Matthew Bolding
Dr. Nelis Potgieter
MATH 40883–015
April 27th, 2022

Predictive Modeling: Project 2

1 Missing Data

First, we import the data, and we may take note of the number of observations.

```
dataset <- read.table("project2.data", header = T, sep = ";")
nrow(dataset)
```

```
## [1] 500
```

After removing all observations that have missing data via the `na.omit` command, we now have a dataset with 480 rows. We have lost 20 observations.

2 Partitioning Data

Using a function in the `caTools` library, we may split the data.

```
# Note we set a seed for repeatability.
set.seed(1984)

# We have a 70/30 split for training and validation data, respectively.
index <- sample.split(dataset$y, SplitRatio = 0.7)
train <- subset(dataset, index == TRUE)
valid <- subset(dataset, index == FALSE)
```

3 Data Visualizations

Before visualizing any data, let us first calculate the means and standard deviations for all variables $x_1, \ldots, x_5$ while also noting successes and failures across the observations. We may find all successes and failures in the following way.

```
success.index <- which(train$y > 0.5)
success <- train[success.index, ]
failure <- train[-success.index, ]
```

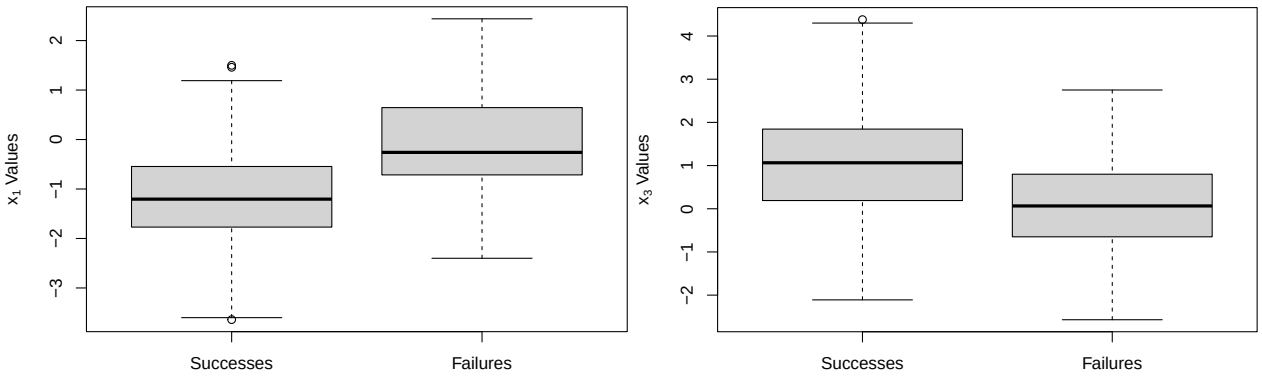
We may then calculate the means and standard deviations for each variable for both outcomes with the `mean()` and `sd()` commands. (These commands are hidden for brevity—see the attached `.r` file all commands.) With such information, we may construct a table to easily view interesting trends that may exist.

	Mean		**Standard Deviation**	
Variable	*Successes*	*Failures*	*Successes*	*Failures*
x_1	-1.09839300	-0.06809524	0.9917071	1.0016350
x_2	0.01166667	0.01559524	1.2673250	0.9951720
x_3	1.07101200	0.06916667	1.2206070	0.9679699
x_4	0.07684524	-0.05625000	0.9197338	1.0410630
x_5	-0.05904762	0.03136905	0.9056451	1.0404580

Observe that the means for x_1 and x_3 across successes and failures compared to other variables differ by at least 1.0. Further note the correlation between x_1 and x_3 to the output variable y. `cor(train$x1, train$y) = -0.4602401` and `cor(train$x3, train$y) = 0.4149753`.

These relatively high correlations combined with means differing between successes and failures gives us the motivation to visualize x_1 and x_3 values, differentiating between successes and failures. The below does visualizes such for x_1 and y. The associated `.r` file contains the other graph's code.

```
boxplot(success$x1, failure$x1,
        ylab = expression('x'[1] * " Values"),
        names = c("Successes", "Failures"))
```

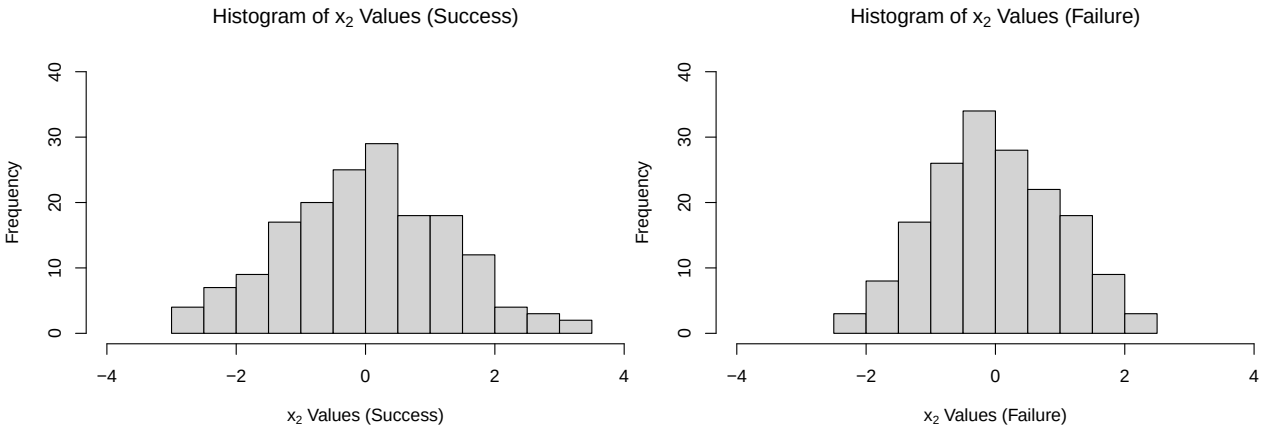


Observe that these box plots show some degree of separation, so linear logistic regression would likely work well—especially when using input variables x_1 and x_3—to predict whether a bank account defaulted in the last three months.

