Project 2: Optimization and Kernel Trick

Appiah Prince[*] University of Texas at El Paso (UTEP)

September 27, 2022

Contents

[*]pappiah@miners.utep.edu

1 Bring in the data

```
data <- read.csv('Shill Bidding Dataset.csv')
names(data)
```

```
##  [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"
```

```
# Remove the first three columns
shill_bidding <- data[, -c(1:3)]
names(shill_bidding)
```

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

```
#change the 0 value of the class variable to -1
shill_bidding$Class[shill_bidding$Class == 0] <- -1
#shill_bidding$Class <- ifelse(shill_bidding$Class == 0, -1, 1)
table(shill_bidding$Class)
```

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

```
dim(shill_bidding)
```

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

Remarks

- The data Shill Bidding was loaded and the first three columns were remove since these were ID variables.
- The original dimension of the data was 6321 rows and 13 columns. However, since we removed three columns, our new dimension is 6,321 rows and 10 columns.
- I printed a table of the target variable(class) to confirm that the level 0 has been changed to -1.

2 Exploratory Data Analysis (EDA)

2.1 Distinct levels or values for each variable

```
aggregate(values ~ ind, unique(stack(shill_bidding)), length)
```

```
##                        ind values
## 1           Bidder_Tendency    489
## 2             Bidding_Ratio    400
## 3   Successive_Outbidding      3
## 4              Last_Bidding   5807
## 5              Auction_Bids     49
## 6  Starting_Price_Average     22
## 7             Early_Bidding   5690
## 8             Winning_Ratio     72
## 9          Auction_Duration      5
## 10                    Class      2
```

```
str(shill_bidding)
```

```
## '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 ...
```

Remarks

The numerical variables **class** and **Successive_Outbidding** have only few distinct values.

2.2 Missing Values

```
library(questionr)
freq.na(shill_bidding)
```

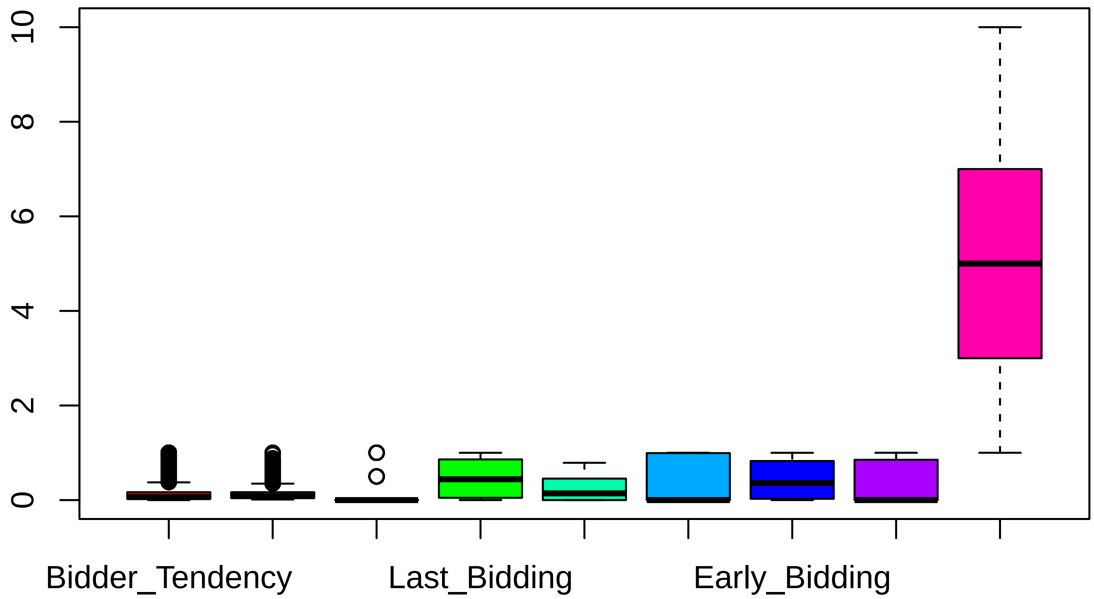
```
##                        missing %
## Bidder_Tendency              0 0
## Bidding_Ratio                0 0
## Successive_Outbidding        0 0
## Last_Bidding                 0 0
## Auction_Bids                 0 0
## Starting_Price_Average       0 0
## Early_Bidding                0 0
## Winning_Ratio                0 0
## Auction_Duration             0 0
## Class                        0 0
```

