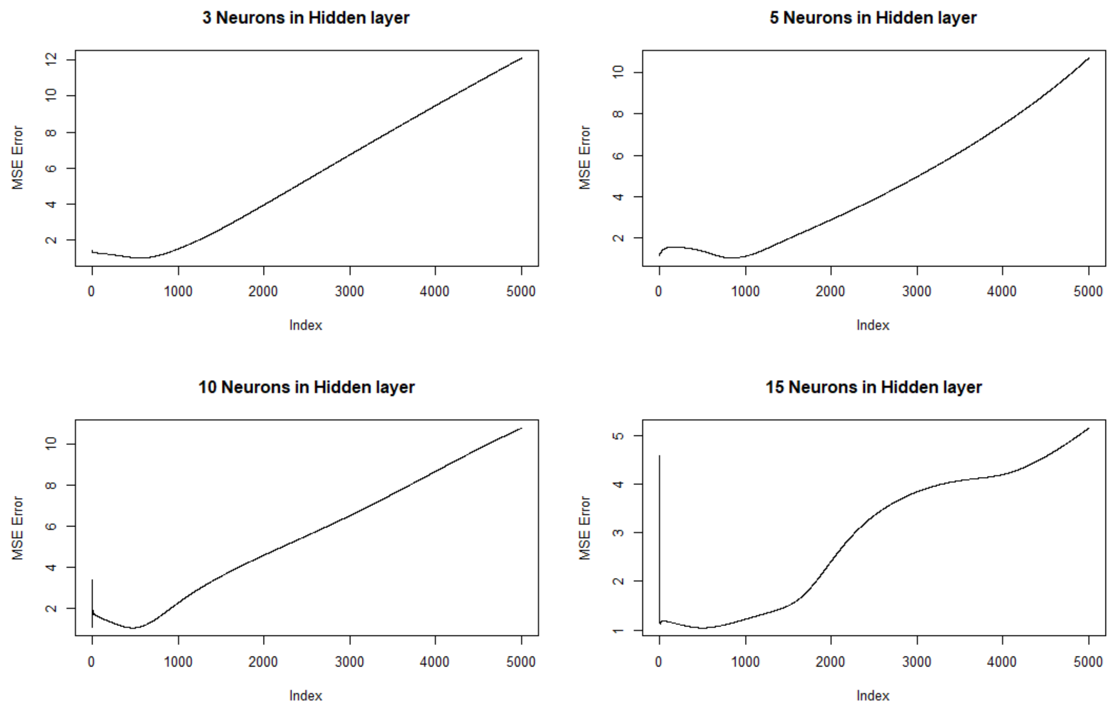


Figure 1: *Plots of several neurons*

```
b2.old<-b2

N<-nrow(X.train)

mse.old<-NN.error(X.train,Y.train,W1.old,b1.old,W2.old,b2.old)

for (i in 1:iter) {

    Z1<-X.train%*%W1.old + matrix(b1.old,nrow(X.train),
    ncol=ncol(W1.old),byrow=TRUE)

    ### activation function layer 1

    sigmoid<-function(x){ 1/(1+exp(-x))}

    A1<-matrix(0,ncol=ncol(W1.old),nrow=nrow(X.train))

    for (j in 1:ncol(A1)){A1[,j]<-sigmoid(Z1[,j])}

    Z2<- A1%*%W2.old + rep(b2.old,nrow(A1))

    dldw2<--(2/N)*t(A1)%*%(y.train-Z2)
    dldb2<- crossprod(-(2/N)*(y.train-Z2),rep(1,length(y.train)))
```

```
        dldw1<--(2/N)*t(X.train)%*% ((y.train-Z2)%*%t(W2.old)*((A1)*(1-A1)))

        ones<-matrix(1,nrow=nrow(A1),ncol=1)

        dldb1<--(2/N)*t(ones)%*%(((y.train-Z2)%*%t(W2.old))*((A1)*(1-A1)))

        W1.new<-W1.old-eta*(dldw1)
        b1.new<-b1.old-eta*(dldb1)
        W2.new<-W2.old-eta*(dldw2)
        b2.new<-b2.old-eta*(dldb2)

        mse.new<-NN.error(X.train,Y.train,W1.new,b1.new,W2.new,b2.new)

        mse.model[i]<- mse.new

        #print(mse.new)

        if (500 < i & i < 1000 & (mse.new-mse.old)< 0.001){
            break
        }
        W1.old<-W1.new
        W2.old<-W2.new
        b1.old<-b1.new
        b2.old<-b2.new
        mse.old<-mse.new
    }
    return(list(model=mse.model,W1.new=W1.new,
    b1.new=b1.new,W2.new=W2.new,b2.new=b2.new,
    W1=W1,b1=b1,W2=W2,b2=b2,iteration=i))

}
```

Comparing the MSEs before and after our model has learnt.

```
model3<-NN1.sigmoid(units=5,eta=0.2,iter=1000)

W1.new<-model3$W1.new
b1.new<-model3$b1.new
W2.new<-model3$W2.new
b2.new<-model3$b2.new

W1.old<-model3$W1
b1.old<-model3$b1
W2.old<-model3$W2
b2.old<-model3$b2
iterations<-model3$iteration
```

```
### Predicting  ###

###### before model #####

#model3$model[1000]

mse.before<-NN.error(X.train,Y.train,W1.old,b1.old,W2.old,b2.old)
mse.before
[1] 3.489708

##### After model ###########

mse.after<-NN.error(X.train,Y.train,W1.new,b1.new,W2.new,b2.new)
mse.after
[1] 1.143464
```

1.2

Obtain a one layer neural network for example 1 (example using Win and Wout with the tanh activation function)

Solution

Similarly, we will implement the algorithm but using tanh as activation function

$$
\begin{aligned}
Z_1 &= XW_{in} + b_1 \qquad\qquad A_1 = tanh(Z_1) \quad \text{(activation function)} \\
\hat{Y} = Z_2 &= A_1 W_{out} + b_2
\end{aligned}
$$

We obtain partial derivatives wrt to our parameters of interest as follows

$$
\begin{aligned}
\frac{\partial l}{\partial W_{out}} &= \frac{-2}{N} A_1'(Y - Z_2) \\
\frac{\partial l}{\partial W_{in}} &= \frac{-2}{N} X'(Y - Z_2) W_{out}'(\mathbf{1} - A_1^2) \\
\frac{\partial l}{\partial b_1} &= \frac{-2}{N} \mathbf{1}_N'(Y - Z_2) W_{out}'(\mathbf{1} - A_1^2) \\
\frac{\partial l}{\partial b_2} &= \frac{-2}{N} (Y - Z_2)' \mathbf{1}_N
\end{aligned}
$$

The following gradient descent algorithm obtains these values

1. Set $t = 0$ and initialize $W_{out}^{(0)}, W_{in}^{(0)}, b_1^{(0)}, b_2^{(0)}$. Choose ϵ to check for convergence
2. For $t = 1, ..., T$

- $W_{out}^{(t+1)} \leftarrow W_{out}^{(t)} - \eta \frac{\partial l}{\partial W_{out}}$ $\quad \eta$ is fixed learning rate

- $b_2^{(t+1)} \leftarrow b_2^{(t)} - \eta \frac{\partial l}{\partial b_2}$
- $W_{in}^{(t+1)} \leftarrow W_{in}^{(t)} - \eta \frac{\partial l}{\partial W_{in}}$
- $b_1^{(t+1)} \leftarrow b_1^{(t)} - \eta \frac{\partial l}{\partial b_1}$

3. if $(loss(t+1) - loss(t)) \leq \epsilon$ stop;
   else set $t \leftarrow t+1$ and return to step 2

We now Implement this algorithm in R and output the MSEs to choose the efficient number of neurons in the hidden layer. We initialize weights by randomly sampling from the uniform(0,1).

