# Project-2

Chitra Karki[*] University of Texas at El Paso (UTEP)

September 26, 2022

## Contents



---

# 1 Import the data

Bring in the data. Remove the first three columns, which are ID variables. Change the value 0 to -1 for Class since we will experiment with a logistic model with +or-1 valued responses.

```
#setwd("C:/Users/chitr/OneDrive - University of Texas at El Paso/data_science/semesters/sem3-f
dat = read.csv(file = "Shill Bidding Dataset.csv",header = T)

# removing first three columns
head(dat)
```

```
##   Record_ID Auction_ID Bidder_ID Bidder_Tendency Bidding_Ratio
## 1         1        732     _***i      0.20000000     0.4000000
## 2         2        732     g***r      0.02439024     0.2000000
## 3         3        732     t***p      0.14285714     0.2000000
## 4         4        732     7***n      0.10000000     0.2000000
```

[*]cbkarki@miners.utep.edu

```
## 5         5        900     z***z      0.05128205      0.2222222
## 6         8        900     i***e      0.03846154      0.1111111
##   Successive_Outbidding Last_Bidding Auction_Bids Starting_Price_Average
## 1                     0 0.0000277778            0              0.9935928
## 2                     0 0.0131226852            0              0.9935928
## 3                     0 0.0030416667            0              0.9935928
## 4                     0 0.0974768519            0              0.9935928
## 5                     0 0.0013177910            0              0.0000000
## 6                     0 0.0168435847            0              0.0000000
##   Early_Bidding Winning_Ratio Auction_Duration Class
## 1  0.0000277778     0.6666667                5     0
## 2  0.0131226852     0.9444444                5     0
## 3  0.0030416667     1.0000000                5     0
## 4  0.0974768519     1.0000000                5     0
## 5  0.0012417328     0.5000000                7     0
## 6  0.0168435847     0.8000000                7     0
```

```
dim(dat) # 6321 rows, 13 colmns
```

```
## [1] 6321   13
```

```
names(dat)
```

```
##  [1] "Record_ID"              "Auction_ID"             "Bidder_ID"
##  [4] "Bidder_Tendency"        "Bidding_Ratio"          "Successive_Outbidding"
##  [7] "Last_Bidding"           "Auction_Bids"           "Starting_Price_Average"
## [10] "Early_Bidding"          "Winning_Ratio"          "Auction_Duration"
## [13] "Class"
```

```
# removing first three columns
dat = dat[,-c(1:3)]
dim(dat); names(dat) # 6321 rows, 10 cols
```

```
## [1] 6321   10
```

```
##  [1] "Bidder_Tendency"        "Bidding_Ratio"          "Successive_Outbidding"
##  [4] "Last_Bidding"           "Auction_Bids"           "Starting_Price_Average"
##  [7] "Early_Bidding"          "Winning_Ratio"          "Auction_Duration"
## [10] "Class"
```

```
# changing 0 to -1 for class variable
table(dat$Class)
```

```
##
##    0    1
## 5646  675
```

```
dat$Class = ifelse(dat$Class==0,-1,1)
table(dat$Class)
```

```
##
##   -1    1
## 5646  675
```

## 2 Exploratory Data Analysis (EDA)

Perform some simple EDA to gain insight of the data.Specifically,

(a) Compute the number of distinct levels or values for each variable. Are there any categorical variable or numerical variable that has only a few distinct values?

(b) Are there any missing data? If so, deal with them with an imputation or list wise deletion accordingly. Document your steps carefully.

(c) Make a parallel boxplot of the data to view the predictors or attributes in the data. Inspect whether they have the same range and variation. This helps us to determine whether scaling is necessary for some modeling approaches.