Remarks

There are no missing values in the data

2.3 Parallel Boxplot of the Data

```
boxplot(shill_bidding[,-10], col = rainbow(ncol(shill_bidding[,-10])))
```



Remarks

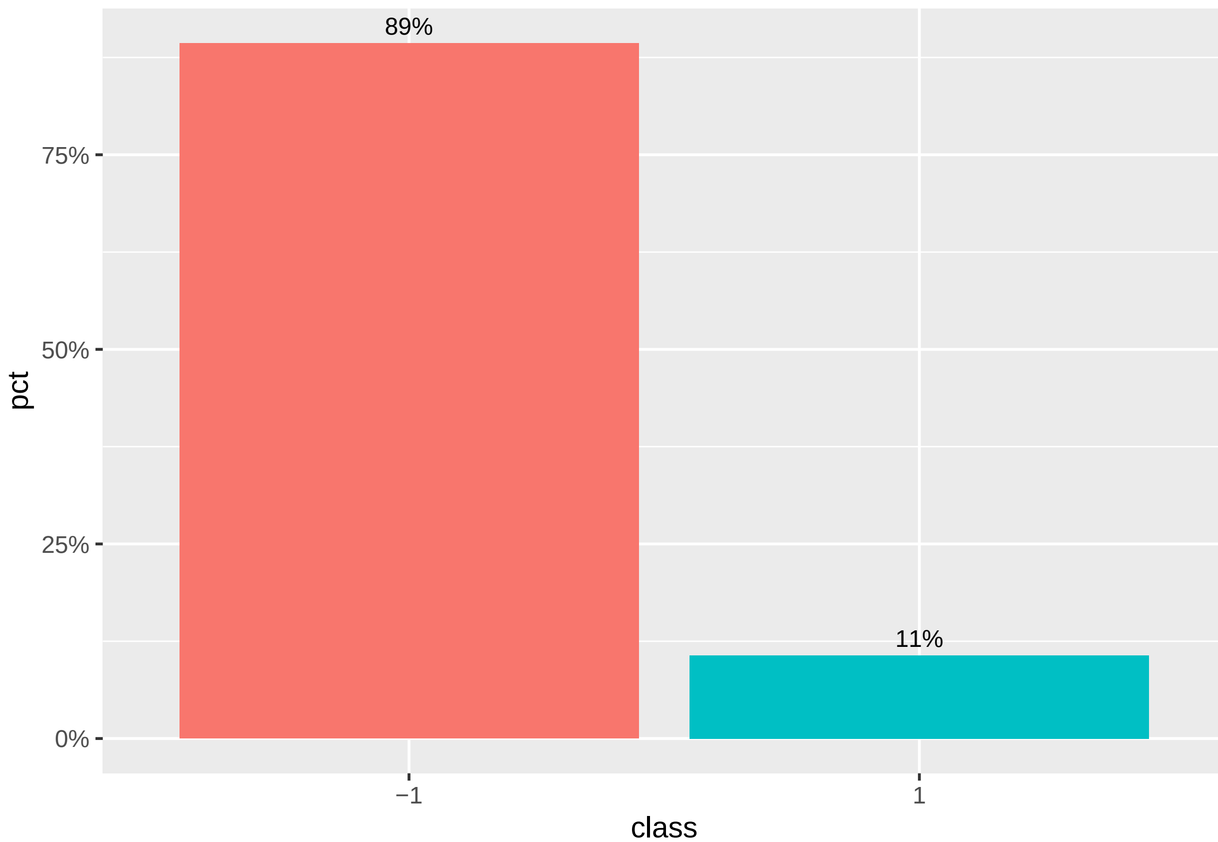
- The predictors have unequal range and unequal variation. In particular, the predictors **Auction__Duration**,**Starting__Price__Average** ,**Winning__Ratio** and **Successive__Outbidding** have notable unequal range and variation.
- Hence,scaling is necessary for some modeling approaches.