Before we implement this algorithm, we first define a function that computes the MSEs using chosen weights. This function will help us monitor our MSEs when our model is learning

```
NN2.error<-function(X,y,w1,b1,w2,b2){

    Z1<-X%*%W1 + matrix(b1,nrow(X),ncol=ncol(W1),byrow=TRUE)

    ### activation function layer 1

    tanh<-function(x){ (2/(1+exp(-2*x))) -1}

    A1<-matrix(0,ncol=ncol(W1),nrow=nrow(X))

    for (j in 1:ncol(A1)){A1[,j]<-tanh(Z1[,j])}

    ## Score function


    Z2<-A1%*%W2 + rep(b2,nrow(A1))

    mse<-mean((y.train-Z2)^2)

    return(mse)
}
```

Lets now implement the algorithm to learn our Neural model. Choosing our fixed learning rate ($\eta = 0.2$)

```
NN2.tanh<-function(units=3,iter=5000,eta=0.1){

    W1<-matrix(runif(units*ncol(X.train)),ncol=units)
    W2<-matrix(runif(units),ncol=1)

    b1<-runif(units)
    W1.old<-W1
    b1.old<-b1
    b2=1
    W2.old<-W2
    b2.old<-b2

    mse.model2<-numeric(iter)

for (i in 1:iter) {

    Z1<-X.train%*%W1.old + matrix(b1.old,
    nrow(X.train),ncol=ncol(W1.old),byrow=TRUE)

    ### activation function layer 1

    tanh<-function(x){ (2/(1+exp(-2*x))) -1}

    A1<-matrix(0,ncol=ncol(W1.old),nrow=nrow(X.train))

    for (j in 1:ncol(A1)){A1[,j]<-tanh(Z1[,j])}

    Z2<- A1%*%W2.old + rep(b2.old,nrow(A1))

    dldw2<--(2/N)*t(A1)%*%(y.train-Z2)

    dldb2<- crossprod(-(2/N)*(y.train-Z2),rep(1,length(y.train)))

    dldw1<--(2/N)*t(X.train)%*% ((y.train-Z2)%*%t(W2.old)*((1-(A1)*A1)))

    ones<-matrix(1,nrow=nrow(A1),ncol=1)

    dldb1<--(2/N)*t(ones)%*%(((y.train-Z2)%*%t(W2.old))*(1-(A1)*(A1)))


    b2.new<-b2.old-eta*(dldb2)
    W2.new<-W2.old-eta*(dldw2)
    W1.new<-W1.old-eta*(dldw1)
    b1.new<-b1.old-eta*(dldb1)

    mse.new<-NN2.error(X.train,Y.train,W1.new,b1.new,W2.new,b2.new)

    #print(mse.new)

    mse.model2[i]<-mse.new

    W1.old<-W1.new
    W2.old<-W2.new
    b1.old<-b1.new
    b2.old<-b2.new
}
 return(mse.model2)
```

```
}
```

We know observe the plot of the MSEs for number of neurons (3,5,10,15) on our training set. These plots can also inform us on the number of iterations to stop.

```
units<-c(3,5,10,15)
for(i in units) {
model3<-NN2.tanh(i,eta=0.1)
plot(model3,type="l",main=paste("Neurons in Hidden layer",i),ylab="MSE Error")
}
```

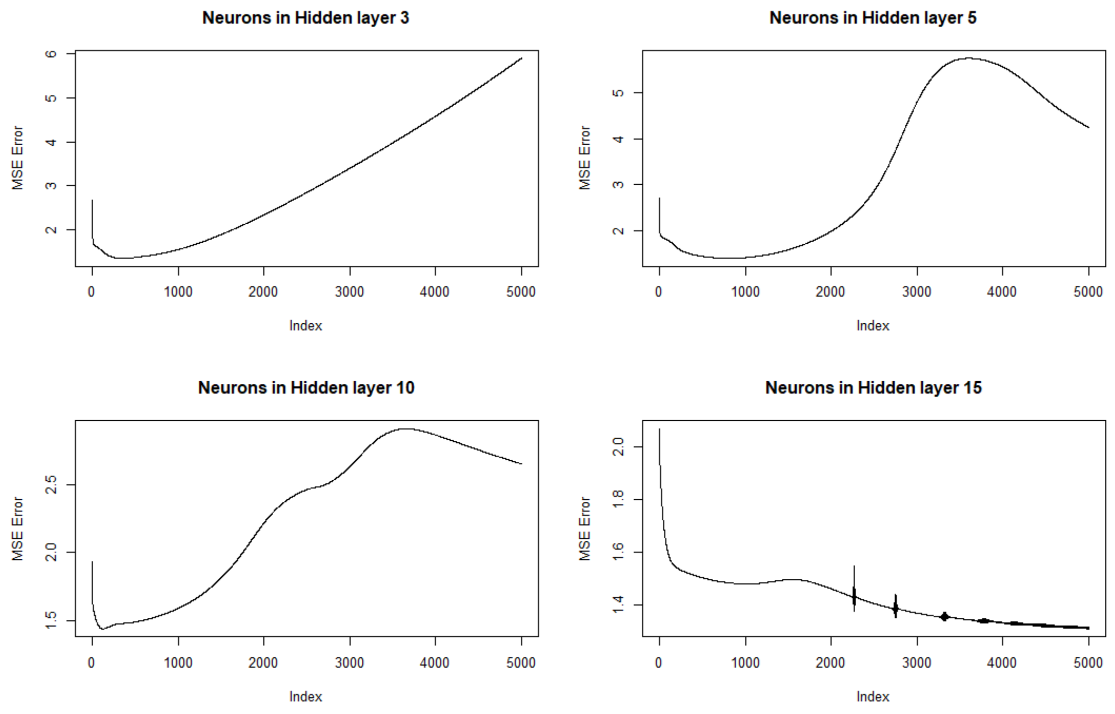


Figure 2: *Plots of several neurons*

We realize from the plots that, we can choose the model with 5 neurons and have an early stopping before 1000 iterations.

Now lets compare the MSEs before and after the model has learnt. Before we do that,we will add a stopping rule to our Neural Network model above. That is, we stop after 500 iterations and the difference in MSE is less than 0.001

```
NN2.tanh<-function(units=3,iter=5000,eta=0.1){

    W1<-matrix(runif(units*ncol(X.train)),ncol=units)
    W2<-matrix(runif(units),ncol=1)

    b1<-runif(units)
    W1.old<-W1
    b1.old<-b1
    b2=1
    W2.old<-W2
    b2.old<-b2

    mse.model2<-numeric(iter)

    for (i in 1:iter) {

        Z1<-X.train%*%W1.old + matrix(b1.old,nrow(X.train),
        ncol=ncol(W1.old),byrow=TRUE)

        ### activation function layer 1

        tanh<-function(x){ (2/(1+exp(-2*x))) -1}

        A1<-matrix(0,ncol=ncol(W1.old),nrow=nrow(X.train))

        for (j in 1:ncol(A1)){A1[,j]<-tanh(Z1[,j])}

        ## Score function

        Z2<- A1%*%W2.old + rep(b2.old,nrow(A1))

        dldw2<--(2/N)*t(A1)%*%(y.train-Z2)

        dldb2<- crossprod(-(2/N)*(y.train-Z2),rep(1,length(y.train)))

        dldw1<--(2/N)*t(X.train)%*% ((y.train-Z2)%*%t(W2.old)*((1-(A1)*A1)))

        ones<-matrix(1,nrow=nrow(A1),ncol=1)

        dldb1<--(2/N)*t(ones)%*%(((y.train-Z2)%*%t(W2.old))*(1-(A1)*(A1)))
        b2.new<-b2.old-eta*(dldb2)
        W2.new<-W2.old-eta*(dldw2)
        W1.new<-W1.old-eta*(dldw1)
        b1.new<-b1.old-eta*(dldb1)

        mse.new<-NN2.error(X.train,Y.train,W1.new,b1.new,W2.new,b2.new)

        #print(mse.new)

        mse.model2[i]<-mse.new

        #print(mse.new)

        if (500 < i & i < 1000 & (mse.new-mse.old)< 0.001){
            break
        }
        W1.old<-W1.new
        W2.old<-W2.new
        b1.old<-b1.new
        b2.old<-b2.new
        mse.old<-mse.new
    }
    return(list(model=mse.model2,W1.new=W1.new,
    b1.new=b1.new,W2.new=W2.new,b2.new=b2.new,W1=W1,
    b1=b1,W2=W2,b2=b2,iteration=i))

}
```

