CASE STUDY REPORT

Group No.: Group-11 (Section-2)

Student Names: Prerit Samria and Seungyeon Ko

Executive Summary

The goal of the case study is to find the best model to predict whether a patient has heart disease or not based on their medical records. There are thousands of deaths every year around the world due to heart diseases. Thus, a model to predict the presence of heart disease would be beneficial to save many lives. Similar methods could be employed to predict other prevalent diseases in the world. The data 'Heart Disease Data Set' has been taken from the University of California Irvine. After preprocessing the data, 5 models, namely k-NN, Naïve Bayes Classifier, Classification Trees, Logistic Regression, and Neural Network, were used. The best performance in terms of accuracy on validation data is achieved by Neural Network.

I. Background and Introduction

The cardiovascular (or heart) disease is a prevalent disease that causes thousands of deaths every year around the world. In general, cardiovascular disease results from the narrowed or blocked blood vessels that can lead to a heart attack, chest pain or stroke. Other heart conditions, such as those that affect your heart's muscle, valves or rhythm, also are considered forms of heart disease. However, many causes of death are still not identified. The goal of this case study is to find the best method to predict whether a patient has heart disease or not based on their medical records. Similarly, different methods could be employed to predict other prevalent diseases in the world.

II. Data Exploration and Visualization

```
Classes 'tbl_df', 'tbl' and 'data.frame':       303 obs. of  14 variables:
 $ Age     : num  63 67 67 37 41 56 62 57 63 53 ...
 $ Sex     : chr  "male" "male" "male" "male" ...
 $ cp      : chr  "angina" "asympt" "asympt" "notang" ...
 $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : num  233 286 229 250 204 236 268 354 254 203 ...
 $ fbs     : chr  "TRUE" "fal" "fal" "fal" ...
 $ restecg : chr  "hyp" "hyp" "hyp" "norm" ...
 $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
 $ exang   : chr  "fal" "TRUE" "TRUE" "fal" ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ Slope   : chr  "down" "flat" "flat" "down" ...
 $ Ca      : chr  "0" "3" "2" "0" ...
 $ thal    : chr  "fix" "norm" "rev" "norm" ...
 $ Target  : chr  "buff" "sick" "sick" "buff" ...
```

Figure-1: Raw data set

The dataset comprises of 13 predictors and 1 outcome variable. The outcome target is a binary variable: 0 means a person who does not have heart disease, and 1 means a person who has heart disease. Figure-1 and table-1 give basic information about the dataset.

Table-1: Classifying the predictors

Age (in years)	Numerical	N
Sex (male:1, female:0)	Categorical	C
Chest Pain Type (CP) (0,1,2,3)	Categorical	C
Resting blood pressure (trestbps)	Numerical	N
Serum cholesterol in mg/dl (chol)	Numerical	N
Fasting blood sugar(fbs) (fbs>120: 1, else:0)	Categorical	C
Resting electrocardiographic (restecg) (hyp: 0, norm: 1, abn: 2)	Categorical	C
Maximum heart rate achieved (thalach)	Numerical	N
Exercise-induced angina (exang) (true:1, else:0)	Categorical	C
ST depression (oldpeak)	Numerical	N
Slope of the peak exercise ST segment (up: 2, flat:1, down: 0)	Categorical	C
Ca number of major vessels (0-3)	Categorical	C
Thal (2:normal, 1:fixed defect, 3:reversable defect)	Categorical	C
Output: Target (buff: 0: not having heart disease, sick: 1: having heart disease)	Categorical	C

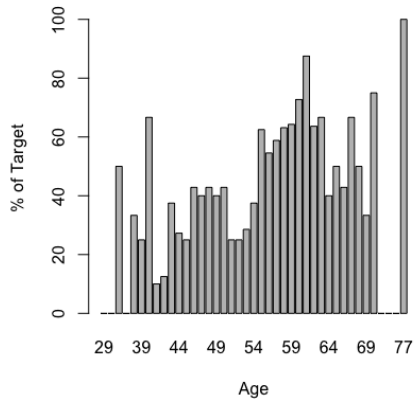

Figure-2: % Target vs Age

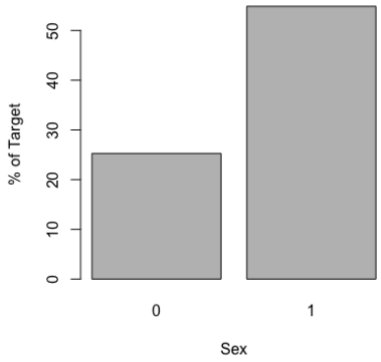

Figure-3: % Target vs Sex

Figure-2 indicates that the majority of the population gets heart disease after the age of 50. Figure-3 indicates that males have a higher chance of having a heart disease.