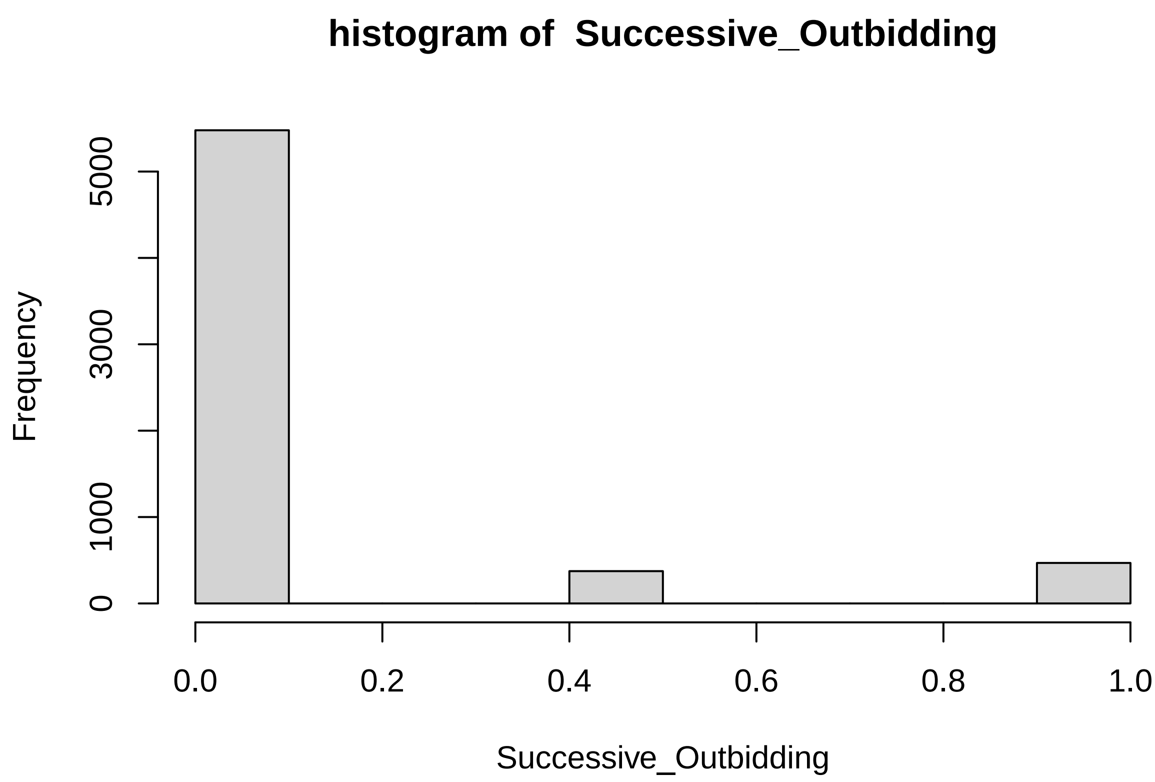
(d) Make a bar plot of the binary response Class. Do we seem to have an unbalanced classification problem?

```
# 2a
str(dat)
```

```
## 'data.frame':    6321 obs. of  10 variables:
##  $ Bidder_Tendency       : num  0.2 0.0244 0.1429 0.1 0.0513 ...
##  $ Bidding_Ratio         : num  0.4 0.2 0.2 0.2 0.222 ...
##  $ Successive_Outbidding : num  0 0 0 0 0 0 0 1 1 0.5 ...
##  $ Last_Bidding          : num  2.78e-05 1.31e-02 3.04e-03 9.75e-02 1.32e-03 ...
##  $ Auction_Bids          : num  0 0 0 0 0 ...
##  $ Starting_Price_Average: num  0.994 0.994 0.994 0.994 0 ...
##  $ Early_Bidding         : num  2.78e-05 1.31e-02 3.04e-03 9.75e-02 1.24e-03 ...
##  $ Winning_Ratio         : num  0.667 0.944 1 1 0.5 ...
##  $ Auction_Duration      : int  5 5 5 5 7 7 7 7 7 7 ...
##  $ Class                 : num  -1 -1 -1 -1 -1 -1 -1 1 1 1 ...
```

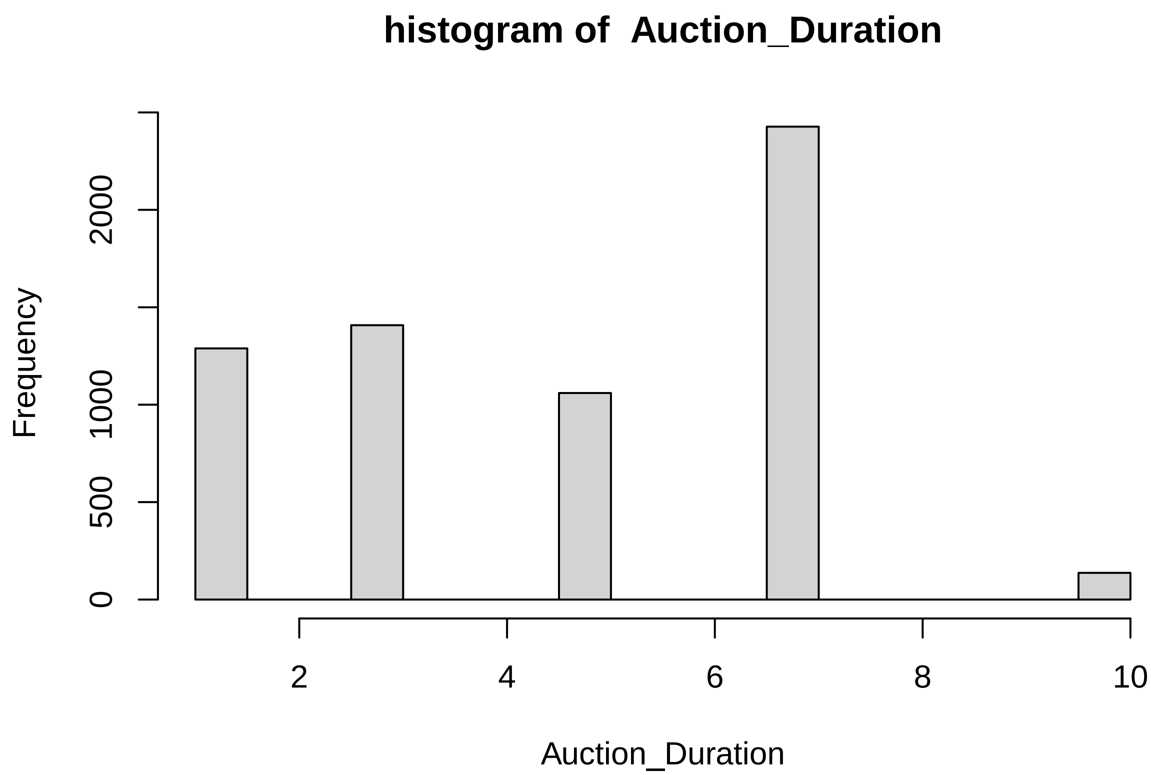
```
# # ploting histograms
# par(mfrow = c(2,5))
xlab = names(dat)
nc = ncol(dat)
# for (i in 1:nc) {
#   hist(dat[,i],xlab = paste(xlab[i]),ylab = "freq",main = paste())
```

```
#
# }


#suppressPackageStartupMessages(library("dplyr"))
# getting varibles with less then 10 unique values
for (i in 1:nc) {
  if (length(unique(dat[,i])) < 10) {
        hist(dat[,i],xlab = paste(xlab[i]),
         main=paste("histogram of ",xlab[i]))

    print(paste("table of variable", xlab[i]))
    print(table(dat[,i]))

    print(paste("The variable",xlab[i],"has",length(unique(dat[,i])),
                "distinct values" ))
  }
}
```

