Project II: Optimization and Kernel Methods

DS 5494 - Statistical Machine Learning II

Willliam Ofosu Agyapong* University of Texas at El Paso (UTEP)

September 27, 2022

Contents

*woagyapong@miners.utep.edu

Introduction

In this project, we consider the Shill Bidding data set available from UCI Machine Learning Repository http://archive.ics.uci.edu/ml/datasets/Shill+Bidding+Dataset. The data set is based on eBay auctions of a popular product.

The objective of the study is **to predict whether or not a bidding has normal behavior based on its characteristics**. The data set has 6,321 rows and 13 columns. A binary indicator in the last column, Class, will be our target variable. Table 1 provides details about the variables contained in the data set.

Table 1: Variable Description

Variable name	Description
Record ID	Unique identifier of a record in the dataset.
Auction ID	Unique identifier of an auction.
Bidder ID	Unique identifier of a bidder.
Bidder Tendency	A shill bidder participates exclusively in auctions of few sellers rather than a diversified lot. This is a collusive act involving the fraudulent seller and an accomplice.
Bidding Ratio	A shill bidder participates more frequently to raise the auction price and attract higher bids from legitimate participants.
Successive Outbidding	A shill bidder successively outbids himself even though he is the current winner to increase the price gradually with small consecutive increments.
Last Bidding	A shill bidder becomes inactive at the last stage of the auction (more than 90% of the auction duration) to avoid winning the auction.
Auction Bids	Auctions with SB activities tend to have a much higher number of bids than the average of bids in concurrent auctions.
Auction Starting Price	a shill bidder usually offers a small starting price to attract legitimate bidders into the auction.
Early Bidding	A shill bidder tends to bid pretty early in the auction (less than 25% of the auction duration) to get the attention of auction users.
Winning Ratio	A shill bidder competes in many auctions but hardly wins any auctions.
Auction Duration	How long an auction lasted.
Class	0 for normal behavior bidding; 1 for otherwise.

1 Bringing in the Shill Bidding dataset

Here, we import the csv file containing the data. From Table 1, we can see that the first three columns are ID variables. These variables are not of any predictive importance, so we remove them.

Also, to be able to experiment with a logistic regression model with ± 1 valued responses we changed the value 0 to -1 for the target variable.

```
# importing data
# shill_bidding_orig <- readr::read_csv("Shill Bidding Dataset.csv")
```

```r
shill_bidding_orig <- read.csv("Shill Bidding Dataset.csv")
# dim(shill_bidding_orig)
# names(shill_bidding_orig)
# kable(head(shill_bidding_orig, 10))

# remove first three columns (ID variables)
shill_bidding <- shill_bidding_orig %>%
  dplyr::select(-c(Record_ID,Auction_ID, Bidder_ID)) %>%
  # Change the value 0 to -1 for Class
  mutate(Class = ifelse(Class == 0, -1, 1))

# confirm the conversion was done correctly
dim(shill_bidding)
```

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

```r
table(shill_bidding_orig$Class)
```

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

```r
table(shill_bidding$Class)
```

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

After the initial preprocessing, the data set that we will be using in this project consist of 6321 observations and 10 variables.

```r
shill_bidding %>%
  slice_sample(n=10) %>%
  kable(booktabs=T, linesep="", align = "c",
        caption = "10 random samples from the data after initial cleaning") %>%
  kable_styling(latex_options = c("scale_down", "HOLD_position")) %>%
  kable_classic()
```