2.4 Bar plot of the binary response Class

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
shill_bidding %>%
    count(class = factor(Class)) %>%
    mutate(pct = prop.table(n)) %>%
    ggplot(aes(x = class, y = pct, label = scales::percent(pct), fill=class)) +
    geom_col(position = 'dodge') +
    geom_text(position = position_dodge(width = .9),
              vjust = -0.5,
              size = 3) +
    scale_y_continuous(labels = scales::percent) +
    theme(legend.position = "none")
```



Remarks

We see from the barplot that the percentage of 1's is 11% and percentage of -1's is 89%. Therefore, we do not have an unbalanced classification problem.

3 Data Partitioning

```r
set.seed(125)
Data <- sample(seq(1, 3), size = nrow(shill_bidding), replace = TRUE,
               prob = c(0.5, 0.25, 0.25))
train_data <- shill_bidding[Data == 1, ] # training data
validation_data <- shill_bidding[Data == 2, ] # validation data
test_data <- shill_bidding[Data == 3, ] # test data

dim(train_data)
```

```
## [1] 3171   10
```

```r
dim(validation_data)
```

```
## [1] 1596   10
```

```r
dim(test_data)
```

```
## [1] 1554   10
```

Remarks

- There are 3171 observations and 10 variables in the training data.
- There are 1596 observations and 10 variables in the validation data.
- There are 1554 observations and 10 variables in the test data.

4 Logistic Regression - Optimization

- Part 4(a)

4.1 Pool the training data and the validation data together

```r
train_valid_data <- rbind(train_data, validation_data)
#head(train_valid_data)
dim(train_valid_data)
```

```
## [1] 4767   10
```

4.2 Negative Likelihood Function and Test On the shill_bidding data

```
# THE NEGATIVE LOGLIKEHOOD FUNCTION FOR Y=+1/-1
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)
}

y <- train_valid_data$Class
X <- as.matrix(train_valid_data[, c(1:9)])
p <- NCOL(X) +1
fit <- optim(par=rep(0,p), fn=nloglik, method="BFGS", X=X, y=y,
             hessian = TRUE)
beta.hat <- fit$par # obtaining the regression parameters
beta.hat
```

```
##  [1] -10.10785135   1.05384204   1.25123047  10.49377567   0.93247462
##  [6]   0.63452561   0.12535724  -0.64309290   4.77652987   0.05764265
```

Remarks

The optimization method that was employed in R function optim() is BFGS

4.3 Standard error from the Hessian matrix

```
hessian <- fit$hessian # Hessian matrix
inv_hessian <- solve(hessian)
standard_error <- sqrt(diag(inv_hessian))
standard_error
```

```
##  [1] 0.77083290 0.53101537 0.96953407 0.64970474 0.77500494 0.73158830
##  [7] 0.33141911 0.77395553 0.63178907 0.05016893
```

4.4 Convergence of the algorithm

```
fit$convergence
```

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

Remarks The algorithm converges since the output of fit$convergence is 0.

4.5 Testing the significance of each attribute and table of results

```
p0 <- length(beta.hat)-1
z.wald <- beta.hat/standard_error
pvalue <- pchisq(z.wald^2, df=1, lower.tail=FALSE)
result <- data.frame(beta.hat, standard_error, z.wald, pvalue)
row.names(result) <- c("Intercept", names(shill_bidding[, -10]))
round(result, digits = 4)
```

```
##                         beta.hat standard_error   z.wald pvalue
## Intercept               -10.1079         0.7708 -13.1129 0.0000
## Bidder_Tendency           1.0538         0.5310   1.9846 0.0472
## Bidding_Ratio             1.2512         0.9695   1.2905 0.1969
## Successive_Outbidding    10.4938         0.6497  16.1516 0.0000
## Last_Bidding              0.9325         0.7750   1.2032 0.2289
## Auction_Bids              0.6345         0.7316   0.8673 0.3858
## Starting_Price_Average    0.1254         0.3314   0.3782 0.7052
## Early_Bidding            -0.6431         0.7740  -0.8309 0.4060
## Winning_Ratio             4.7765         0.6318   7.5603 0.0000
## Auction_Duration          0.0576         0.0502   1.1490 0.2506
```

Remarks

- Taken $\alpha = 0.05$ as a threshold, we see from our results that the p-values for the predictors **Bidder_Tendency**, **Successive_Outbidding** and **Winning_Ratio** are less that $\alpha = 0.05$. Hence, these attributes are statistically significant.
- Part (4b)

4.6 Comparing results in 4(a) with fitting results from glm()

```r
y <- factor(train_valid_data$Class)
fit.logit <- glm(y~Bidder_Tendency+Bidding_Ratio+Successive_Outbidding+
                   Last_Bidding+Auction_Bids+Starting_Price_Average+
                   Early_Bidding+Winning_Ratio+Auction_Duration,
                 data=train_valid_data, family=binomial(link = "logit"))
result <- summary(fit.logit)
round(result$coefficients, 4)
```