Comparing the MSEs before and after our model has learnt

```
model3<-NN2.tanh(units=5,eta=0.1,iter=2000)

#model3$model
W1.new<-model3$W1.new
b1.new<-model3$b1.new
W2.new<-model3$W2.new
b2.new<-model3$b2.new

W1.old<-model3$W1
b1.old<-model3$b1
W2.old<-model3$W2
b2.old<-model3$b2
iterations<-model3$iteration

### Predicting  ###
```

```
###### before model #####

#model3$model[1000]

mse.before<-NN2.error(X.train,Y.train,W1.old,b1.old,W2.old,b2.old)

mse.before
[1] 3.46479
##### After model ###########

mse.after<-NN2.error(X.train,Y.train,W1.new,b1.new,W2.new,b2.new)

mse.after
[1] 1.482779
```

Conclusion

Using the the sigmoid activation function resulted in lower Mean sqaured errors (MSE) of 1.143464 after training our Neural Network model as compared to an MSE of 1.482779 for the tanh activation function.

Problem 2

We want to solve the L1 SVDD problem in R using the dataset "PHY TRAIN". We will use the first 50 observations of the first class as the training set and the next 10 observations as the test set.

2.1

Solve the L1 SVDD problem using Quadratic Programming. Evaluate the performance of your L1 SVDD by evaluating the accuracy on the training and test sets.

Solution

The L1 SVDD problem is reduced to solving the following dual problem

$$arg\max_{\alpha} \quad \sum_{i=1}^{N} \alpha_i K(x_i, x_i) - \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j)$$

$$\text{such that} \sum_{i=1}^{N} \alpha_i = 1 \quad \text{and } 0 \leq \alpha_i \leq C \quad \forall i$$

where $K(x_i, x_j) = exp(-\frac{||x_i - x_j||}{2\sigma^2})$ is the gaussian kernel function.Since this is a quadratic programming, we can solve it using "quadprog" in R.
After the solution is obtained using "quadprog", we will determine the radius (R) of our sphere

from the center (**a**) using support vectors.

R^2 is obtained as

$$R^2 = \frac{1}{NSv}\sum_{s=1}^{NSv}\left(K(x_s,x_s) - 2\sum_{x_i \in SVs}\alpha_i K(x_i,x_s) + \sum_{x_i,x_j \in SVs}\alpha_i\alpha_j K(x_i,x_j)\right)$$

where SVs is the set of support vectors, NSv is the number of support vectors and α_i is the coordinate value of the dual problem solution corresponding to the support vector x_i.

The following R codes implements the algorithm using "quadprog" in R to obtain solution to our dual problem.

```
library(quadprog)

rbf_kernel <- function(x1,x2,gamma){
    K<-exp(-(1/gamma^2)*t(x1-x2)%*%(x1-x2))
    return(K)
}

l1svdd.train<- function(X,C=Inf, gamma=1.5,esp=1e-10){

    #C=2 ; gamma=1.5; esp=1e-10
    X<-data.matrix(X)
    start_time<-proc.time()
    N<-nrow(X)
    Dm<-matrix(0,N,N)

    for(i in 1:N){
        for(j in 1:N){
             Dm[i,j]<-(rbf_kernel(X[i,],X[j,],gamma))
        }
    }
    dv<-diag(Dm)
    Dm<-Dm+diag(N)*1e-12 # adding a very small number to the diag, some trick
    meq<-1
    Am<-cbind(rep(1,N),diag(N))
    bv<-c(1,rep(0,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<-rbind(matrix(bv,ncol=1),matrix(rep(-C,N),ncol=1))
    }
        alpha_org<-solve.QP(Dm,dv,Am,meq=meq,bvec=bv)$solution
        alphaindx<-which(alpha_org>esp,arr.ind=TRUE)
        alpha<-alpha_org[alphaindx]
        nSV<-length(alphaindx)
        if(nSV==0){
            throw("QP is not able to give a solution for these data points")
        }
        Xv<-X[alphaindx,]

        R.2<-numeric(nSV)
        for (q in 1:nSV) {

        C.R <- numeric(nSV)
        aaK <- numeric(nSV)
        for (i in 1:nSV){
            for (m in 1:nSV){

    aaK[m] <- alpha[m]*alpha[i]*(rbf_kernel(Xv[m,],Xv[i,],gamma))
            }
                C.R[i]<-sum(aaK)

        }
        C.R<-sum(C.R)


            z<-Xv[q,]
            A.R<- rbf_kernel(z,z,gamma)

            B.R<-numeric(nSV)

            for (j in 1:nSV){
               B.R[j]<- alpha[j]*rbf_kernel(Xv[j,],z,gamma)
            }

            B.R<-sum(B.R)

            R.2[q] <- A.R-2*B.R + C.R

        }

        R.2<-mean(R.2)

            time<-proc.time()-start_time

        list(alpha=alpha, R.2=R.2, nSV=nSV, Xv=Xv, gamma=gamma, C=C,time=time)
    }
```

Now we can predict new observation into outlier or normal using the following criterion

We woud classify new observation "z" as an outlier if

$$\left(K(z,z) - 2\sum_{x_i \in SVs} \alpha_i K(x_i,z) + \sum_{x_i,x_j \in SVs} \alpha_i \alpha_j K(x_i,x_j) \right) > R^2$$

The R code to define the prediction function is below

```
L1svdd.pred <- function(X,model){
    X<-data.matrix(X)
    alpha<-model$alpha
    R.2<-model$R.2
    Xv<-model$Xv
    nSV<-model$nSV
    C<-model$C
    gamma<-model$gamma

    aaK <- numeric(nSV)
    result<-numeric(nrow(X))

    C.R <- numeric(nSV)
    aaK <- numeric(nSV)
    for (i in 1:nSV){
        for (m in 1:nSV){

            aaK[m] <- alpha[m]*alpha[i]*(rbf_kernel(Xv[m,],Xv[i,],gamma))
        }
        C.R[i]<-sum(aaK)
    }
    C.R<-sum(C.R)

    for (j in 1:nrow(X)){

        z<-X[j,]
        A.R<- rbf_kernel(z,z,gamma)

        B.R<-numeric(nSV)

        for (q in 1:nSV){
            B.R[q]<- alpha[q]*rbf_kernel(Xv[q,],z,gamma)
        }

        B.R<-sum(B.R)

        R.2.pred <- A.R-2*B.R + C.R

        if (R.2.pred-R.2>0) {
            result[j]<-1
        }
        else{
            result[j]<-0
        }
    }

    return(result)
}
```

We are ready to apply the model to the training and testing data set. Class category label for normal observations is 0 and for predicted outlier is 1.