Table 2: 10 random samples from the data after initial cleaning

Bidder_Tendency	Bidding_Ratio	Successive_Outbidding	Last_Bidding	Auction_Bids	Starting_Price_Average	Early_Bidding	Winning_Ratio	Auction_Duration	Class
0.0084	0.0238	0	0.5586	0.5714	0.0000	0.5586	0.0000	3	-1
0.5000	0.1818	0	0.1113	0.0000	0.0000	0.0404	0.5000	7	-1
0.0500	0.1000	0	0.1872	0.0000	0.0000	0.1872	0.9167	7	-1
0.0526	0.0172	0	0.2852	0.6897	0.9936	0.2852	0.0000	1	-1
0.2353	0.2500	1	0.3756	0.0000	0.0000	0.3756	1.0000	7	1
0.0256	0.0159	0	0.7437	0.7143	0.9936	0.7437	0.0000	7	-1
0.1000	0.0323	0	0.0130	0.4194	0.9936	0.0130	0.0000	7	-1
0.0082	0.0476	0	0.0228	0.1429	0.9936	0.0228	0.0000	7	-1
0.0164	0.0800	0	0.2579	0.2800	0.9936	0.0510	0.0000	1	-1
0.0357	0.0714	0	0.9817	0.0000	0.0000	0.9817	0.0000	1	-1

2 Exploratory Data Analysis

2.1 Part (a): Investigating distinct levels or values for each variable

```
# cols <- 1:NCOL(shill_bidding)
# for (j in cols){
#   x <- shill_bidding[,j]
#   print(names(shill_bidding)[j])
#   # print(sort(unique(x, incomparables=TRUE)))
#   print(table(x, useNA="ifany"))
# }


output <- NULL
roles <- c(rep("Predictor", 9), "Target")
for(i in seq_along(shill_bidding)) {
  output <- rbind(output, c(names(shill_bidding)[i],
                  class(shill_bidding[[i]]),
                  roles[i],
                  length(unique(shill_bidding[[i]])))
                  )
}

as.data.frame(output) %>%
  kable(booktabs=T, linesep="", align = "lccc",
        col.names = c("Variable name", "Data type", "Role", "Number of distinct values"))%>%
  kable_styling(latex_options = c("HOLD_position")) %>%
  kable_classic()
```

Variable name	Data type	Role	Number of distinct values
Bidder_Tendency	numeric	Predictor	489
Bidding_Ratio	numeric	Predictor	400
Successive_Outbidding	numeric	Predictor	3
Last_Bidding	numeric	Predictor	5807
Auction_Bids	numeric	Predictor	49
Starting_Price_Average	numeric	Predictor	22
Early_Bidding	numeric	Predictor	5690
Winning_Ratio	numeric	Predictor	72
Auction_Duration	integer	Predictor	5
Class	numeric	Target	2

- The variables `Class`, `Successive Outbidding`, and `Auction Duration` have fewer distinct values.

2.2 Part (b): Checking for missing data

```
# inspecting missing values using the "naniar" package
naniar::miss_var_summary(shill_bidding) %>%
  kable(booktabs = T, linesep="", align = "lcc",
        col.names = c("Variable", "Number missing", "Percent missing"),
        cap = "Amount of missing values in the shill bidding dataset") %>%
  kable_styling(latex_options = c("HOLD_position")) %>%
  kable_classic()
```