```
##                        Estimate Std. Error  z value Pr(>|z|)
## (Intercept)            -10.1079     0.7708 -13.1135   0.0000
## Bidder_Tendency          1.0539     0.5310   1.9847   0.0472
## Bidding_Ratio            1.2512     0.9695   1.2906   0.1969
## Successive_Outbidding   10.4938     0.6497  16.1522   0.0000
## Last_Bidding             0.9325     0.7750   1.2032   0.2289
## Auction_Bids             0.6345     0.7316   0.8673   0.3858
## Starting_Price_Average   0.1254     0.3314   0.3782   0.7052
## Early_Bidding           -0.6431     0.7739  -0.8309   0.4060
## Winning_Ratio            4.7765     0.6318   7.5606   0.0000
## Auction_Duration         0.0576     0.0502   1.1490   0.2506
```

```r
fit.logit$converged
```

```
## [1] TRUE
```

Remarks

- The glm() result also converges.
- Taken $\alpha = 0.05$ as a threshold, we see from our results that the p-values for the attributes **Bidder_Tendency**, **Successive_Outbidding** and **Winning_Ratio** are less that $\alpha = 0.05$. Hence, these attributes are statistically significant.
- The coefficient of Winning_Ratio says that, holding the other predictors at a fixed value, we will see $e^{4.7765} = 11868.82\%$ increase in the odds of getting into a positive Class for a unit increase in Winning_Ratio.
- There appears to be no difference between the results of the two methods.
- Part 4(c)

4.7 Making prediction using the test data

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

test_data <- test_data
new_X <- as.matrix(cbind(1,test_data[,-10]))
y_hat_prime <- sign(my_fun(new_X%*%fit$par) - 0.5)
conf_matt <- table(test_data$Class, y_hat_prime) # gives the confusion matrix
#round(mean(test_data$Class == y_hat_prime),)
pred_acc <- sum(diag(conf_matt))/sum(conf_matt) # gives the accuracy
pred_acc
```

```
## [1] 0.9787645
```

Remarks

With a threshold of 0.5, our prediction accuracy is 0.9787645. This means, the algorithm in 4(a) predicts or models the data very well.

5 Primitive LDA (The Kernel Trick)

5.1 Matrix of all predictors and Scaling X1 and X2

- Part 5(a)

```
# matrix of all predictors for the three train, validation and test sets
X1 <- as.matrix(train_data[,-10])
X2 <- as.matrix(validation_data[,-10])
X3 <- as.matrix(test_data[,-10])

#scale X1
X1_scale <- scale(X1, center = TRUE, scale = TRUE)

# scale X2 according to the column means and SDs computed from X1
mean_X1 <- attributes(X1_scale)$'scaled:center'
sd_X1 <- attributes(X1_scale)$'scaled:scale'
X2_scale <- scale(X2, center = mean_X1, scale = sd_X1)
```

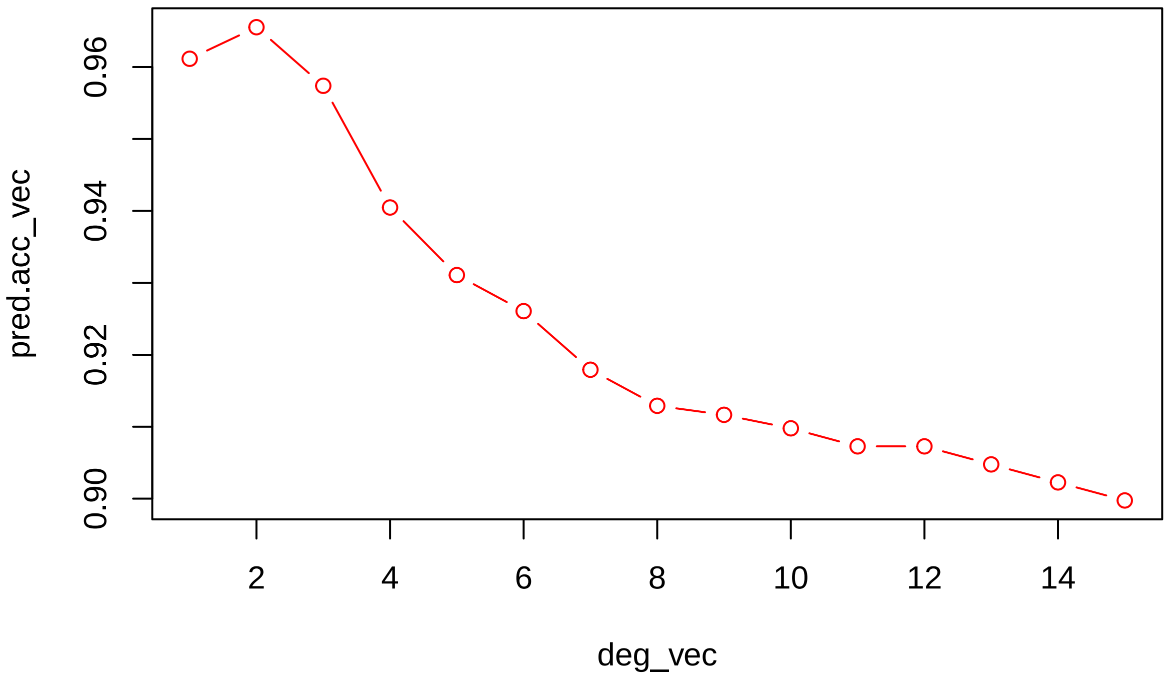
5.2 Train the primitive LDA classifier with D1 and use the prediction accuracy on D2