To train the model, we chose C=1 and $\sigma = \sqrt{0.0005}$

```
### Modeling an Predicting
model_l1svdd <- l1svdd.train(X_train,C=1,gamma=1/sqrt(0.0005),esp=1e-10)

model_l1svdd$time
   user  system elapsed
   0.36    0.00    0.36
###Training
l1svdd_result<-L1svdd.pred(X_train,model_l1svdd)
table(predict=as.factor(l1svdd_result),truth=as.factor(Y_train))
       truth
predict  0
      0 36
      1 14

### testing
l1svdd_result<-L1svdd.pred(X_test,model_l1svdd)
table(predict=as.factor(l1svdd_result),truth=as.factor(Y_test))
        truth
predict 0
      0 7
      1 3
```

Our model has accuracy rate of 72% on the training set and 70% on the test set.

2.2

Solve the L1 SVDD problem using Gradient Descent algorithm. Eval- uate the performance of your L1 SVDD by evaluating the accuracy on the training and test sets.

Solution

To solve the L1 SVDD problem using Gradient Descent, we formulate the problem as

$$\min_{R,\mathbf{a}} \quad \frac{\lambda}{2}R^2 + \frac{1}{2}\sum_{i=1}^{N} max(0, ||x_i - \mathbf{a}||^2 - R^2)$$

where $\lambda = 1/C$, $\mathbf{a}$ is the center and R is the radius.

The Sub-gradient descent algorithm to solve the problem is below

1. Set $t = 0$ and initialize $R^{(0)}$ and $\mathbf{a}^{(0)}$. Choose ϵ to check for convergence
2. For $t = 1, ..., T$
   Cycle through the whole training set
   - if $||x_i - a^{(t)}||^2 - R^{2(t)} \leq 0$
     $R^{(t+1)} \leftarrow \lambda R^{(t)}$
     $\mathbf{a}^{(t+1)} \leftarrow 0$
   - else
     $R^{(t+1)} \leftarrow R^{(t)} - \eta R^{(t)}(\lambda - 1)$ (η is the learning rate)
     $\mathbf{a}^{(t+1)} \leftarrow \mathbf{a}^{(t)} - \eta(\mathbf{a}^{(t)} - x_i)$
3. if $|R^{(t+1)} - R^{(t)}| \leq \epsilon$ & $mean(|\mathbf{a}^{(t+1)} - \mathbf{a}^{(t)}|) \leq \epsilon$ stop;
   else set $t \leftarrow t + 1$ and return to step 2

The following R codes implement this algorithm

```
l1svdd_grad_descent<- function(X,R0,a0,C,eta,eps=1e-8,iter=1000){

    start_time<-proc.time()

    R_old<-R0
    a_old<-matrix(a0,ncol=1)

    X<-data.matrix(X)
    N<-nrow(X)
    t=0
    lambda<-1/C



for (i in 1:iter){
        t=t+1
        #dist<-numeric(nrow(X))
        for (j in 1:nrow(X)){
            xj<-X[j,]

            dist<-crossprod(xj-a_old)

            if((dist-R_old^2)<=0){

                R_new<-(lambda)*R_old
                a_new<- rep(0,ncol(X))
            }else{
                R_new<-R_old*(1-eta*(lambda-1))
                dist1<-(a_old-xj)
                a_new<-a_old-eta*dist1
            }
            #print(abs(R_old-R_new))

        }

            if (abs(R_old-R_new)<eps & mean(abs(a_old-a_new))<eps){

                break
            }

        if (t==10){

            print(mean(abs(a_old-a_new)))
        }
        R_old<-R_new
        a_old<-a_new

    }

    time<-proc.time()-start_time

    return(list("R"=R_new,"a"=a_new,"iteration"=i,"time"=time))
}

###### predict #######

svdd.pred<-function(X,R,a){
    X<-data.matrix(X)
    result<-numeric(nrow(X))
    for (i in 1:nrow(X)){

        xi<-X[i,]
        if (crossprod(xi-a)<R){
            result[i]<-0
        }else{
            result[i]<-1
        }
    }
    return(result)
}
```

We would classify new observation z as normal or outlier using the following criterion

An observation z is an outlier if

$$||z - \mathbf{a}||^2 - R^2 > 0$$

The codes below defines the prediction function

```
svdd.pred<-function(X,R,a){
    X<-data.matrix(X)
    result<-numeric(nrow(X))
     for (i in 1:nrow(X)){

         xi<-X[i,]
         if (crossprod(xi-a)<R){
             result[i]<-0
         }else{
             result[i]<-1
         }
     }
    return(result)
}
```

We now apply our model to the train and test data set. We label class category for normal observation as 0 and outlier as 1

```
###Training
R0<-100
a0<-rep(0,ncol(X_train))

grad_l1svdd<-l1svdd_grad_descent(X_train,R0,a0,C=1,eta=0.0001,iter=100000)

R<-grad_l1svdd$R
a<-grad_l1svdd$a
grad_l1svdd$iteration

grad_l1svdd$time



###Training

grad.descent_result<-svdd.pred(X_train,R,a)
table(predict=as.factor(grad.descent_result),truth=as.factor(Y_train))
       truth
predict  0
      0 28
      1 22
### testing

grad.descent_result<-svdd.pred(X_test,R,a)
table(predict=as.factor(grad.descent_result),truth=as.factor(Y_test))
       truth
```

```
predict 0
      0 5
      1 5
```

Our model has accuracy rate of 56% on the training set and 50% on the test set.

2.3

Solve the L1 SVDD problem using Stochastic Gradient Descent algorithm. Eval- uate the performance of your L1 SVDD by evaluating the accuracy on the training and test sets.

Solution

To solve the L1 SVDD problem using Stochastic Gradient Descent, we formulate the problem as

$$\max_{R,\mathbf{a}} \quad \frac{\lambda}{2}R^2 + \sum_{i=1}^{N} max(0, ||x_i - \mathbf{a}||^2 - R^2)$$

where $\lambda = 1/C$, $\mathbf{a}$ is the center and R is the radius.

The Sub-stochastic gradient descent algorithm to solve the problem is below

```
1. Set $t = 0$ and initialize $R^{(0)}$ and $\mathbf{a}^{(0)}$. Choose $\epsilon$ to check for convergence
2. For $t = 1, ..., T$
   randomly select one observation $(x_i)$ from the training set
     • if $||x_i - a^{(t)}||^2 - R^{2(t)} \leq 0$
       $R^{(t+1)} \leftarrow \lambda R^{(t)}$
       $\mathbf{a}^{(t+1)} \leftarrow 0$
     • else
       $R^{(t+1)} \leftarrow R^{(t)} - \eta R^{(t)}(\lambda - 1)$ ($\eta$ is the learning rate)
       $\mathbf{a}^{(t+1)} \leftarrow \mathbf{a}^{(t)} - \eta(\mathbf{a}^{(t)} - x_i)$
3. if $|R^{(t+1)} - R^{(t)}| \leq \epsilon$ & $mean(|\mathbf{a}^{(t+1)} - \mathbf{a}^{(t)}|) \leq \epsilon$ stop;
   else set $t \leftarrow t + 1$ and return to step 2
```