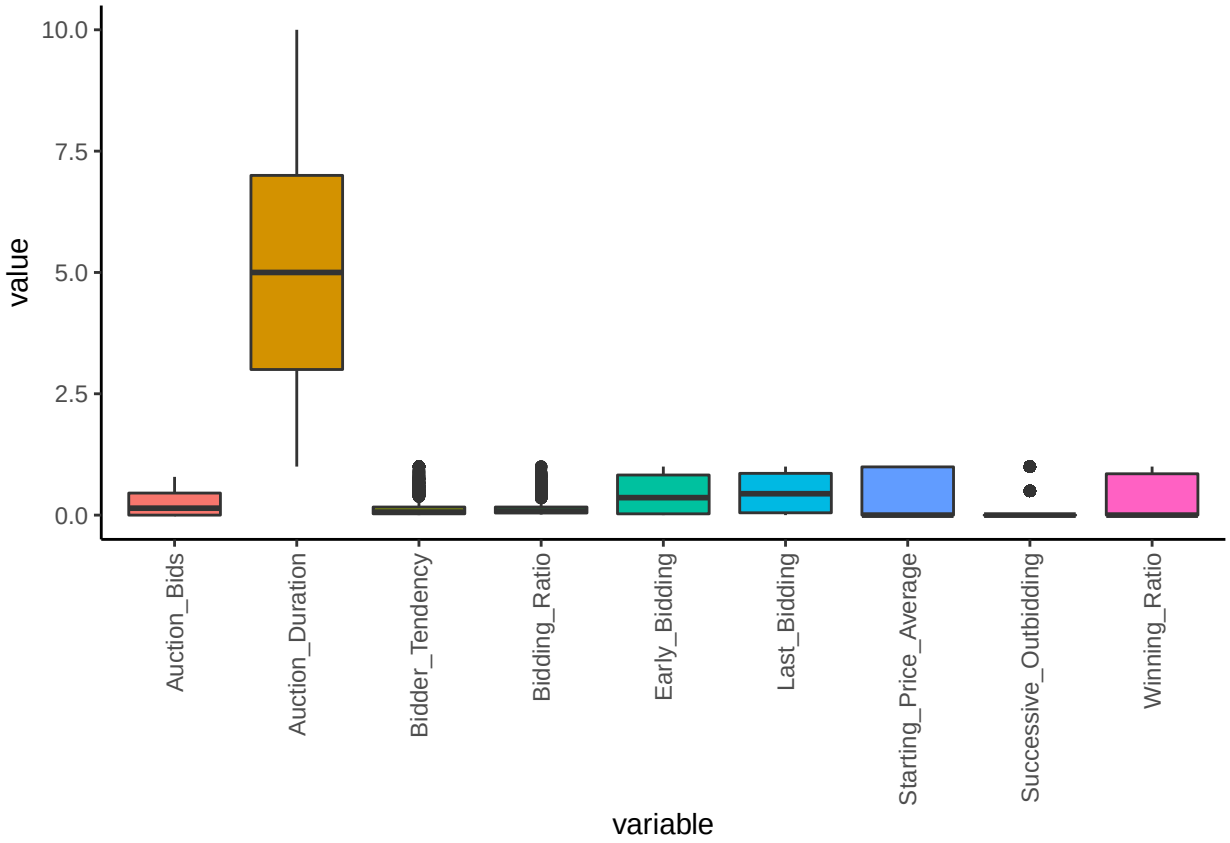
Table 3: Amount of missing values in the shill bidding dataset

Variable	Number missing	Percent 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

Clearly, the above output indicates that the data has no missing values.