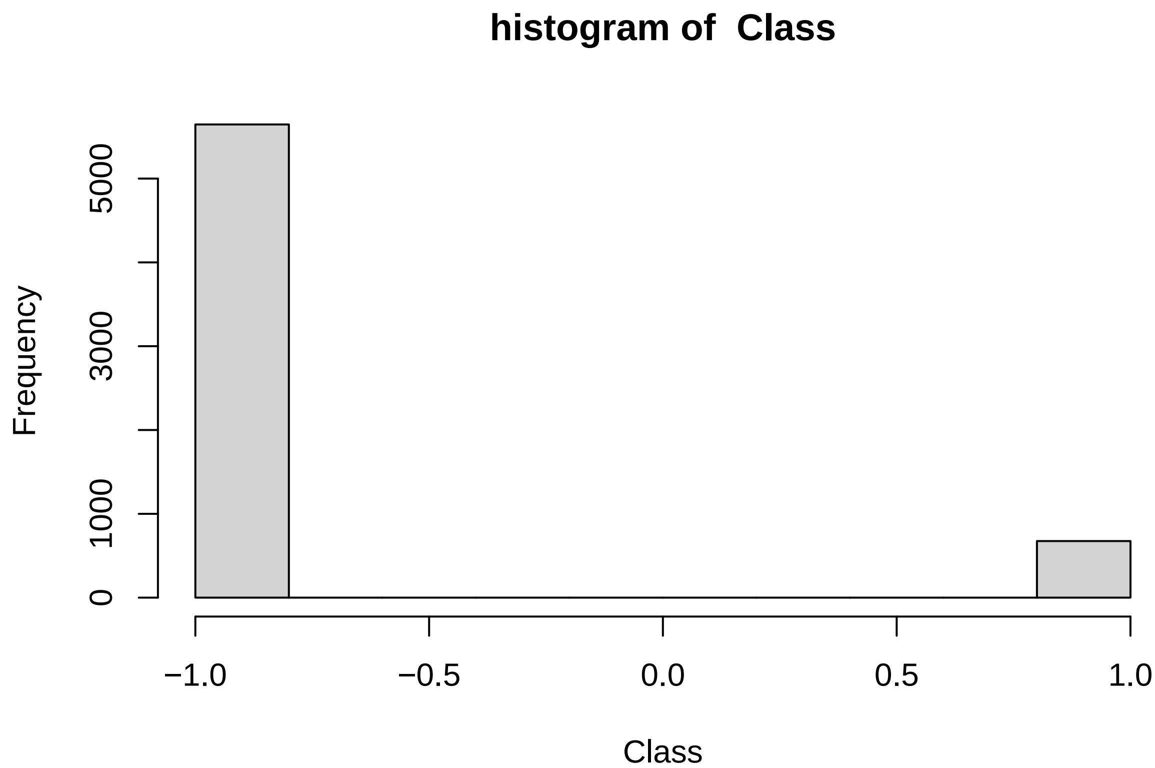



```
## [1] "table of variable Successive_Outbidding"
##
##    0  0.5    1
```

```
## 5478  374  469
## [1] "The variable Successive_Outbidding has 3 distinct values"
```




```
## [1] "table of variable Auction_Duration"
##
##    1    3    5    7   10
## 1289 1408 1060 2427  137
## [1] "The variable Auction_Duration has 5 distinct values"
```




```
## [1] "table of variable Class"
##
##   -1    1
## 5646  675
## [1] "The variable Class has 2 distinct values"
```