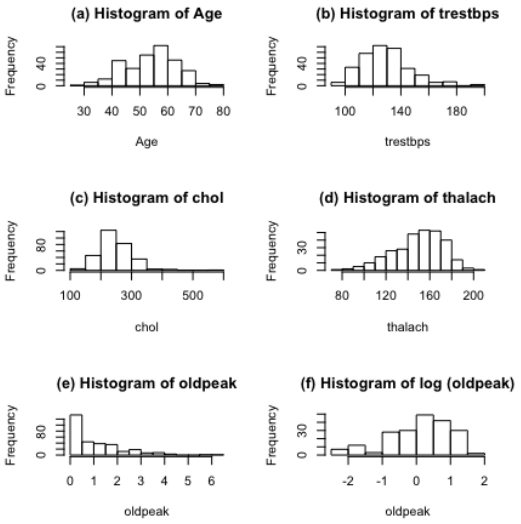

Figure-4: Histogram of the numerical variables

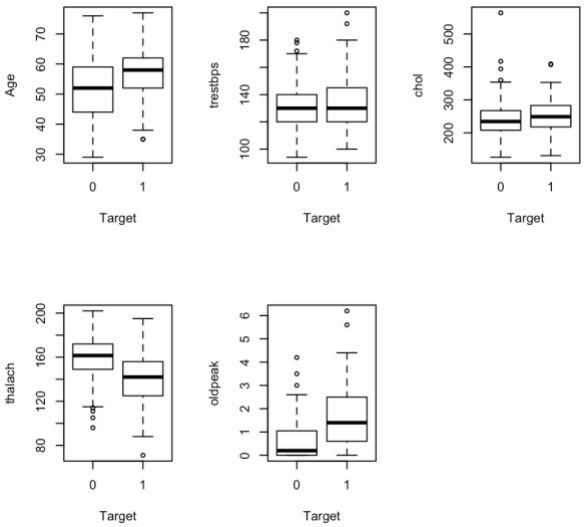

Figure-5: Boxplot of numerical variables

In figure-4 (a), the histogram of age seems to follow a normal distribution. In figure-4 (b), the histogram of trestbps is slightly right-skewed. In figure-4 (e), the histogram of oldpeak is highly left-skewed. Thus, it is converted using a log transformation.

In figure-5, for the boxplot of age, there is 1 outlier for someone with heart disease with age < 40 years. However, this does not seem like an error since the person might have heart disease due to some other medical conditions. Thus, we are not omitting any outliers from the dataset. In the boxplot of oldpeak, there is a higher probability that oldpeak can cause heart disease than other numerical variables. This is because oldpeak has a bigger difference between 0 and 1 target variables.

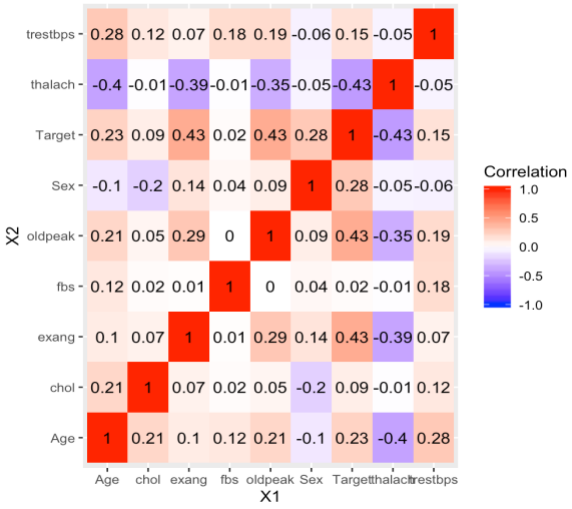

Figure-6: Heatmap for correlation of numerical variables with target

Figure-6 is used for correlation analysis to detect duplication of variables. The heatmap indicates that the target variable has a little stronger correlation with thalach, oldpeak and exang than other variables, and oldpeak is correlated with thalach. We observe that none of the variables are highly correlated with the 'Target' variable. This means that there is some underlying structure or a combination of predictors that leads to the outcome variable 'Target'. Thus, we are going to use all the predictors for the model.