2.3 Part (c): Parallel boxplots

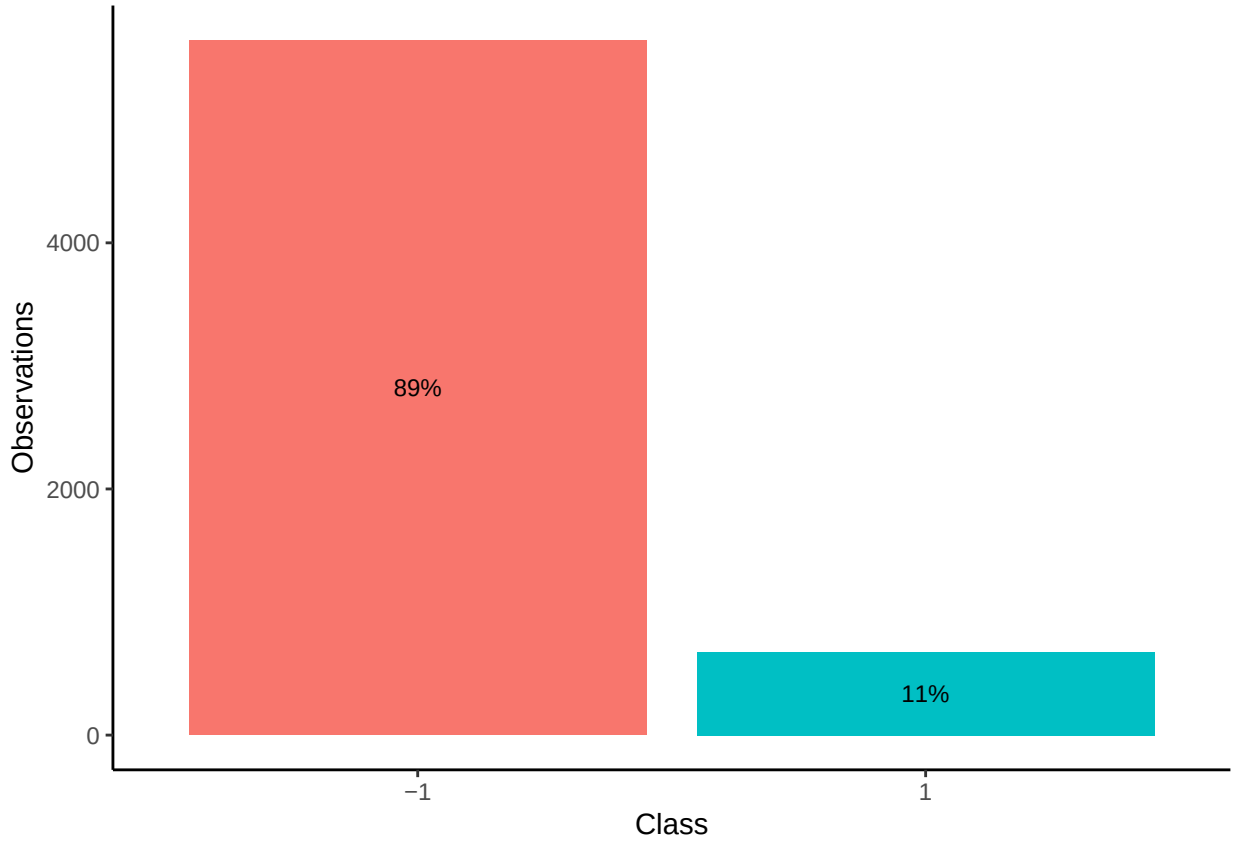
```
shill_bidding %>%
  tidyr::pivot_longer(-Class,names_to="variable", values_to="value") %>%
  ggplot(aes(variable, value, fill=variable)) +
  geom_boxplot() +
  theme(legend.position = "none",
        axis.text.x = element_text(angle = 90, vjust = .5, hjust = 1))
```



Judging from the sizes of the boxes and the minimum and maximum values for each variable, it is evident that **the predictors do not have the same range and variation**. Take for instance, the range for `Auction Duration` is about 9 and its IQR is about 4 compared to the rest of the predictors whose range and IQR do not come close. Interestingly, the 25% quantile of `Auction_Duration` is even higher that the 75% quantile of all the other attributes. It will therefore be necessary to scale the variables in our modeling process.

2.4 Part (d): A bar plot of the binary response `Class`

```
shill_bidding %>%
  mutate(Class = factor(Class)) %>%
  group_by(Class) %>%
  summarise(n=n()) %>%
  mutate(pct = n/sum(n),
         lbl = scales::percent(pct)) %>%
  ggplot(aes(Class, n, fill=Class)) +
  geom_bar(stat = "identity", position = "dodge",show.legend = F) +
  geom_text(aes(label = lbl), size=3, position = position_stack(vjust = 0.5))  +
  labs(y="Observations")
```




```
# geom_text(aes(label = scales::label_comma()(n)), size=2,
#                             position = position_stack(vjust = 1))
```

The -1 class has 78% more observations than the 1 class, signifying a marked difference between the two classes. However, this difference does not seem to present us with an unbalanced classification problem since our sample size (6321) is relatively large.