On one hand, variables x_2, x_4, and x_5 would likely contribute very little to the predictive ability of the model trained from linear predictors, as their means are so similar between successes and failures; yet on another, recall that, within the context of quadratic predictors, differing standard deviations between successes and predictors might improve a model's predictive ability.

Observe that, across successes and failures, the standard deviations of both x_2 and x_3 differ by about a quarter, x_4 and x_5 differ by about an eight, and x_1 by about a hundredth. Therefore, since we have found that variables x_2, x_3 have the greatest differences between standard deviations across outcomes, we might expect these predictors, added as quadratic terms to the logistic model, to improve the predictive ability most. We might also expect x_1 as a quadratic predictor to contribute very little to the model's performance.

For the sake of this section, we may also construct two histograms to display the differing standard deviations of `success$x2` and `failure$x2`. As the graph confirms, we may see that the x_2 predictor's standard deviation is different when considering the two outcomes! (See the `.r` file for the code.)



4 Prediction Models

Before starting the training process, we may initialize our environment.

```
n <- nrow(train)
folds <- sample(rep(1:8, length = n))
```

4.1 Logistic Regression with Linear Terms

We may train a model with only linear terms with logistic regression.

```
Linear.MSE <- NULL

# We iterate 8 times to implement 8-fold cross-validation.
for(k in 1:8) {
  # Make the train and validation set.
  sub.train <- train[folds!=k,]
  sub.valid <- train[folds==k,]

  # Train the mode.
  logistic.mod <- glm(y ~ ., data = sub.train, family = binomial)

  # Calculate y hats.
  logistic.pred <- predict(logistic.mod, newdata = sub.valid, type = "response")

  # Calculate the MSE for this pass.
  Linear.MSE[k] <- mean((logistic.pred - sub.valid$y)^2)
}
Linear.CV <- mean(Linear.MSE)
Linear.CV
```

```
## [1] 0.1502794
```

4.2 Logistic Regression with Linear and Quadratic Terms

We may also train a model with both linear and quadratic terms with logistic regression. Before this, however, we must create the quadratic terms.

```
quad.train <- train
quad.train$x1.2 <- quad.train$x1^2
quad.train$x2.2 <- quad.train$x2^2
quad.train$x3.2 <- quad.train$x3^2
quad.train$x4.2 <- quad.train$x4^2
quad.train$x5.2 <- quad.train$x5^2
```

Train the model.

```
# We proceed very similarly to the linear logistic regression.
Linear.Quad.MSE <- NULL
for(k in 1:8) {
  sub.train <- quad.train[folds!=k,]
  sub.valid <- quad.train[folds==k,]

  logistic.mod <- glm(y ~ ., data = sub.train, family = binomial)
  logistic.pred <- predict(logistic.mod, newdata = sub.valid, type = "response")
  Linear.Quad.MSE[k] <- mean((logistic.pred - sub.valid$y)^2)
}
```

```
Linear.Quad.CV <- mean(Linear.Quad.MSE)
Linear.Quad.CV
```

```
## [1] 0.1337652
```

5 Fitting the Best Model

Observe that the cross-validation score for the logistic model with both linear and quadratic terms (0.1337652) is less than the one with only linear terms (0.1502794), so we may conclude that the model with both linear and quadratic preforms better.

Then, let us train a logistic model with with the appropriate dataset, i.e., the one with quadratic terms, and predict values from the validation data. Prior to predicting values, however, we must add the quadratic terms to the presently linear `valid` dataset. We do this similarly as in **Section 4.2**. We make a copy of `valid`, named `quad.valid` and add the quadratic terms.

```
best.mod <- glm(y ~ ., data = quad.train, family = binomial)
best.pred <- predict(best.mod, newdata = quad.valid, type = "response")

# Categorize the predictions.
best.pred <- ifelse(best.pred > 0.5, 1, 0)
```

With these predictions, we may calculate the overall mis-classification rate, the false-positive rate, and the false-negative rate. First, we may create the confusion matrix and calculate the other values.

```
Conf.Matrix <- table(Predicted = best.pred, True = valid$y)

Mis.Class.Rate <- 1 - (sum(diag(Conf.Matrix))/sum(Conf.Matrix))
False.Neg <- Conf.Matrix[2]/(Conf.Matrix[1] + Conf.Matrix[2])
False.Pos <- Conf.Matrix[4]/(Conf.Matrix[3] + Conf.Matrix[4])
```

We may view these values, computed from the confusion matrix.

```
##           True
## Predicted  0  1
##         0 56 22
##         1 16 50
```

```
## [1] "Overall Mis-classification Rate: 0.264"
```

```
## [1] "False Negative Rate:             0.222"
```

```
## [1] "False Positive Rate:             0.694"
```

The false negative rate is similar to the overall mis-classification rate, however, the false positive rate is much larger than that figure! As statisticians analyzing the data, we might not know the direct effect of having a high false positive rate within the bank's application. This large value may or may not be of concern to the bank, but it's certainly worth pointing out.