III. Data Preparation and Preprocessing

```
Classes 'tbl_df', 'tbl' and 'data.frame':       301 obs. of  14 variables:
 $ Age     : num  63 67 67 37 41 56 62 57 63 53 ...
 $ Sex     : num  1 1 1 1 0 1 0 0 1 1 ...
 $ cp      : Factor w/ 4 levels "1","3","0","2": 2 3 3 4 1 1 3 3 3 3 ...
 $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : num  233 286 229 250 204 236 268 354 254 203 ...
 $ fbs     : num  1 0 0 0 0 0 0 0 0 1 ...
 $ restecg : Factor w/ 3 levels "2","0","1": 2 2 2 3 2 3 2 3 2 2 ...
 $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
 $ exang   : num  0 1 1 0 0 0 0 1 0 1 ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ Slope   : Factor w/ 3 levels "0","1","2": 1 2 2 1 3 3 1 3 2 1 ...
 $ Ca      : Factor w/ 4 levels "0","1","2","3": 1 4 3 1 1 1 3 1 2 1 ...
 $ thal    : Factor w/ 3 levels "1","2","3": 1 2 3 2 2 2 2 2 3 3 ...
 $ Target  : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 2 1 2 2 ...
```

Figure-7: Preprocessed Data set

Missing Values

Categorical variable Ca has 5 missing values marked with '?'. These missing values are imputed using the median value. Also, the categorical variable thal has 2 missing values marked with '?'. These missing values are deleted from the dataset. After data preprocessing, the dataset contains 301 records of 13 predictors and 1 outcome.

Output variable 'Target' is a binary variable with 164 samples as '0' and 137 samples as '1'. Thus the data set is balanced. There is no need for oversampling for classification.

Since the case study does not have new data, data is partitioned into training, validation and test sets. Training data is used to train the models, then validation data is used to measure the performance of the. If normalization or standardization is needed for the model, the entire data set is normalized (or standardized) using the normalized values of the training data. This is done for adjusting values that are measured on different scales to a notionally common scale.

IV. Data Mining Techniques and Implementation

The data set is balanced since the proportion of the Target variable '1' is 45% and the Target variable '0' is 55%. For all models, a cutoff value of 0.5 has been used.

The outcome variable (Target) is categorical. Therefore, it is a classification problem. The different models employed are:

1. k-Nearest Neighbors (k-NN)
2. Naive Bayes Classifier
3. Classification Tree
   a. Default Tree
   b. Full-Grown Tree
   c. Pruned tree
   d. Random Forests
4. Logistic Regression
   a. Default Logistic Regression
   b. Stepwise Logistic Regression
5. Neural Networks

1. k-Nearest Neighbors

To use k-NN, all categorical predictors with m classes should have m dummies. So now we have 25 predictors and 1 outcome variable.

```
'data.frame':   301 obs. of  26 variables:
 $ Age     : num  63 67 67 37 41 56 62 57 63 53 ...
 $ Sex     : num  1 1 1 1 0 1 0 0 1 1 ...
 $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : num  233 286 229 250 204 236 268 354 254 203 ...
 $ fbs     : num  1 0 0 0 0 0 0 0 0 1 ...
 $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
 $ exang   : num  0 1 1 0 0 0 0 1 0 1 ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ cp0     : num  0 0 0 0 1 1 0 0 0 0 ...
 $ cp1     : num  1 0 0 0 0 0 0 0 0 0 ...
 $ cp2     : num  0 1 1 0 0 0 1 1 1 1 ...
 $ cp3     : num  0 0 0 1 0 0 0 0 0 0 ...
 $ restecg0: num  0 0 0 0 0 0 0 0 0 0 ...
 $ restecg1: num  1 1 1 0 1 0 1 0 1 1 ...
 $ restecg2: num  0 0 0 1 0 1 0 1 0 0 ...
 $ Slope0  : num  1 0 0 1 0 0 1 0 0 1 ...
 $ Slope1  : num  0 1 1 0 0 0 0 0 1 0 ...
 $ Slope2  : num  0 0 0 0 1 1 0 1 0 0 ...
 $ Ca0     : num  1 0 0 1 1 1 0 1 0 1 ...
 $ Ca1     : num  0 0 0 0 0 0 0 0 1 0 ...
 $ Ca2     : num  0 0 1 0 0 0 1 0 0 0 ...
 $ Ca3     : num  0 1 0 0 0 0 0 0 0 0 ...
 $ thal1   : num  1 0 0 0 0 0 0 0 0 0 ...
 $ thal2   : num  0 1 0 1 1 1 1 1 0 0 ...
 $ thal3   : num  0 0 1 0 0 0 0 0 1 1 ...
 $ Target  : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 2 1 1 2 2 ...
```

Figure-8: Structure of data set for k-NN

Dataset is partitioned into training (50%), validation (30%) and test (20%) sets. The partitioned data sets are normalized into a scale of -1 to 1 using the normalized values of the training data.

```
   k accuracy error
1  1    81.11 18.89
2  2    81.11 18.89
3  3    84.44 15.56
4  4    83.33 16.67
5  5    83.33 16.67
6  6    83.33 16.67
7  7    83.33 16.67
8  8    85.56 14.44
9  9    86.67 13.33
10 10   86.67 13.33
11 11   86.67 13.33
12 12   83.33 16.67
13 13   82.22 17.78
14 14   82.22 17.78
15 15   82.22 17.78
16 16   83.33 16.67
17 17   83.33 16.67
18 18   82.22 17.78
19 19   83.33 16.67
20 20   84.44 15.56
```