3 Data Partition

The resulting data is therefore partitioned as follows. A ***125*** seed was used throughout to ensure reproducibility of results affected by random generations.

```
set.seed(125) # set seed for reproducibility
n <- nrow(shill_bidding)
split_id <- sample(1:3, size = n, prob = c(2,1,1)/4, replace = T)
# train_index <- sample(1:n, size=trunc(n*train_ratio), replace=FALSE)
# rem_index <- (setdiff(1:n, train_index))
# val_index <- sample(rem_index, size=trunc(length(rem_index)*0.5), replace=FALSE)
# test_index <- setdiff(rem_index, val_index)


train_set <- shill_bidding[split_id==1, ] # training data D1
validation_set <- shill_bidding[split_id==2, ] # validation data D2
test_set <- shill_bidding[split_id==3, ] # test data D3
```

```r
dim(train_set); dim(validation_set); dim(test_set)
```

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

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

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

Using the ratio 2:1:1, the shill bidding data set was partitioned into 3171 training observations, 1596 validation observations and 1554 test observations, respectively, all with 10 variables as expected.

4 Logistic Regression - Optimization

In this section, we first apply optimization techniques to implement the logistic regression model by directly minimizing the negative log-likelihood function. Next, we compare our results to those obtained from the standard R `glm()` function and finally evaluate the performance of the model we implemented manually on a test data.

4.1 Part (a)

4.1.1 Obtaining the maximum likelihood estimates of regression parameters

Here, we used the `optim()` function in R to minimize the negative log-likelihood function as a means of obtaining estimates for the logistic regression parameters. The **BFGS** optimization algorithm was specified and the parameter estimates were all initialized at 0. The table below presents the results obtained.

```r
# merge training and validation data
train_valid_set <- dplyr::bind_rows(train_set, validation_set)
dim(train_valid_set)
```

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

```r
# prepare the data
X <- train_valid_set %>% dplyr::select(-Class)
X <- as.matrix(X)
y <- train_valid_set$Class
p <- ncol(X)

# The negative loglikelihood function for Y=+1/-1
nloglik <- function(beta, X, y){
    if (length(unique(y)) !=2) stop("The target y must be binary!")
    X <- cbind(1, X)
    nloglik <- sum(log(1+ exp(-y*X%*%beta)))
    return(nloglik)
}

# FIT LOG-LINEAR MODEL
b0 <- rep(0, (p+1))
optim_fit <- optim(par = b0, fn=nloglik, y=y, X=X, method = "BFGS", hessian = TRUE)
```

```r
beta.hat <- optim_fit$par

# get the standard errors and corresponding p-values for each parameter
VCOV.est <- solve(optim_fit$hessian) # the variance covariance matrix
se <- sqrt(diag(VCOV.est))
z.wald <- beta.hat/se
pvalue <- pchisq(z.wald^2, df=1, lower.tail=FALSE)

# display results
result <- data.frame(estimate=beta.hat, se, z.wald, pvalue)
row.names(result) <- c("Intercept", names(train_valid_set)[-10])
round(result, digits = 4) %>%
    kable(booktabs = T, linesep="",
          cap = "Regression parameters with standard errors and corresponding p-values") %>%
   kable_styling(latex_options =c("HOLD_position"))%>%
   kable_classic()
```

Table 4: Regression parameters with standard errors and corresponding p-values

	estimate	se	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

- At 5% significance level, the p-values suggest that only three predictors, `Bidder Tendency`, `Successive Outbidding` and `Winning Ratio` appear to be statistically significant.

4.2 Part (b): Comparing results to the standard R function `glm()`

```r
# fit a logistic model via glm()
D1D2_new <- train_valid_set %>% mutate(Class = ifelse(Class == -1,0,1))

fit <- glm(Class ~., data=D1D2_new, family = "binomial")

# extract and display output
glm_estimates <- summary(fit)$coeff

kable(glm_estimates, booktabs = T, linesep="",
          cap = "") %>%
```

```
  kable_styling(latex_options =c("HOLD_position"))%>%
  kable_classic()
```

Table 5:

	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

- Interestingly, using the standard R function `glm()` to train a logistic regression model on the same data yielded identical results compared to the logistic model implemented with the `optim()` function via maximum likelihood estimation. The results are approximately the same. In fact, judging from the p-values, the same predictors, `Bidder Tendency`, `Successive Outbidding` and `Winning Ratio`, as obtained in 4(a) appear to be the significant predictors when a threshold of 5% is used.
- By looking at the coefficients, it can be said that all the predictors, with the exception of `Early Bidding`, are positively associated with the target `Class` (they increase the log odds of being in the positive class). For example, the coefficient for `Bidder Tendency` is **1.0539** which means that a unit increase in `Bidder Tendency` increases the odds of being in the positive class by approximately 186.9% (exp(1.0539)=2.869, 2.869-1=1.869) more than being in the negative class, holding all other predictors fixed.