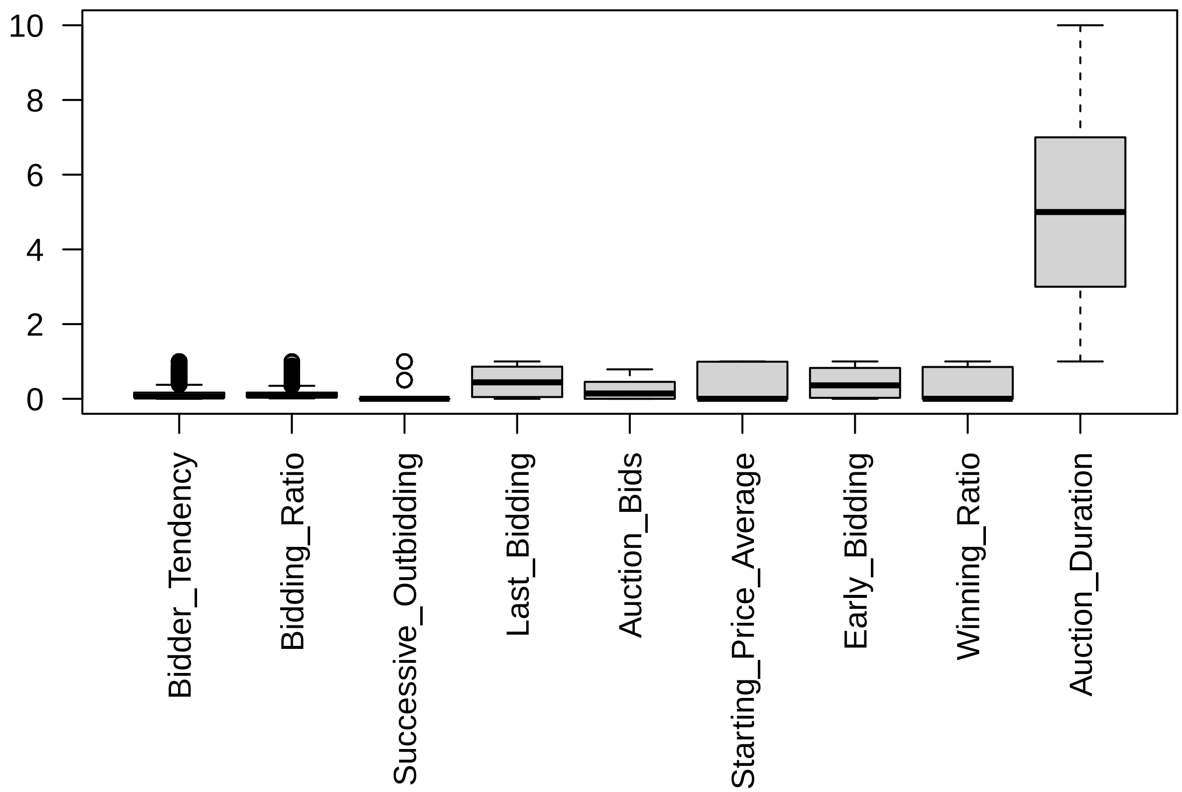
The details of the variables with their corresponding distinct values are mentioned above along with the histogram plots.

```
# 2b
sum(is.na(dat))
```

```
## [1] 0
```

There are no missing values.

```
# 2c
# box plots of predictor variables
par(mar=c(10,2,2,2))
boxplot(dat[,-nc],las=2)
```




The Auction_duration variable has media, which is higher then the other variables.

```
# 2d
barplot(dat$Class)
```


```r
table(dat$Class)
```