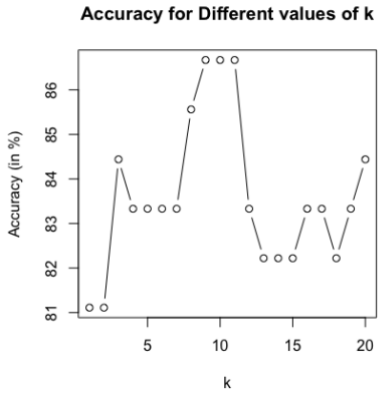

Figure-9: (a) Accuracy and Error on validation data for different values of k.

(b) Plot of Accuracy vs values of k

A low k value would result in overfitting. To reduce overfitting, we choose k such that the accuracy is still high. Using figure-9, the accuracy is highest for k = 9, 10 and 11. The second highest accuracy is for k = 8. Thus, we choose k = 8 whose accuracy is worse only to k = 9, which would result in overfitting.

```
          Reference
Prediction  0  1
         0 32  7
         1  3 19
```

Figure-10: Confusion Matrix for test data using k = 8

The accuracy on test data with k = 8 is 83.61%, which is quite high. The accuracy on validation data is 85.56%.

2. Naive Bayes Classifier

To use Naive Bayes, all predictors must be categorical. We need to convert all numerical predictors into categorical variables by binning into groups.

Age: Binning it in groups of 5 since people with a gap of 5 years should have similar medical records.
trestbps: Binning it in groups of 5.
chol: Binning it in groups of 10.
thalach: Binning it in groups of 5.
Oldpeak: No need to bin. Changing it into a factor.

```
'data.frame':   301 obs. of  14 variables:
 $ Age     : Factor w/ 10 levels "6","7","8","9",..: 8 8 8 2 3 6 7 6 8 6 ...
 $ Sex     : Factor w/ 2 levels "0","1": 2 2 2 2 1 2 1 1 2 2 ...
 $ cp      : Factor w/ 4 levels "1","3","0","2": 2 3 3 4 1 1 3 3 3 3 ...
 $ trestbps: Factor w/ 20 levels "19","20","21",..: 11 14 6 8 8 6 10 6 8 10 ...
 $ chol    : Factor w/ 28 levels "13","14","15",..: 11 17 11 13 8 12 15 23 13 8 ...
 $ fbs     : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 1 1 1 2 ...
 $ restecg : Factor w/ 3 levels "2","0","1": 2 2 2 3 2 3 2 3 2 2 ...
 $ thalach : Factor w/ 24 levels "14","18","19",..: 14 6 10 21 18 20 16 17 13 15 ...
 $ exang   : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 1 2 1 2 ...
 $ oldpeak : Factor w/ 6 levels "0","1","2","3",..: 3 3 4 5 2 2 5 2 2 4 ...
 $ Slope   : Factor w/ 3 levels "0","1","2": 1 2 2 1 3 3 1 3 2 1 ...
 $ Ca      : Factor w/ 4 levels "0","1","2","3": 1 4 3 1 1 1 3 1 2 1 ...
 $ thal    : Factor w/ 3 levels "1","2","3": 1 2 3 2 2 2 2 2 3 3 ...
 $ Target  : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 2 1 2 2 ...
```

Figure-11: Structure of data set for Naive Bayes

Data is partitioned into training (60%) and validation (40%) data.

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 92 10
         1  9 69

               Accuracy : 0.8944
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 53  8
         1 10 50

               Accuracy : 0.8512
```

Figure-12: Confusion Matrix for (a) training data and (b) validation data

From figure-12, accuracy on training data is 89.44% and on validation data is 85.12%.

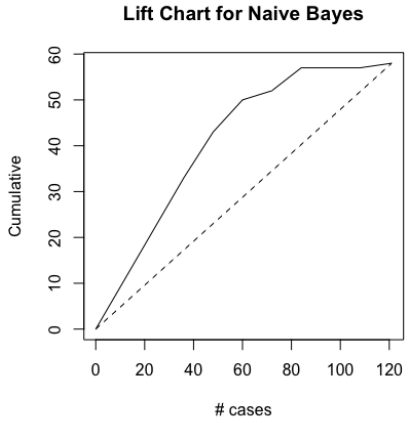

Figure-13: Lift Chart for Naive Bayes

From figure-13, we observe that our Naïve Bayes model is doing well in separating the important outcome (1) from outcome = 0.

3. Classification Tree

Variables Sex, fbs, and exang have binary values. Thus, we convert them into categorical predictors.