4.2.1 Checking for convergence

```
optim_fit$convergence
```

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

The output of 0 indicates that the optimization algorithm converged.

4.3 Part (c): Evaluating the trained logistic model in 4(a) on the test data

For the purpose of computing prediction accuracy, we obtain predictions based on the following formula:

$$\hat{y}' = sgn\left[\frac{exp(X'\hat{\beta})}{1+exp(X'\hat{\beta})} - 0.5\right], \qquad (1)$$

where X' and $\hat{\beta}$ denote the design matrix from the test data with additional first column of 1's, and a vector of estimated regression coefficients, respectively.

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

Xprime <- as.matrix(cbind(1, test_set[, -NCOL(test_set)]))
predicted_y <- sign(expit(Xprime%*%optim_fit$par)-0.5) # a vector of +-1's
observed_y <- test_set$Class
conf_mat <- table(observed_y, predicted_y); conf_mat
```

```
##           predicted_y
## observed_y   -1    1
##         -1 1380   23
##         1    10  141
```

```
(logit.pred_acc <- sum(diag(conf_mat))/sum(conf_mat))
```

```
## [1] 0.9788
```

Out of the 1554 observations in the test data, **1521** were correctly classified leading to a high prediction accuracy of 97.88%.

5 Primitive LDA - The Kernel Trick

We implement the primitive LDA (linear discriminant analysis) classifier below

$$\hat{y} = sgn\{(m_+ - m_-)^T z - (m_+ - m_-)^T m\} = m_+^T z - m_-^T z + \frac{m_-^T m_- - m_+^T m_+}{2}, \qquad (2)$$

where all the four terms, $m_+^T z$, $m_-^T z$, $m_-^T m_-$, and $m_+^T m_+$, are computed using the kernel trick procedure.

5.1 Part (a): Standardizing the predictor matrices

Letting X_1 and X_2 to denote the matrix of all predictors corresponding to the training data and the validation data, respectively, we first scale X_1 according to its column means and SDs and later scale X_2 according to the column means and SDs computed from X_1 as follows.

```
# obtain the predictor matrices
ncols <- NCOL(shill_bidding)
X1 <- as.matrix(train_set[-ncols])
X2 <- as.matrix(validation_set[-ncols])
X3 <- as.matrix(test_set[-ncols])