The following R codes implement this algorithm

```
l1svdd_stoc_descent<- function(X,R0,a0,C,eta,eps=1e-8,iter=1000){

    start_time<-proc.time()

    R_old<-R0
    a_old<-matrix(a0,ncol=1)

    X<-data.matrix(X)
    N<-nrow(X)
    t=0
    lambda<-1/C

    for (i in 1:iter){
        t=t+1

        j<-sample(1:nrow(X),1)
            xj<-X[j,]
            dist<-crossprod(xj-a_old)

        if((dist-R_old^2)<=0){

            R_new<-(lambda)*R_old
            a_new<- rep(0,ncol(X))
        }else{
            R_new<-R_old*(1-eta*(lambda-1))
            dist1<-(a_old-xj)
            a_new<-a_old-eta*dist1
        }
            #print(abs(R_old-R_new))

        if (abs(R_old-R_new)<eps & mean(abs(a_old-a_new))<eps){

            break
        }

        if (t==10000){

            print(mean(abs(a_old-a_new)))
        }
        R_old<-R_new
        a_old<-a_new

    }
    time<-proc.time()-start_time

    return(list("R"=R_new,"a"=a_new,"iteration"=i,"time"=time))
}
```

We would classify new observation z as normal or outlier using the following criterion

An observation z is an outlier if

$$||z - \mathbf{a}||^2 - R^2 > 0$$

We now apply our model to the train and test data set. We label class category for normal observation as 0 and outlier as 1

```
###Training
set.seed(12345)
R0<-10
a0<-rep(0,ncol(X_train))

stoc_l1svdd<-l1svdd_stoc_descent(X_train,R0,a0,C=1,eta=0.0001,eps=1e-8,iter=50000)

R<-stoc_l1svdd$R
a<-stoc_l1svdd$a
stoc_l1svdd$iteration
stoc_l1svdd$time



###Training set ###
stoc.descent_result<-svdd.pred(X_train,R,a)
table(predict=as.factor(stoc.descent_result),truth=as.factor(Y_train))
       truth
predict  0
      0 28
      1 22
### testing set ###
stoc.descent_result<-svdd.pred(X_test,R,a)
table(predict=as.factor(stoc.descent_result),truth=as.factor(Y_test))
      truth
predict 0
      0 5
      1 5
```

Our model has accuracy rate of 56% on the training set and 50% on the test set.

2.3

Compare the results from (1), (2), and (3) in terms of speed, number of iterations before convergence and accuracy.

Solution

Measure	L1-SVDD(quadprog)	L1-SVDD(GD)	L1-SVDD(SGD)
Time	0.36	0	0
Accuracy(Train)	0.72	0.56	0.56
Accuracy(Test)	0.70	0.50	0.50
Iteration	-	1	1

Conclusion
Overall, the model obtained from quadprog has the highest accuracy.

Problem 3

We want to solve the L2 SVDD problem in R using the dataset "PHY TRAIN". We will use the first 50 observations of the first class as the training set and the next 10 observations as the test set. We will use the Gaussian kernel for the choice of our kernel.

3.1

Solve the L2 SVDD problem using Quadratic Programming. Evaluate the performance of your L2 SVDD by evaluating the accuracy on the training and test sets.

Solution

The L2 SVDD problem is reduced to solving the following dual problem

$$\begin{aligned}
arg \max_{\alpha} \quad & \sum_{i=1}^{N} \alpha_i K(x_i, x_i) - \sum_{i,j} \alpha_i \alpha_j \left(K(x_i, x_j) + \frac{1}{2C}\delta_{i,j} \right) \\
\text{subject to} & \sum_{i=1}^{N} \alpha_i = 1 \quad \text{and } 0 \leq \alpha_i \leq \infty \quad \forall i \\
\text{where } \delta_{ij} = & \begin{cases} 1 & if \quad i \neq j \\ 0 & if \quad otherwise \end{cases}
\end{aligned}$$

where $K(x_i, x_j) = exp(-\frac{||x_i - x_j||}{\sigma^2})$ is the gaussian kernel function.Since this is a quadratic programming, we can solve it using "quadprog" in R.
After the solution is obtained using "quadprog", we will determine the radius (R) of our sphere from the center ($\mathbf{a}$) using support vectors.

R^2 is obtained as

$$R^2 = \frac{1}{NSv} \sum_{s=1}^{NSv} \left(K(x_s, x_s) - 2 \sum_{x_i \in SVs} \alpha_i K(x_i, x_s) + \sum_{x_i, x_j \in SVs} \alpha_i \alpha_j K(x_i, x_j) \right)$$

where SVs is the set of support vectors, NSv is the number of support vectors and α_i is the coordinate value of the dual problem solution corresponding to the support vector x_i.

The following R codes implements the algorithm using "quadprog" in R to obtain solution to our dual problem.

```
l2svdd.train <- function(X,C=Inf,gamma=1.5,esp=1e-10){

    #C=2 ; gamma=1.5; esp=1e-10
    X<-data.matrix(X)
    start_time<-proc.time()
    N<-nrow(X)
```

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

for(i in 1:N){
    for(j in 1:N){
        if (i==j){
            Dm[i,j]<-(rbf_kernel(X[i,],X[j,],gamma)+1/2*C)
        }
        else{
            Dm[i,j]<-rbf_kernel(X[i,],X[j,],gamma)
        }
    }
}

dv<-diag(Dm)
Dm<-2*Dm #+diag(N)*1e-7 # adding a very small number to the diag, some trick
meq<-1
Am<-Am<-cbind(rep(1,N),diag(N))
bv<-c(1,rep(0,N))  # the 1 is for the sum(alpha)==0, others for each alpha_i >= 0
alpha_org<-solve.QP(Dm,dv,Am,meq=meq,bvec=bv)$solution
alphaindx<-which(alpha_org>esp,arr.ind=TRUE)
alpha<-alpha_org[alphaindx]
nSV<-length(alphaindx)
if(nSV==0){
    throw("QP is not able to give a solution for these data points")
}
Xv<-X[alphaindx,]

R.2<-numeric(nSV)

for (p in 1:nSV){

z<-Xv[p,]
A.R<- rbf_kernel(z,z,gamma)

B.R<-numeric(nSV)

for (j in 1:nSV){
    B.R[j]<- alpha[j]*rbf_kernel(Xv[j,],z,gamma)
}

B.R<-sum(B.R)

C.R <- numeric(nSV)
aaK <- numeric(nSV)
for (i in 1:nSV){
    for (m in 1:nSV){
            aaK[m] <- alpha[m]*alpha[i]*(rbf_kernel(Xv[m,],Xv[i,],gamma))
        }
        C.R[i]<-sum(aaK)

    }
    C.R<-sum(C.R)


    R.2[p] <- A.R-2*B.R + C.R

    }
    R.2<-mean(R.2)

    time<-proc.time()-start_time

    list(alpha=alpha, R.2=R.2, nSV=nSV, Xv=Xv, gamma=gamma, C=C,time=time)
}
```

Now we can predict new observation into outlier or normal using the following criterion

We woud classify new observation ”z” as an outlier if

$$\left(K(z,z) - 2\sum_{x_i \in SVs} \alpha_i K(x_i,z) + \sum_{x_i,x_j \in SVs} \alpha_i \alpha_j K(x_i,x_j) \right) > R^2$$

We are now ready to apply our model to the training and testing data sets.