```
'data.frame':   301 obs. of  14 variables:
 $ Age     : num  63 67 67 37 41 56 62 57 63 53 ...
 $ Sex     : Factor w/ 2 levels "0","1": 2 2 2 2 1 2 1 1 2 2 ...
 $ cp      : Factor w/ 4 levels "1","3","0","2": 2 3 3 4 1 1 3 3 3 3 ...
 $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : num  233 286 229 250 204 236 268 354 254 203 ...
 $ fbs     : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 1 1 1 2 ...
 $ restecg : Factor w/ 3 levels "2","0","1": 2 2 2 3 2 3 2 3 2 2 ...
 $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
 $ exang   : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 1 2 1 2 ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ Slope   : Factor w/ 3 levels "0","1","2": 1 2 2 1 3 3 1 3 2 1 ...
 $ Ca      : Factor w/ 4 levels "0","1","2","3": 1 4 3 1 1 1 3 1 2 1 ...
 $ thal    : Factor w/ 3 levels "1","2","3": 1 2 3 2 2 2 2 2 3 3 ...
 $ Target  : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 2 1 2 2 ...
```

Figure-14: Structure of data set for Classification Tree

Data is partitioned the data into training (60%), validation (30%) and test (10%) sets.

(a) Default Tree

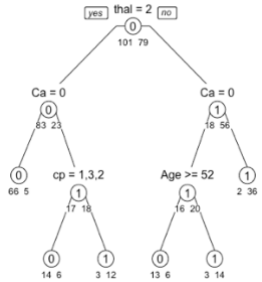

Figure-15: Default Tree with leaves = 6

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 93 17
         1  8 62

               Accuracy : 0.8611
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 41 13
         1  6 30

               Accuracy : 0.7889
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 15  2
         1  1 13

               Accuracy : 0.9032
```

Figure-16: Confusion Matrix for (a) training data, (b) validation data and (c) test data

From figure-16, we see that accuracy on the training data is 86.11%, on the validation data is 78.89% and on the test data is 90.32%.

(b) Full-Grown Tree

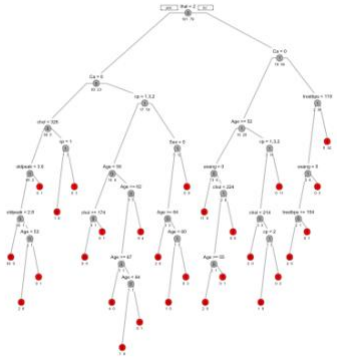

Figure-17: Full-grown Tree with leaves = 28

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 101   0
         1   0  79

      Accuracy : 1
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 34 13
         1 13 30

      Accuracy : 0.7111
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 14  2
         1  2 13

      Accuracy : 0.871
```

Figure-18: Confusion Matrix for (a) training data, (b) validation data and (c) test data

From figure-18, we see that accuracy on the training data is 100% since it is a full-grown tree. The accuracy on the validation data is 71.11%, which is quite low since a full-grown tree overfits the data. However, the accuracy on the test data is 87.1% which is higher than expected.

(c) Best-Pruned Tree

First, cross-validation is used to build a full-grown tree. Then, the tree with an error equal to the sum of the minimum error tree and its standard deviation is chosen from the cp table shown in figure-20.

Figure-19: Cross-Validation Tree with leaves = 28

	CP	nsplit	rel error	xerror	xstd
1	0.4810127	0	1.000000	1.00000	0.084277
2	0.0569620	1	0.518987	0.65823	0.076974
3	0.0443038	3	0.405063	0.62025	0.075591
4	0.0253165	6	0.265823	0.55696	0.072985
5	0.0126582	9	0.189873	0.51899	0.071226
6	0.0084388	20	0.050633	0.55696	0.072985
7	0.0063291	23	0.025316	0.53165	0.071829
8	0.0000100	27	0.000000	0.53165	0.071829

Figure-20: cp table for cross-validation

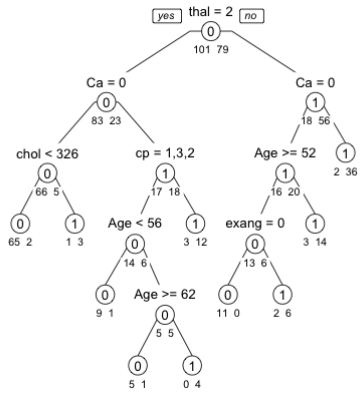

Figure-21: Best-pruned Tree with leaves = 10

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 90  4
         1 11 75

               Accuracy : 0.9167
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 35  8
         1 12 35

               Accuracy : 0.7778
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 14  2
         1  2 13

               Accuracy : 0.871
```

Figure-22: Confusion Matrix for (a) training data, (b) validation data and (c) test data

From figure-22, we see that the accuracy on the training data is 91.67%, on the validation data is 77.78%, and on the test data is 87.1%. The accuracy on validation data is higher than that for a full-grown tree.