# standardize X1 and X2
scaledX1 <- scale(X1)
X1_col_means <- attributes(scaledX1)$`scaled:center`
X1_col_sds <- attributes(scaledX1)$`scaled:scale`
scaledX2 <- scale(X2, center = X1_col_means, scale = X1_col_sds)
```

5.2 Part (b): Training the LDA-P classifier with a polynomial kernel family

For convenience, the LDA classifier depicted in equation (2) is implemented in a function called `kernLDA` for use throughout the rest of the project.

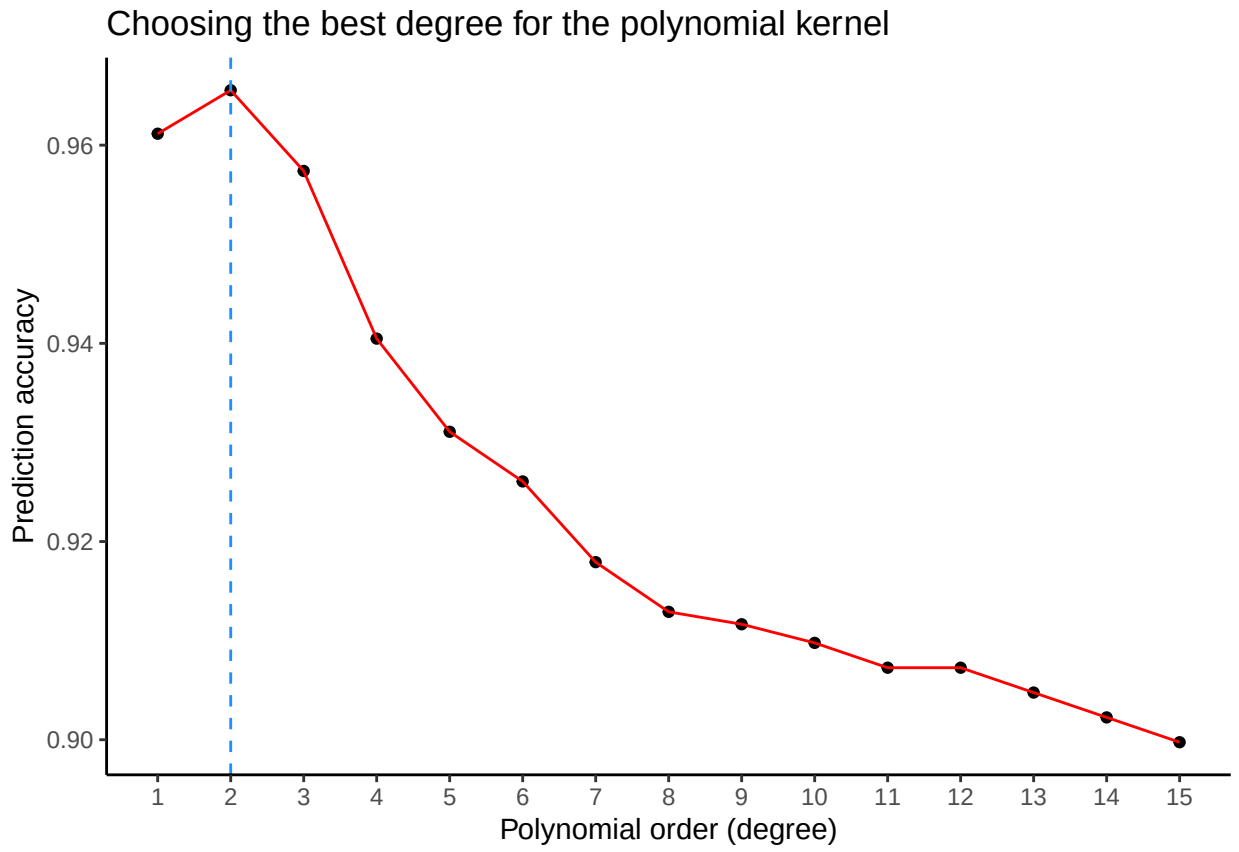
In kernel methods, the choice of a kernel function and the choices of its parameters play an important role on the outcome of a given learning task. For this project, we chose a ***polynomial kernel family*** as implemented in the **kernlab** R package. A validation technique was then employed to determine the optimal choice of the degree parameter ranging from **1 to 15**, while leaving the scale and offset parameters at their default values of **1**.

```
#' Primitive LDA via the "kernel trick"
#'
#' @param kernel the kernel function to be used to calculate the kernel matrix.
#' @param data1 a data matrix of scaled features from train set to be used to calculate
#` the kernel matrix.
#' @param data2 second data matrix of scaled features to calculate the kernel matrix,
#` either validation data for parameter tuning or test data for predictions.
#' @param target
#'
#' @return a list containing a vector of predictions, the constant b, and
#` the vector w.z.
#`
kernLDA <- function (kernel, data1, data2=NULL, target) {

    if(!all(unique(target) %in% c(-1,1))) stop("Please use the plus 1 minu 1 class
                                        coding for the target.")
    # compute the terms in equation 1 after expanding
    kernmat <- kernlab::kernelMatrix
    term1 <- colMeans(kernmat(kernel, x=data1[target==1,], y=data2))
    term2 <- colMeans(kernmat(kernel, x=data1[target==-1,], y=data2))
    term3 <- mean(kernmat(kernel, data1[target==-1,]))
    term4 <- mean(kernmat(kernel, data1[target==1,]))