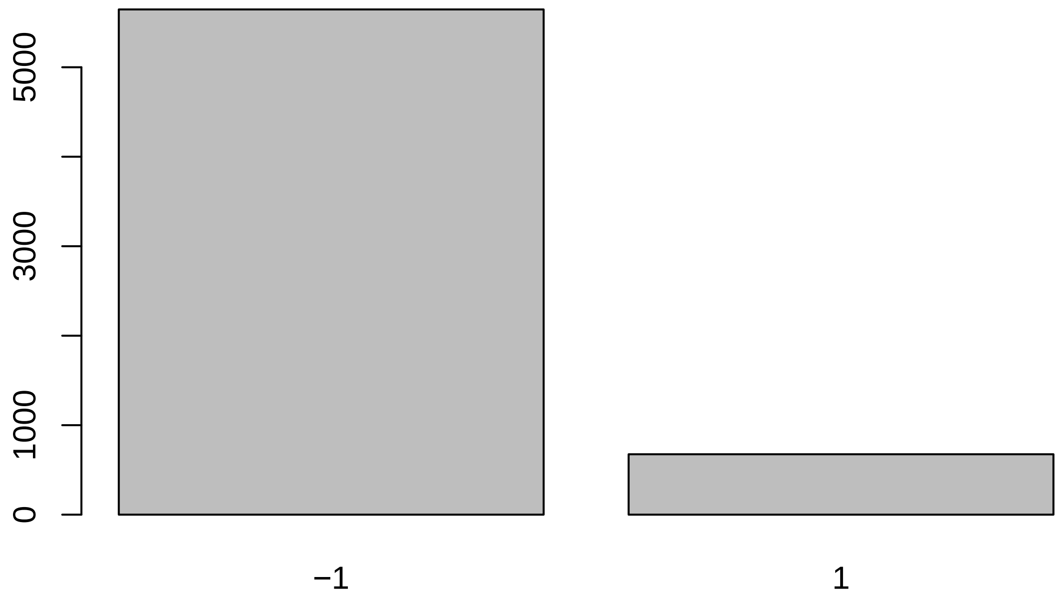
```
##
##   -1    1
## 5646  675
```

```r
675/nrow(dat);5646/nrow(dat)
```

```
## [1] 0.1067869
```

```
## [1] 0.8932131
```

```r
plot(as.factor(dat$Class))
```




Form the barplot and the percentage calculation, it is observed that, +1 has around 10.67% and -1 has 89.32 %.

# 3 Data Partitioning

Partition the data D into three sets: the training data D1, the validation data D2, and the test data D3 with a ratio approximately of 2:1:1.

```
index = 1:nrow(dat)

# d1
d1.index = sample(index,(1/2)*nrow(dat))
d1 = dat[d1.index,]

# d2
d2.index = sample(index[-d1.index],(1/4)*nrow(dat))
d2 = dat[d2.index,]

# d3
d3 = dat[-c(d1.index,d2.index),]
```

# 4 Logistic Regression – Optimization

Referring to class notes 'Introduction to Optimization' and the R example, implement the logistic regression model by minimizing the negative loglikelihood function.

(a) Pool the training data and the validation data together into D' = D1∪D2. Based on D', obtain the maximum likelihood estimates (MLE) $\hat{\beta} = (\hat{\beta}_j)$ of regression parameters and their standard errors from the resultant Hessian matrix. Test the significance of each attribute and obtain the corresponding p-values. Tabulate the results. Also, specify the optimization method that you use in R function optim(), e.g., BFGS. Check to make sure that the algorithm converges by looking at the "convergence" value in the output, which should be 0 if success.

(b) Compare your results in 4(a) with the fittting results from standard R function **glm()**.

(c) Apply your trained logistic model in 4(a) to predict the response in the test data D3: Specifically, let X' denote the design matrix from the test data; don't forget to add the first column of all 1's. We have
$\hat{y}' = sig[\frac{exp((X')\hat{\beta})}{1+exp((X')\hat{\beta})} - 0.5]$
with the default threshold 0.5. Compute the prediction accuracy.

```
d.prime = rbind(d1,d2)

# THE NEGATIVE LOGLIKEHOOD FUNCTION FOR Y=+1/-1
# non-negative loglikihood
nloglik <- function(beta, X, y){
    if (length(unique(y)) !=2) stop("Are you sure you've got Binary Target?")
    X <- cbind(1, X)
    nloglik <- sum(log(1+ exp(-y*X%*%beta)))
    return(nloglik)
}
```

```
#Preparing data to run optim function
X <- as.matrix(d.prime[,-ncol(d.prime)])
y = d.prime[,ncol(d.prime)]
p <- NCOL(X) +1
fit <- optim(par=rep(0,p), fn=nloglik, method="BFGS", X=X, y=y,hessian = T)
beta.hat <- fit$par; beta.hat
```

```
##  [1] -11.4726205   0.5274209   1.2386191  10.9779409   0.7665486   0.6185415
##  [7]   0.1732276  -0.5524577   5.8793456   0.1281143
```

```
# convergence
fit$convergence
```

```
## [1] 0
```

```r
### The square roots of the diagonal elements of the inverse of the Hessian
#(or the negative Hessian , when minimizing Likelihood) are the estimated
#standard errors.

vcov = solve(fit$hessian)
# standard errors
se.beta.hat = sqrt(diag(vcov)) ; se.beta.hat
```

```
##  [1] 0.91379174 0.58581934 0.98479707 0.71545748 0.76913884 0.73676345
##  [7] 0.32597460 0.75800769 0.72666059 0.05299258
```