(d) Random Forest

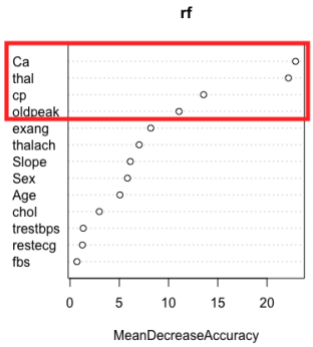

Figure-23: Important Variables for making the forest

From figure-23, we see that the 4 most important variables are Ca, thal, cp, and oldpeak.

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 100   1
         1   1  78

               Accuracy : 0.9889
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 44 11
         1  3 32

               Accuracy : 0.8444
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 16  2
         1  0 13

               Accuracy : 0.9355
```

Figure-24: Confusion Matrix for (a) training data, (b) validation data and (c) test data

From figure-22, we see that the accuracy on the training data is 98.89%, on the validation data is 84.44%, and on the test data is 93.55%.

Summarizing the accuracy on the data sets and the number of leaves for all the trees.

	Leaves	Accuracy_Training	Accuracy_Validation	Accuracy_Test
Default Tree	6	86.11	78.89	90.32
Full Grown Tree	28	100.00	71.11	87.10
Best-Pruned Tree	10	91.67	77.78	87.10
Random Forest	NA	98.89	84.44	93.55

Figure-25: Important Variables for making the forest

From figure-25, we see that among the 4 types of trees, the model with the highest accuracy on validation data is Random Forest.
If the tree size is big (i.e., if the number of leaves is higher), then there is a risk of overfitting the training data. For reducing error, pruning and random forest are used.

4. Logistic Regression

```
'data.frame':   301 obs. of  14 variables:
 $ Age     : num  63 67 67 37 41 56 62 57 63 53 ...
 $ Sex     : num  1 1 1 1 0 1 0 0 1 1 ...
 $ cp      : Factor w/ 4 levels "1","3","0","2": 2 3 3 4 1 1 3 3 3 3 ...
 $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : num  233 286 229 250 204 236 268 354 254 203 ...
 $ fbs     : num  1 0 0 0 0 0 0 0 0 1 ...
 $ restecg : Factor w/ 3 levels "2","0","1": 2 2 2 3 2 3 2 3 2 2 ...
 $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
 $ exang   : num  0 1 1 0 0 0 0 1 0 1 ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ Slope   : Factor w/ 3 levels "0","1","2": 1 2 2 1 3 3 1 3 2 1 ...
 $ Ca      : Factor w/ 4 levels "0","1","2","3": 1 4 3 1 1 1 3 1 2 1 ...
 $ thal    : Factor w/ 3 levels "1","2","3": 1 2 3 2 2 2 2 2 3 3 ...
 $ Target  : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 2 1 2 2 ...
```

Figure-26: Structure of data set for Logistic Regression

Data is partitioned into 60% training and 40% validation data.

(a) **Default Logistic Regression** (using all predictors)

The ‘glm’ function automatically takes ‘m-1’ dummies from m categories. 20 predictors including dummies are used.

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 94 11
         1  7 68

               Accuracy : 0.9
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 57 12
         1  6 46

               Accuracy : 0.8512
```

Figure-27: Confusion Matrix for (a) training data, (b) validation data

From figure-27, we see that the accuracy on training data is 90% and the validation data is 85.12%.

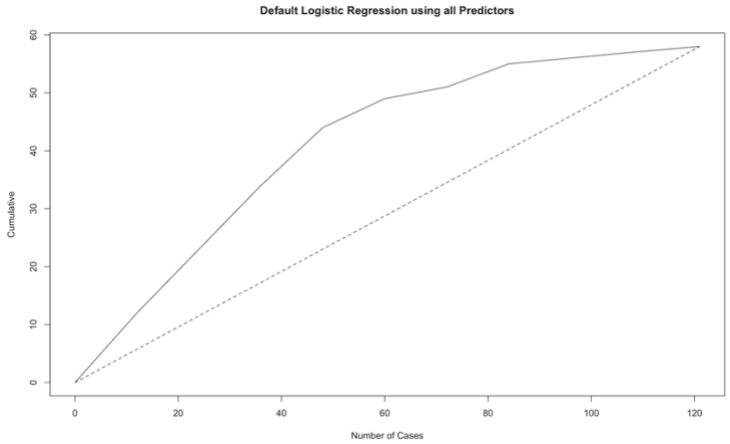

Figure-28: Lift Chart for Default Logistic Regression

From figure-28, we observe that our Default Logistic regression model is doing well in separating the important outcome (1) from outcome = 0.