```
model_l2svdd <- l2svdd.train(X_train,C=1,gamma=1/sqrt(0.00005),esp=1e-10)

model_l2svdd$nSV

model_l2svdd$time
user  system elapsed
 2.20    0.01    2.22

###Training
l2svdd_result<-L1svdd.pred(X_train,model_l2svdd)
table(predict=as.factor(l2svdd_result),truth=as.factor(Y_train))

       truth
predict  0
      0 34
      1 16
### testing
l2svdd_result<-L1svdd.pred(X_test,model_l2svdd)
table(predict=as.factor(l2svdd_result),truth=as.factor(Y_test))
```

```
        truth
predict 0
      0 7
      1 3
```

Our model has accuracy rate of 68% on the training set and 70% on the test set.

3.2

Solve the L2 SVDD problem using Gradient Descent algorithm. Eval uate the performance of your L2 SVDD by evaluating the accuracy on the training and test sets.

Solution

To solve the L2 SVDD problem using Gradient Descent, we formulate the problem as

$$\max_{R,\mathbf{a}} \quad \frac{\lambda}{2}R^2 + \sum_{i=1}^{N} max(0, ||x_i - \mathbf{a}||^2 - R^2)^2$$

where $\lambda = 1/C$, $\mathbf{a}$ is the center and R is the radius.

The Sub-gradient descent algorithm to solve the problem is below

```
1. Set $t = 0$ and initialize $R^{(0)}$ and $\mathbf{a}^{(0)}$. Choose $\epsilon$ to check for convergence
2. For $t = 1, ..., T$
   Cycle through the whole training set
     • if $||x_i - a^{(t)}||^2 - R^{2(t)} \leq 0$
       $R^{(t+1)} \leftarrow \lambda R^{(t)}$
       $\mathbf{a}^{(t+1)} \leftarrow 0$
     • else
       $R^{(t+1)} \leftarrow R^{(t)} - \eta R^{(t)}(\lambda - 2(||x_i - a^{(t)}||^2 - R^{2(t)}))$ ($\eta$ is the learning rate)
       $\mathbf{a}^{(t+1)} \leftarrow \mathbf{a}^{(t)} - \eta 2(\mathbf{a}^{(t)} - x_i)(||x_i - a^{(t)}||^2 - R^{2(t)})$
3. if $|R^{(t+1)} - R^{(t)}| \leq \epsilon$ & $mean(|\mathbf{a}^{(t+1)} - \mathbf{a}^{(t)}|) \leq \epsilon$ stop;
   else set $t \leftarrow t + 1$ and return to step 2
```

The following R codes implement this algorithm

```
l2svdd_grad_descent<- function(X,R0,a0,C,eta,eps=1e-8,iter=1000){

    start_time<-proc.time()

    R_old<-R0
    a_old<-matrix(a0,ncol=1)

    X<-data.matrix(X)
    N<-nrow(X)
    t=0
    lambda<-1/C

    for (i in 1:iter){
        t=t+1
        #dist<-numeric(nrow(X))
        for (j in 1:nrow(X)){
            xj<-X[j,]
            dist<-crossprod(xj-a_old)

            if((dist-R_old^2)<=0){
                R_new<-(lambda)*R_old

               # R_new<-R_old
                a_new<- rep(0,ncol(X))
            } else {
                R_new<-R_old*(1-eta*(lambda-2*(dist-R_old^2)))

                #R_new<-R_old*(1-eta*(1-2*C*(dist-R_old^2)))

                dist1<-(xj-a_old)*as.vector((crossprod(xj-a_old)-R_old^2))

                #a_new<-a_old+2*eta*C*dist1

                a_new<-a_old+2*eta*dist1
            }

        }
        if (abs(R_old-R_new)<eps & mean(abs(a_old-a_new))<eps){

            break
        }

        if (t==10){

            print(mean(abs(a_old-a_new)))
        }
        R_old<-R_new
        a_old<-a_new

    }

    time<-proc.time()-start_time

    return(list("R"=R_new,"a"=a_new,"iteration"=i,"time"=time))
}
```

We would classify new observation z as normal or outlier using the following criterion

An observation z is an outlier if

$$||z - \mathbf{a}||^2 - R^2 > 0$$

We now apply our model to the train and test data set. We label class category for normal observation as 0 and outlier as 1

```
R0<-1
a0<-rep(0,ncol(X_train))

grad_l2svdd<-l2svdd_grad_descent(X_train,R0,a0,C=1,eta=0.001,iter=10000)

R<-grad_l1svdd$R
a<-grad_l1svdd$a
grad_l1svdd$iteration
grad_l1svdd$time

###### predict #######

###Training set ##
grad.descent_result<-svdd.pred(X_train,R,a)
table(predict=as.factor(grad.descent_result),truth=as.factor(Y_train))
      truth
predict  0
      0 28
      1 22

### testing ##
grad.descent_result<-svdd.pred(X_test,R,a)

table(predict=as.factor(grad.descent_result),truth=as.factor(Y_test))
       truth
predict 0
      0 5
      1 5
```

Our model has accuracy rate of 56% on the training set and 50% on the test set.

3.3

Solve the L2 SVDD problem using Stochastic Gradient Descent algo- rithm. Evaluate the performance of your L2 SVDD by evaluating the accuracy on the training and test sets.

Solution

The Sub-Stochastic gradient descent algorithm to solve the problem is below

1. Set $t = 0$ and initialize $R^{(0)}$ and $\mathbf{a}^{(0)}$. Choose ϵ to check for convergence

2. For $t = 1, ..., T$
   Randomly choose one observation from the training set
   - if $||x_i - a^{(t)}||^2 - R^{2(t)} \leq 0$
     $R^{(t+1)} \leftarrow \lambda R^{(t)}$
     $\mathbf{a}^{(t+1)} \leftarrow 0$
   - else
     $R^{(t+1)} \leftarrow R^{(t)} - \eta R^{(t)}(\lambda - 2(||x_i - a^{(t)}||^2 - R^{2(t)}))$ (η is the learning rate)
     $\mathbf{a}^{(t+1)} \leftarrow \mathbf{a}^{(t)} - 2\eta(\mathbf{a}^{(t)} - x_i)(||x_i - a^{(t)}||^2 - R^{2(t)})$
3. if $|R^{(t+1)} - R^{(t)}| \leq \epsilon$ & $mean(|\mathbf{a}^{(t+1)} - \mathbf{a}^{(t)}|) \leq \epsilon$ stop;
   else set $t \leftarrow t + 1$ and return to step 2

The following R codes implement this algorithm

```
l2svdd_stoc_descent<- function(X,R0,a0,C,eta,eps=1e-8,iter=1000){

    start_time<-proc.time()

    R_old<-R0
    a_old<-matrix(a0,ncol=1)

    X<-data.matrix(X)
    N<-nrow(X)
    t=0
    lambda<-1/C

    for (i in 1:iter){
        t=t+1
        j<-sample(1:nrow(X),1)
        xj<-X[j,]

        dist<-crossprod(xj-a_old)

        if((dist-R_old^2)<=0){
           #R_new<-(lambda)*R_old

            R_new<-R_old

            a_new<- rep(0,ncol(X))
        } else {
            #R_new<-R_old*(1-eta*(lambda-2*(dist-R_old^2)))

            R_new<-R_old*(1-eta*(1-2*C*(dist-R_old^2)))

            dist1<-(xj-a_old)*as.vector((crossprod(xj-a_old)-R_old^2))

            #<-a_old+2*eta*C*dist1
            #a_new<-a_old+2*eta*dist1
        }


        if (abs(R_old-R_new)<eps & mean(abs(a_old-a_new))<eps){

            break
        }

        #print(mean(abs(a_old-a_new)))
        #print(i)
        if (t==100){
            print((abs(R_old-R_new)))