The optimization algorithm connverged to 0.

```r
# 2(b)

# testing the significance of each attributes
# Wald"s test h0: beta = 0 not significant , h1: beta!=0
# test statistic, Z = beta(i)/sd(beta(i))
# use standard normal curve to determine p-values.

z.beta.hat = beta.hat/se.beta.hat
suppressPackageStartupMessages(library(scales))
p.value = 2*pnorm(abs(z.beta.hat),lower.tail = F) # 2-tail test
decission = ifelse(p.value<0.05,"significant","not-significant")

data.frame(beta.hat = paste("beta",0:(ncol(d.prime)-1)),
           beta.hat = round(beta.hat,4), se.beta.hat=round(se.beta.hat,4),
           z.beta.hat= round(z.beta.hat,4),p.value = scientific(p.value,4),
           decission= decission)
```

```
##    beta.hat beta.hat.1 se.beta.hat z.beta.hat   p.value       decission
## 1    beta 0   -11.4726      0.9138   -12.5550 3.734e-36     significant
## 2    beta 1     0.5274      0.5858     0.9003 3.680e-01 not-significant
## 3    beta 2     1.2386      0.9848     1.2577 2.085e-01 not-significant
## 4    beta 3    10.9779      0.7155    15.3439 3.888e-53     significant
## 5    beta 4     0.7665      0.7691     0.9966 3.189e-01 not-significant
## 6    beta 5     0.6185      0.7368     0.8395 4.012e-01 not-significant
## 7    beta 6     0.1732      0.3260     0.5314 5.951e-01 not-significant
## 8    beta 7    -0.5525      0.7580    -0.7288 4.661e-01 not-significant
## 9    beta 8     5.8793      0.7267     8.0909 5.922e-16     significant
## 10   beta 9     0.1281      0.0530     2.4176 1.562e-02     significant
```

```r
y.prime.org = ifelse(d.prime$Class==-1,0,1)
fit0 <- glm( y.prime.org~.,data=d.prime[,-ncol(d.prime)],
             family=binomial(link = 'logit'))
summary(fit0)
```

```
##
## Call:
## glm(formula = y.prime.org ~ ., family = binomial(link = "logit"),
##     data = d.prime[, -ncol(d.prime)])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.8997  -0.0751  -0.0091  -0.0068   3.0179
##
## Coefficients:
##                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)           -11.47261    0.91379 -12.555  < 2e-16 ***
## Bidder_Tendency         0.52735    0.58582   0.900   0.3680
## Bidding_Ratio           1.23861    0.98480   1.258   0.2085
## Successive_Outbidding  10.97792    0.71545  15.344  < 2e-16 ***
## Last_Bidding            0.76654    0.76914   0.997   0.3189
## Auction_Bids            0.61850    0.73676   0.839   0.4012
## Starting_Price_Average  0.17321    0.32597   0.531   0.5952
## Early_Bidding          -0.55245    0.75801  -0.729   0.4661
## Winning_Ratio           5.87940    0.72666   8.091 5.92e-16 ***
## Auction_Duration        0.12811    0.05299   2.418   0.0156 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3266.53  on 4739  degrees of freedom
## Residual deviance:  434.16  on 4730  degrees of freedom
## AIC: 454.16
##
## Number of Fisher Scoring iterations: 10
```

The results for optimization algorithm and the glm() function are comparable, which can be observed from table and the summary of glm fit above. The significant attributes are the same from these two procedures.

```
# 2(c)

x.prime = data.frame(Intercept=rep(1,nrow(d3)),d3[,-ncol(d3)])

predict.opt = function(x,beta) {
  x = as.matrix(x)
  y.hat = sign(exp(x%*%beta)/(1 + exp(x%*%beta))-0.5)
  return(y.hat)
}

y.hat = predict.opt(x.prime,beta.hat)
```

```r
#confusion matrix
table(y.hat,d3[,ncol(d3)])
```

```
##
## y.hat   -1    1
##    -1 1400   16
##    1    23  142
```

```r
#accuracy
sum(y.hat == d3[,ncol(d3)])/nrow(d3)
```

```
## [1] 0.9753321
```

```r
#table(y.hat)
```

The accuracy of the model is 0.9784946. Meaning that for approximately 98 percentage of the times it predicted positive for positive and negative for negative in testing data set.

# 5 Primitive LDA-The kernel trick

```r
# (a)
x1 = d1[,-ncol(d1)];
x1.y = d1[,ncol(d1)]

x2 = d2[,-ncol(d2)]
x2.y = d2[,ncol(d2)]

x3 = d3[,-ncol(d3)]
x3.y = d3[,ncol(d3)]

# scaling x1
x1.mean = as.vector(apply(x1, 2, mean))
x1.sd = as.vector(apply(x1,2, sd))

x1.scaled = scale(x1,center = x1.mean,scale = x1.sd)

# scaling x2

x2.scaled = scale(x2,center = x1.mean,scale = x1.sd)
```

```
# (b) training kernel and tunning sigma for rbf kernel
library("kernlab")
```