- Part 5(b)

```
library(kernlab)
```

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

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
LDA_P <- function (kernel, X, Y=NULL, target) {
    kernmat <- kernelMatrix
    w.z <- colMeans(kernmat(kernel, x=X[target==1,], y=Y)) -
             colMeans(kernmat(kernel, x=X[target==-1,], y=Y))
    b <- (mean(kernmat(kernel, X[target==-1,])) -
    mean(kernmat(kernel, X[target==1,])))*.5
    yhat <- sign(w.z + b)
    return(yhat)
}

deg_vec <- 1:15
pred.acc_vec <- rep(0, length(deg_vec))
for (i in 1:length(deg_vec)) {
    d <- deg_vec[i]
     kern <- polydot(degree = d, offset = 1, scale = 1)

    #compute prediction accuracy
    ypred <- LDA_P(kern, X1_scale, X2_scale, train_data$Class)
    yobserved <- validation_data$Class
    conf_mat <- table(ypred, yobserved)
    pred_accuracy <- sum(diag(conf_mat))/sum(conf_mat)
    pred.acc_vec[i] <- pred_accuracy

}
 plot(deg_vec, pred.acc_vec, type = "b", col="red")
```




```
max(pred.acc_vec); min(pred.acc_vec)
```

```
## [1] 0.9655388
```

```
## [1] 0.8997494
```

Remarks

- The kernel family used was polynomial kernel.
- From the plot of the prediction accuracy values versus the candidate parameter values,the best choice of our parameter(degree) is 2.
- The maximum value of the prediction accuracy is **0.9655388**. Thus, we see that the polynomial kernel helps well in the classification.
- The minimum value of the prediction accuracy is **0.8997494**.

5.3 Apply the trained classifier with the 'best' kernel found in 5(b) to the test data D3.

- Part 5(c)

```r
# Scale X.prime
X_prime <- as.matrix(train_valid_data[, -10])
X_prime_scale <- scale(X_prime, center = TRUE,   scale = TRUE)

# Scale X3 according to the column means and SDs computed from X.prime
mean_X_prime_scale <- attributes(X_prime_scale)$'scaled:center'
sd_X_prime_scale <- attributes(X_prime_scale)$'scaled:scale'
X3_scale <- scale(X3, center = mean_X_prime_scale , scale = sd_X_prime_scale)
```

```r
# Apply the best kernel to the test data
kern <- polydot(degree = which.max(pred.acc_vec))

    #compute prediction accuracy
    ypred <- LDA_P(kern, X_prime_scale, X3_scale, train_valid_data$Class)
    yobserved <- test_data$Class
    conf_mat <- table(ypred, yobserved)
    pred_accuracy_test <- sum(diag(conf_mat))/sum(conf_mat)
    pred_accuracy_test
```

```
## [1] 0.972973
```

Remarks

- The prediction accuracy obtained after applying the trained classifier with the ‘best’ kernel found in 5(b) to the test data is **0.972973**. This shows an excellent prediction or classification ability of the model.

5.4 Comparison of the prediction accuracy obtained in 4(c) and 5(c)

```r
comparison <- data.frame(pred_accuracy_test,pred_acc )
knitr::kable(comparison,
        col.names = c("Primitive LDA", "Logistic regression (optimization)"),
        caption = "Prediction accuracy", align = "cc")
```

Table 1: Prediction accuracy

Primitive LDA	Logistic regression (optimization)
0.972973	0.9787645

Remarks

There is a small difference between the prediction accuracy obtained in 4(c) and 5(c)