    # assemble all terms to obtain the predicted y
    w.z <- (term1 - term2)
    b <- (term3 - term4)/2
    yhat <- sign(w.z + b)
    return(list(yhat=yhat, w.z=w.z, b=b))
}

#-----
deg_vec <- 1:15
pred.acc_vec <- vector("numeric", length(deg_vec))
for (i in seq_along(deg_vec)) {
    # choose the polynomial kernel family and tune the degree parameter, leaving
    # the scale and offset parameters at their defaults of 1.
    d <- deg_vec[i]
    kern <- kernlab::polydot(degree = d)
```

```
    # train and validate the primitive LDA classifier
    mod_LDA <- kernLDA(kernel=kern, data1=scaledX1, data2=scaledX2, target=train_set$Class)

    #compute prediction accuracy
    ypred <- mod_LDA$yhat
    yobserved <- validation_set$Class
    conf_mat <- table(ypred, yobserved)
    pred_accuracy <- sum(diag(conf_mat))/sum(conf_mat)
    pred.acc_vec[i] <- pred_accuracy
}

summary(pred.acc_vec)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.900   0.907   0.913   0.924   0.936   0.966
```

```
best.d <- deg_vec[which.max(pred.acc_vec)] # get the best degree

# plot the prediction accuracy
data.frame(d=deg_vec, pred_accuracy=pred.acc_vec) %>%
    ggplot(aes(d, pred_accuracy)) +
    geom_point() +
    geom_line(color = "red") +
    scale_x_continuous(breaks = deg_vec) +
    geom_vline(xintercept = best.d, linetype="dashed", color="dodgerblue") +
    labs(x="Polynomial order (degree)", y="Prediction accuracy",
         title = "Choosing the best degree for the polynomial kernel")
```



From the above plot, the optimal degree for the polynomial kernel occurred at 2 with a prediction accuracy of 96.55%. Thus, the best classifier among the polynomial kernel family considered corresponds to an inhomogeneous (positive offset of 1) quadratic kernel.

5.3 Part (c): Applying the best model to the test data

```
# first pool the validation and training sets together
Dprime <- dplyr::bind_rows(train_set, validation_set)
# obtain the scaled predictor matrices
Xprime <- as.matrix(Dprime[-NCOL(Dprime)])
scaledXprime <- scale(Xprime)
scaledX3 <- scale(X3, attributes(scaledXprime)$`scaled:center`,
                  attributes(scaledXprime)$`scaled:scale`)

# apply the best trained classifier to the test data D3
kern <- kernlab::polydot(degree = best.d)
ypred <- kernLDA(kernel = kern, data1 = scaledXprime, data2 = scaledX3,
                 target = Dprime$Class)$yhat
yobserved <- test_set$Class
conf_mat <- table(ypred, yobserved)
(lda.pred_acc <- sum(diag(conf_mat))/sum(conf_mat))
```

```
## [1] 0.973
```

The prediction accuracy on the test set is 97.3%, signifying a satisfactory performance.

5.3.1 Comparing the performance of the primitive LDA to the logistic regression

```
data.frame(lda.pred_acc, logit.pred_acc) %>%
    kable(booktabs=T, align = "c", col.names = c("LDA-P (quadratic kernel)", "Logistic
                                                  Reg. (optimization)")) %>%
    add_header_above(header = c("Prediction Accuracy"=2)) %>%
    kable_styling(latex_options = c("HOLD_position")) %>%
    kable_classic()
```

Prediction Accuracy	
LDA-P (quadratic kernel)	Logistic Reg. (optimization)
0.973	0.9788

From the above table, with prediction accuracy of over 97%, it is clear that both models did very well on the test data, with the logistic regression model in part 4(c) however performing slightly better than the primitive LDA via a quadratic kernel function. These results suggest that the polynomial kernel appears to be a good choice for this classification problem.

Reference

- Data retrieved from: http://archive.ics.uci.edu/ml/datasets/Shill+Bidding+Dataset