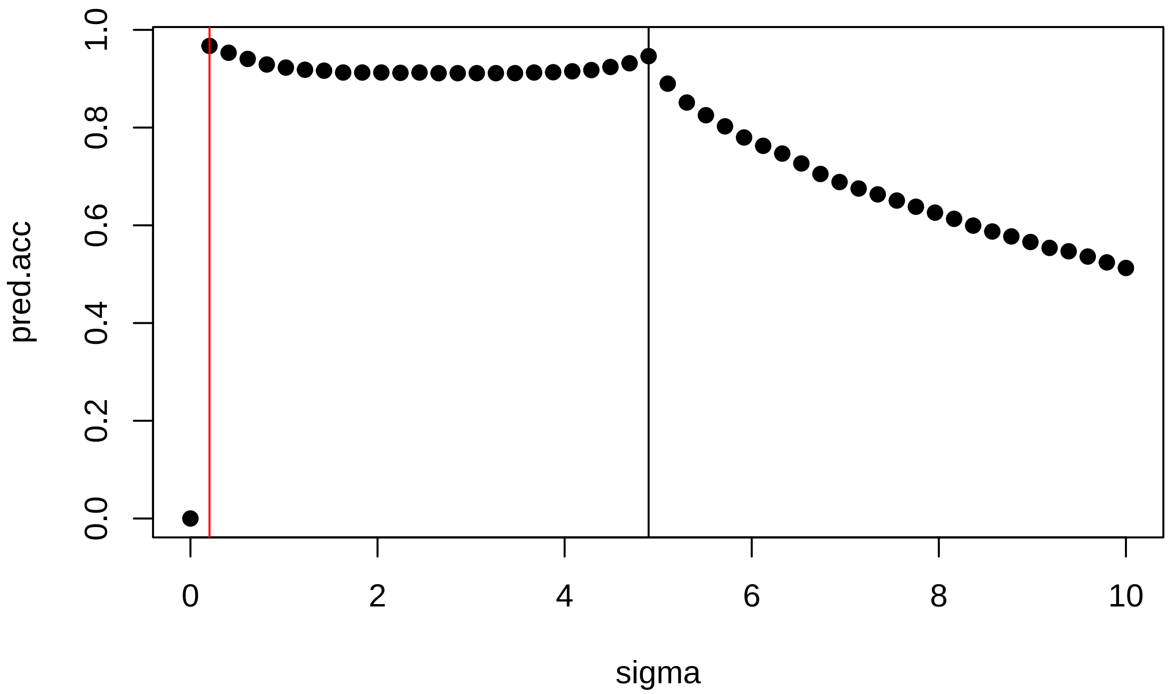
```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:scales':
##
##     alpha
```

```
sigma = seq(0,10,length.out = 50)

pred.acc = NULL
for(i in 1:length(sigma)) {

#initialization kernel
kernel.class = rbfdot(sigma = sigma[i])

# linear classifier y = sgn(wz+b)
wz.plus = colMeans(kernelMatrix(kernel = kernel.class,
                           x=as.matrix(x1.scaled[which(x1.y==1),]),
                           y = as.matrix(x2.scaled)))

wz.minus =  colMeans(kernelMatrix(kernel = kernel.class,
                           x=as.matrix(x1.scaled[which(x1.y==-1),]),
                           y = as.matrix(x2.scaled)))

wz = wz.plus - wz.minus
b = (mean(kernelMatrix(kernel = kernel.class,
                           x=as.matrix(x1.scaled[which(x1.y==1),]),
                       y = as.matrix(x1.scaled[which(x1.y==1),]))) -
       mean(kernelMatrix(kernel = kernel.class,
                           x=as.matrix(x1.scaled[which(x1.y==-1),]),
                           y = as.matrix(x1.scaled[which(x1.y==-1),])))) * 0.5

y.hat = sign(wz+b)
pred.acc[i] = mean(y.hat == x2.y)

print(paste("prediction acurracy for",round(sigma[i],4),":",
            round(pred.acc[i],4)))
}
```

```
## [1] "prediction acurracy for 0 : 0"
## [1] "prediction acurracy for 0.2041 : 0.9671"
## [1] "prediction acurracy for 0.4082 : 0.9532"
## [1] "prediction acurracy for 0.6122 : 0.9405"
```

```
## [1] "prediction acurracy for 0.8163 : 0.9291"
## [1] "prediction acurracy for 1.0204 : 0.9228"
## [1] "prediction acurracy for 1.2245 : 0.9184"
## [1] "prediction acurracy for 1.4286 : 0.9165"
## [1] "prediction acurracy for 1.6327 : 0.9127"
## [1] "prediction acurracy for 1.8367 : 0.9127"
## [1] "prediction acurracy for 2.0408 : 0.9127"
## [1] "prediction acurracy for 2.2449 : 0.912"
## [1] "prediction acurracy for 2.449 : 0.9127"
## [1] "prediction acurracy for 2.6531 : 0.9114"
## [1] "prediction acurracy for 2.8571 : 0.9114"
## [1] "prediction acurracy for 3.0612 : 0.9114"
## [1] "prediction acurracy for 3.2653 : 0.9114"
## [1] "prediction acurracy for 3.4694 : 0.9114"
## [1] "prediction acurracy for 3.6735 : 0.9127"
## [1] "prediction acurracy for 3.8776 : 0.9133"
## [1] "prediction acurracy for 4.0816 : 0.9152"
## [1] "prediction acurracy for 4.2857 : 0.9177"
## [1] "prediction acurracy for 4.4898 : 0.9241"
## [1] "prediction acurracy for 4.6939 : 0.9316"
## [1] "prediction acurracy for 4.898 : 0.9462"
## [1] "prediction acurracy for 5.102 : 0.8899"
## [1] "prediction acurracy for 5.3061 : 0.8513"
## [1] "prediction acurracy for 5.5102 : 0.8253"
## [1] "prediction acurracy for 5.7143 : 0.8025"
## [1] "prediction acurracy for 5.9184 : 0.7797"
## [1] "prediction acurracy for 6.1224 : 0.7627"
## [1] "prediction acurracy for 6.3265 : 0.7468"
## [1] "prediction acurracy for 6.5306 : 0.7266"
## [1] "prediction acurracy for 6.7347 : 0.7051"
## [1] "prediction acurracy for 6.9388 : 0.6886"
## [1] "prediction acurracy for 7.1429 : 0.6753"
## [1] "prediction acurracy for 7.3469 : 0.6633"
## [1] "prediction acurracy for 7.551 : 0.6506"
## [1] "prediction acurracy for 7.7551 : 0.638"
## [1] "prediction acurracy for 7.9592 : 0.6259"
## [1] "prediction acurracy for 8.1633 : 0.6133"
## [1] "prediction acurracy for 8.3673 : 0.5994"
## [1] "prediction acurracy for 8.5714 : 0.5873"
## [1] "prediction acurracy for 8.7755 : 0.5772"
## [1] "prediction acurracy for 8.9796 : 0.5658"
## [1] "prediction acurracy for 9.1837 : 0.5538"
## [1] "prediction acurracy for 9.3878 : 0.5468"
## [1] "prediction acurracy for 9.5918 : 0.5361"
## [1] "prediction acurracy for 9.7959 : 0.5241"
## [1] "prediction acurracy for 10 : 0.5127"
```

```
plot(y=pred.acc,x=sigma,pch=19);
abline(v=sigma[which(pred.acc==max(pred.acc))],col="red")
abline(v= 4.898)
```




```
# sigma which has maximum prediction accuracy
sigma[which(pred.acc==max(pred.acc))]
```

```
## [1] 0.2040816
```