(b) Stepwise Logistic Regression

By using stepwise backward variable selection, we decrease the number of predictors including dummies to 16.

```
Confusion Matrix and Statistics          Confusion Matrix and Statistics

          Reference                                Reference
Prediction  0  1                         Prediction  0  1
         0 93  9                                   0 55 12
         1  8 70                                   1  8 46

               Accuracy : 0.9056                        Accuracy : 0.8347
```

Figure-29: Confusion Matrix for (a) training data, (b) validation data

From figure-29, we see that the accuracy on training data is 90.56% and the validation data is 83.47%.

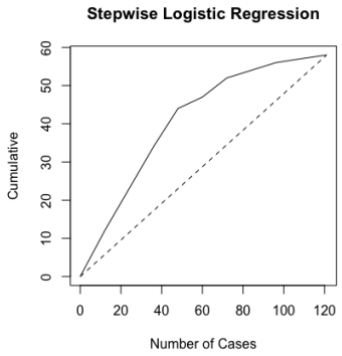

Figure-30: Lift Chart for Stepwise Logistic Regression

From figure-30, we observe that our Default Logistic regression model is doing well in separating the important outcome (1) from outcome = 0.

On comparing the confusion matrix in figure-27 and figure-29, we observe that for stepwise logistic regression, the accuracy on training data is slightly higher and on the validation data, it is slightly lower than that for default logistic regression. However, we choose the stepwise logistic regression model among the two since it gives a more parsimonious model (i.e., a model with fewer predictors).

5. Neural Network

To use a neural network, all categorical predictors with m classes must be converted into 'm-1' dummies. Also, we only want 1 output node. Thus, we change the Target variable into class numeric. This is equivalent to creating 2 dummies.

```
'data.frame':   301 obs. of  21 variables:
 $ Age     : num  63 67 67 37 41 56 62 57 63 53 ...
 $ Sex     : num  1 1 1 1 0 1 0 0 1 1 ...
 $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : num  233 286 229 250 204 236 268 354 254 203 ...
 $ fbs     : num  1 0 0 0 0 0 0 0 0 1 ...
 $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
 $ exang   : num  0 1 1 0 0 0 0 1 0 1 ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ cp0     : num  0 0 0 0 1 1 0 0 0 0 ...
 $ cp1     : num  1 0 0 0 0 0 0 0 0 0 ...
 $ cp2     : num  0 1 1 0 0 0 1 1 1 1 ...
 $ restecg0: num  0 0 0 0 0 0 0 0 0 0 ...
 $ restecg1: num  1 1 1 0 1 0 1 0 1 1 ...
 $ Slope0  : num  1 0 0 1 0 0 1 0 0 1 ...
 $ Slope1  : num  0 1 1 0 0 0 0 0 1 0 ...
 $ Ca0     : num  1 0 0 1 1 1 0 1 0 1 ...
 $ Ca1     : num  0 0 0 0 0 0 0 0 1 0 ...
 $ Ca2     : num  0 0 1 0 0 0 1 0 0 0 ...
 $ thal1   : num  1 0 0 0 0 0 0 0 0 0 ...
 $ thal2   : num  0 1 0 1 1 1 1 1 0 0 ...
 $ Target  : num  0 1 1 0 0 0 1 0 1 1 ...
```

Figure-31: Structure of data set for Neural Network

After data is partitioned into 75 % training data and 25% validation data, both the training and validation data are standardized (i.e., between 0 and 1).
We need to decide on the number of hidden nodes and layers to use.
First, we decide on the number of hidden nodes using 1 hidden layer.

	Hidden_Nodes	Accuracy_Train	Accuracy_Valid
1	5	98.67	84.21
2	6	98.67	76.32
3	7	98.67	82.89
4	8	98.67	81.58
5	9	99.11	88.16
6	10	98.67	78.95
7	11	99.11	80.26
8	12	99.11	81.58
9	13	99.11	84.21
10	14	98.67	89.47
11	15	99.11	82.89
12	16	98.67	81.58
13	17	98.67	86.84
14	18	99.11	86.84
15	19	99.11	84.21
16	20	99.11	81.58

Figure-32: Accuracy for training and validation for changing the number of hidden nodes

From figure-32, we observe that the training data accuracy increases as the number of hidden nodes increase. The validation data accuracy is maximum at nodes = 9. Thus, we choose the number of hidden nodes = 9.
Now, we decide on the number of hidden layers using 9 hidden nodes in each layer.