            #break
        }
        R_old<-R_new
        a_old<-a_new
    }
    time<-proc.time()-start_time

    return(list("R"=R_new,"a"=a_new,"iteration"=i,"time"=time))
}
```

We would classify new observation z as normal or outlier using the following criterion

An observation z is an outlier if

$$||z - \mathbf{a}||^2 - R^2 > 0$$

We now apply our model to the train and test data set. We label class category for normal observation as 0 and outlier as 1

```
###Training
set.seed(12345)
R0<-1
a0<-rep(0,ncol(X_train))

stoc_l2svdd<-l2svdd_stoc_descent(X_train,R0,a0,C=1,eta=0.01,iter=100000)

R<-stoc_l2svdd$R
a<-stoc_l2svdd$a
stoc_l2svdd$iteration

stoc_l2svdd$time

user  system elapsed
 0       0       0
```

```
###Training
stoc.descent_result<-svdd.pred(X_train,R,a)
table(predict=as.factor(stoc.descent_result),truth=as.factor(Y_train))
       truth
predict  0
      0 33
      1 17


### testing ###
stoc.descent_result<-svdd.pred(X_test,R,a)
table(predict=as.factor(stoc.descent_result),truth=as.factor(Y_test))
       truth
predict 0
      0 7
      1 3
```

Our model has accuracy rate of 66% on the training set and 70% on the test set.

3.4

Compare the results from (1), (2), and (3) in terms of speed, number of iterations before convergence and accuracy.

Solution

Measure	L2-SVDD(quadprog)	L2-SVDD(GD)	L2-SVDD(SGD)
Time	0.36	0	0
Accuracy(Train)	0.68	0.56	0.66
Accuracy(Test)	0.70	0.50	0.70
Iteration	-	1	3

Conclusion

The model fitted with gradient descent has the lowest accuracy rate.

Problem 4

We want to solve the Least Squares SVDD (LS SVDD) problem in R using the dataset ”PHY TRAIN”. We will use the first 50 observations of the first class as the training set and the next 10 observations as the test set. We will use the Gaussian kernel for the choice of our kernel.

4.1

Solve the LS SVDD problem using Quadratic Programming. Evaluate the performance of your LS SVDD by evaluating the accuracy on the training and test sets.

Solution

The LS SVDD problem is reduced to solving the following dual problem

$$
\begin{aligned}
arg\max_{\alpha} \quad & \sum_{i=1}^{N} \alpha_i K(x_i, x_i) - \sum_{i,j} \alpha_i \alpha_j \left(K(x_i, x_j) + \frac{1}{2C}\delta_{i,j} \right) \\
\text{subject to} & \sum_{i=1}^{N} \alpha_i = 1 \\
\text{where } \delta_{ij} = & \begin{cases} 1 & if \quad i \neq j \\ 0 & if \quad otherwise \end{cases}
\end{aligned}
$$

where $K(x_i, x_j) = exp(-\frac{||x_i - x_j||}{2\sigma^2})$ is the gaussian kernel function. The solution to this problem can be obtained analytically as

$$
\alpha^* = \frac{1}{2}\mathbf{H}^{-1}\left(\mathbf{k} + \frac{2 - \mathbf{e}'\mathbf{H}\mathbf{k}}{\mathbf{e}'\mathbf{H}\mathbf{e}}\mathbf{e} \right)
$$

where $\mathbf{H} = \mathbf{K} + \frac{1}{2}I_N$ where $\mathbf{K}$ is the Gram matrix with entries $K_{i,j} = K(x_i, x_j)$, $\mathbf{k}$ is a vector with entries $K_j = K(x_i, x_j), \quad j = 1, 2, 3, ..., N$ and $\mathbf{e} = (1, 1, ..., 1)'$

After the solution is obtained, we will determine the radius (R) of our sphere from the center ($\mathbf{a}$) as.

R^2 is obtained as

$$
R^2 = \frac{1}{N}\sum_{s=1}^{N}\left(K(x_s, x_s) - 2\sum_{i} \alpha_i K(x_i, x_s) + \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) \right)
$$

The following R codes is used to obtain the solution to the Dual problem

```
LS.svddtrain <- function(X,C=C,gamma=1.5,esp=1e-10){

    X<-data.matrix(X)
    N<-nrow(X)
    K<-matrix(0,N,N)

    start_time<-proc.time()
    for(i in 1:N){
        for(j in 1:N){
            K[i,j]<-(rbf_kernel(X[i,],X[j,],gamma))
        }
    }
    H<- K+diag(N)*(1/2*C)
    k<-diag(K)
    e<-matrix(rep(1,N),ncol=1)
alpha<-(1/2)*solve(H)%*%
(k+ as.vector(((2-t(e)%*%solve(H)%*%k)/(t(e)%*%solve(H)%*%e)))*e)

    R.2<-numeric(N)
    for(p in 1:N){

        z<-X[p,]
        A.R<- rbf_kernel(z,z,gamma)

        B.R<-numeric(N)

        for (j in 1:N){
            B.R[j]<- alpha[j]*rbf_kernel(X[j,],z,gamma)
        }

        B.R<-sum(B.R)

    C.R <- numeric(N)
    aaK <- numeric(N)
    for (i in 1:N){
        for (m in 1:N){

            aaK[m] <- alpha[m]*alpha[i]*(rbf_kernel(X[m,],X[i,],gamma))
        }
        C.R[i]<-sum(aaK)
    }
    C.R<-sum(C.R)

    R.2[p] <- A.R-2*B.R + C.R
    }

    R.2<-mean(R.2)

    time<-proc.time()-start_time

    list(alpha=alpha,X=X,R.2=R.2,C=C,gamma=gamma,time=time)
}
```

We woud classify new observation "z" as an outlier if

$$\left(K(z,z) - 2\sum_i \alpha_i K(x_i,z) + \sum_{i,j} \alpha_i \alpha_j K(x_i,x_j)\right) > R^2$$

We now apply our model to the training and testing set.We choose $C = 1$ and $\sigma = 1/\sqrt{0.0005}$

```
model.LSsvdd <- LS.svddtrain(X_train,C=1,gamma=1/sqrt(0.0005),esp=1e-10)

###Training
LSsvdd_result<-LS.svdd.pred(X_train,model.LSsvdd)
table(predict=as.factor(LSsvdd_result),truth=as.factor(Y_train))
       truth
predict  0
      0 31
      1 19

### testing
LSsvdd_result<-LS.svdd.pred(X_test,model.LSsvdd)
table(predict=as.factor(LSsvdd_result),truth=as.factor(Y_test))

       truth
predict 0
      0 7
      1 3
```

Our model has 62% accuracy on training set and 70% accuracy on the test set.

4.2

Solve the LS SVDD problem using Gradient Descent algorithm. Evaluate the performance of your LS SVDD by evaluating the accuracy on the training and test sets.

Solution

To solve the LS SVDD problem using Gradient Descent, we formulate the problem as

$$\min_{R,\mathbf{a}} \quad \frac{\lambda}{2}R^2 + \frac{1}{2}\sum_{i=1}^{N}(||x_i - \mathbf{a}||^2 - R^2)^2$$

where $\lambda = 1/C$, $\mathbf{a}$ is the center and R is the radius.

The gradient descent algorithm to solve the problem is below

1. Set $t = 0$ and initialize $R^{(0)}$ and $\mathbf{a}^{(0)}$. Choose ϵ to check for convergence
2. For $t = 1, ..., T$

- $R^{(t+1)} \leftarrow R^{(t)} - \eta(\lambda - 2\sum_i (||x_i - \mathbf{a}^{(t)}||^2 - R^2)$ where η is fixed learning rate
- $\mathbf{a}^{(t+1)} \leftarrow \mathbf{a}^{(t)} + 2\eta \sum_i (x_i - \mathbf{a}^{(t)})(||x_i - \mathbf{a}^{(t)}||^2 - R^2)$

3. if $|R^{(t+1)} - R^{(t)}| \leq \epsilon$ & $mean(|\mathbf{a}^{(t+1)} - \mathbf{a}^{(t)}|) \leq \epsilon$ stop;
   else set $t \leftarrow t + 1$ and return to step 2

We now implement the algorithm using the following R codes