The tuning parameter for sigma in rbf kernel function was set between 0 to 10 in a step size of 0.2041. The maximum prediction accuracy was produced by the sigma(0.2040816) with accuracy 0.9949. After sigma(4.898) the accuracy started decreasing in a faster rate.

```
#(c)
d.prime = rbind(d1,d2)
X.prime = d.prime[,-ncol(d.prime)]
X.prime.mean = colMeans(X.prime)
X.prime.sd = apply(X.prime, 2,sd)

# scaling
X.prime.scaled = scale(X.prime,center = X.prime.mean,scale = X.prime.sd)
```

```r
X3.scaled = scale(d3[,-ncol(d3)],center = X.prime.mean,scale = X.prime.sd )

# prediction on d3 based on the best kernel in 5c
kernel.class = rbfdot(sigma = sigma[which(pred.acc==max(pred.acc))])

# linear classifier y = sgn(wz+b)
wz.plus = colMeans(kernelMatrix(kernel = kernel.class,
            x=as.matrix(X.prime.scaled[which(d.prime[,ncol(d.prime)]==1),]),
            y = as.matrix(X3.scaled)))

wz.minus =  colMeans(kernelMatrix(kernel = kernel.class,
            x=as.matrix(X.prime.scaled[which(d.prime[,ncol(d.prime)]==-1),]),
            y = as.matrix(X3.scaled)))

wz = wz.plus - wz.minus
b = (mean(kernelMatrix(kernel = kernel.class,
              x=as.matrix(X.prime.scaled[which(d.prime[,ncol(d.prime)]==1),]),
           y = as.matrix(X.prime.scaled[which(d.prime[,ncol(d.prime)]==1),]))) -
       mean(kernelMatrix(kernel = kernel.class,
x=as.matrix(X.prime.scaled[which(d.prime[,ncol(d.prime)]==-1),]),
y = as.matrix(X.prime.scaled[which(d.prime[,ncol(d.prime)]==-1),])))) * 0.5

y.hat = sign(wz+b)

# prediction accuracy
mean(y.hat == d3[,ncol(d3)])
```

```
## [1] 0.974067
```

The prediction accuracy on the testing set is 0.970272. The value of the sigma was tuned from 5b.