```
LSsvdd_grad_descent<- function(X,R0,a0,C,eta,eps=1e-8,iter=1000){

    start_time<-proc.time()

    R_old<-R0
    a_old<-matrix(a0,ncol=1)

    X<-data.matrix(X)
    N<-nrow(X)
    t=0
    lambda<-1/C

    for (i in 1:iter){
        t=t+1
        dist<-numeric(nrow(X))
        for (j in 1:nrow(X)){
            xj<-X[j,]
            dist[j]<- (crossprod(xj-a_old)-R_old^2)
        }
        dist<-sum(dist)

            #R_new<-R_old*(1-eta*(lambda- 2*(dist)))

            R_new<-R_old*(1-eta*(1- 2*C*(dist)))

            dist1<-matrix(0,ncol=ncol(X),nrow=nrow(X))
            for (q in 1:nrow(X)){
                xq<-X[q,]
                dist1[q,]<-(xj-a_old)*as.vector((crossprod(xj-a_old)-R_old^2))
            }

            dist1<-apply(dist1,2,sum)

            #a_new<-a_old+ 2*eta*dist1
            a_new<-a_old+ 2*eta*C*dist1

        #print(abs(R_old-R_new))
        if (abs(R_old-R_new)<eps && mean(abs(a_old-a_new))<eps){

            break
        }
        #print(i)
        if (t==5000){

            print(mean(abs(a_old-a_new)))

            print(mean(abs(R_old-R_new)))
        }
        R_old<-R_new
        a_old<-a_new

    }

    time<-proc.time()-start_time

    return(list("R"=R_new,"a"=a_new,"iteration"=i,"time"=time))
}
```

Applying our model to the training and testing data sets. We would classify new observation z as normal or outlier using the following criterion

An observation z is an outlier if

$$||z - \mathbf{a}||^2 - R^2 > 0$$

```
R0<-10
a0<-rep(0,ncol(X_train))

grad_LSsvdd<-LSsvdd_grad_descent(X_train,R0,a0,C=0.000001,eta=0.000000001,iter=50000)

R<-grad_LSsvdd$R
a<-grad_LSsvdd$a
grad_LSsvdd$iteration

grad_LSsvdd$time




###Training

grad.descent_result<-svdd.pred(X_train,R,a)
table(predict=as.factor(grad.descent_result),truth=as.factor(Y_train))
       truth
predict  0
      1 50

### testing
grad.descent_result<-svdd.pred(X_test,R,a)
```

```
table(predict=as.factor(grad.descent_result),truth=as.factor(Y_test))
       truth
predict  0
      1 10
```

The model predicted all training and testing data set as outliers.

4.3

Solve the LS SVDD problem using Stochastic Gradient Descent algo- rithm. Evaluate the performance of your LS SVDD by evaluating the accuracy on the training and test sets.

Solution

To solve the LS SVDD problem using Stochastic Gradient Descent, we formulate the problem as

$$\min_{R,\mathbf{a}} \quad \frac{\lambda}{2}R^2 + \frac{1}{2}\sum_{i=1}^{N}(||x_i - \mathbf{a}||^2 - R^2)^2$$

where $\lambda = 1/C$, $\mathbf{a}$ is the center and R is the radius.

The stochastic gradient descent algorithm to solve the problem is below

1. Set $t = 0$ and initialize $R^{(0)}$ and $\mathbf{a}^{(0)}$. Choose ϵ to check for convergence
2. For $t = 1, ..., T$
   Randomly choose one observation from the training set
   - $R^{(t+1)} \leftarrow R^{(t)} - \eta(\lambda - 2(||x_i - \mathbf{a}^{(t)}||^2 - R^2)$ where η is fixed learning rate
   - $\mathbf{a}^{(t+1)} \leftarrow \mathbf{a}^{(t)} + 2\eta(x_i - \mathbf{a}^{(t)})(||x_i - \mathbf{a}^{(t)}||^2 - R^2)$
3. if $|R^{(t+1)} - R^{(t)}| \leq \epsilon$ & $mean(|\mathbf{a}^{(t+1)} - \mathbf{a}^{(t)}|) \leq \epsilon$ stop;
   else set $t \leftarrow t + 1$ and return to step 2

Implementing the algorithm in R

```
LSsvdd_stoc_descent<- function(X,R0,a0,C,eta,eps=1e-6,iter=1000){

    start_time<-proc.time()

    R_old<-R0
    a_old<-matrix(a0,ncol=1)

    X<-data.matrix(X)
    N<-nrow(X)
    t=0
    lambda<-1/C
    for (i in 1:iter){
        t=t+1

            j<-sample(1:nrow(X),1)
            xj<-X[j,]

            dist<-crossprod(xj-a_old)

            #R_new<-R_old*(1-eta*(lambda-2*(dist-R_old^2)))
                R_new<-R_old*(1-eta*(1-2*C*(dist-R_old^2)))

                dist1<-(xj-a_old)*as.vector((crossprod(xj-a_old)-R_old^2))

                #a_new<-a_old+2*eta*dist1
                a_new<-a_old+2*eta*C*dist1


            if (abs(R_old-R_new)<eps & mean(abs(a_old-a_new))<eps){

                break
            }
        #print(i)
                print(mean(abs(R_old-R_new)))
                print(mean(abs(a_old-a_new)))
        if (t==10000){

            print(mean(abs(a_old-a_new)))

            print(mean(abs(R_old-R_new)))
        }

        R_old<-R_new
        a_old<-a_new
    }

    time<-proc.time()-start_time

    return(list("R"=R_new,"a"=a_new,"iteration"=i,"time"=time))
}
```

We would classify new observation z as normal or outlier using the following criterion

An observation z is an outlier if

$$||z - \mathbf{a}||^2 - R^2 > 0$$

```
###Training ####

set.seed(1234)
```

```
R0<-10
a0<-rep(0,ncol(X_train))

stoc_LSsvdd<-LSsvdd_stoc_descent(X_train,R0,a0,C=0.0001,eta=0.0000001,iter=200000)

R<-stoc_LSsvdd$R
a<-stoc_LSsvdd$a
stoc_LSsvdd$iteration

grad_l1svdd$time
  user  system elapsed
  0.01    0.00    0.02

###Training
grad.descent_result<-svdd.pred(X_train,R,a)
table(predict=as.factor(grad.descent_result),truth=as.factor(Y_train))
      truth
predict  0
      0  7
      1 43

### testing
grad.descent_result<-svdd.pred(X_test,R,a)
table(predict=as.factor(grad.descent_result),truth=as.factor(Y_test))
       truth
predict 0
      0 1
      1 9
```

Our model has 14% accuracy on training set and 10% accuracy on the test set.

4.4

Compare the results from (1), (2), and (3) in terms of speed, number of iterations before convergence and accuracy.

Solution

Measure	LS-SVDD(quadprog)	LS-SVDD(GD)	LS-SVDD(SGD)
Time	2.05	37.47	0.01
Accuracy(Train)	0.62	0	0.14
Accuracy(Test)	0.70	0	0.10
Iteration	-	13812	41

Conclusion
The model obtained from quadprog has the highest accuracy rate