	Hidden_layers	Accuracy_Train	Accuracy_Valid
1	1	99.11	81.58
2	2	98.67	80.26
3	3	98.22	81.58
4	4	97.78	81.58
5	5	97.78	81.58
6	6	97.33	81.58
7	7	98.67	86.84
8	8	95.56	80.26

Figure-33: Accuracy for training and validation for changing the number of hidden layers

From figure-33, we observe that the training data accuracy increases as the number of hidden nodes increase. The validation data accuracy is maximum at layers = 7. However, overfitting will occur if we use 7 hidden layers. The accuracy on validation data is the same for hidden layers from 3 to 6. Thus, for the sake of a parsimonious model, we choose the number of hidden layers = 3.

Therefore, we will build a neural network model with 3 hidden layers with 9 nodes each and 1 output node.

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 122   3
         1   1  99

               Accuracy : 0.9822
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 37  7
         1  4 28

               Accuracy : 0.8553
```

Figure-34: Confusion Matrix for (a) training data, (b) validation data

From figure-34, we see that the accuracy on training data is 98.22% and the validation data is 85.53%.

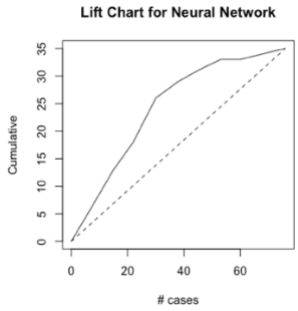

Figure-35: Lift Chart for Neural Network

From figure-35, we observe that our Neural Network model is doing well in separating the important outcome (1) from outcome = 0.

V. Performance Evaluation

For comparing the different classification models, we look at accuracy on the validation data, speed, robustness, scalability and interpretability.

	Methods	Accuracy_Validation
1	k-NN	85.56000
2	Naive Bayes	85.12397
3	Random Forest	84.44000
4	Logistic Regression (Stepwise)	83.47107
5	Neural Network	85.52632

Figure-36: Accuracy on validation data for the different models

From figure-36, we observe that the highest accuracy on validation data is achieved by k-NN, closely followed by the Neural Network. However, we choose the neural network model to be implemented for real-life since k-NN requires a large data set to get better results and the data set used in this case study was not large.

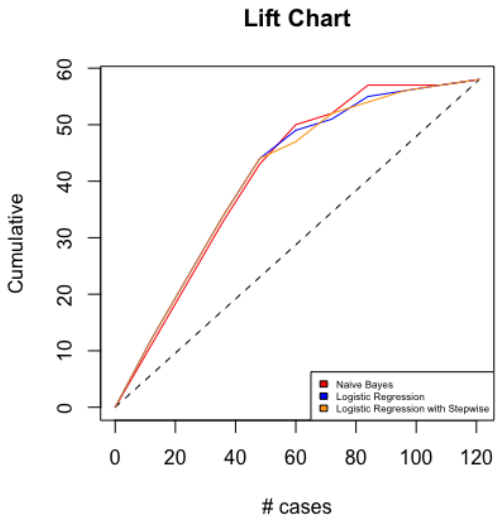

Figure-37: Lift Chart for the different models

We could not compare all the models using the lift chart since for Neural Network, the data is partitioned in different ratios. Also, for k-NN, it was not possible to build a lift chart. From figure-37, we observe that the Naïve Bayes model and both the Logistic Regression models perform almost equivalently. Thus, we cannot decide which model is better for ranking purposes based on the lift chart.

VI. Discussion and Recommendation

Since the data set is balanced, all the models have high accuracy on the validation data as can be seen in figure-36. However, the records in the data set is only 301. Thus, models that are data-driven are not effective in classifying new data. For this reason, implementing k-NN, Naïve Bayes classifier, and Random Forest into the real-world poses a problem. The most effective model is thus, neural network. Also, to decrease the running time of the neural network model, variable selection can help.
To make the model more robust, a model can be developed that classifies by taking a vote from the results of the 5 models already implemented. In this way, the newer model would have higher accuracy on new data than from any of the previous 5 models.

VII. Summary

5 different models were used for the case study of heart disease classification. The data set was clean and had only 5 missing values which were imputed with the median. The accuracy on validation data for all the models was relatively close. The best model was chosen to be the Neural Network model since other models with similar accuracy were highly data-driven.

Appendix: R Code for use case study

```
library(readxl)
library(ggplot2)
library(reshape)
library(caret)
library(FNN)
library(e1071)
library (gains)
library (neuralnet)
library (stats)
library (rpart)
library (rpart.plot)
library (randomForest)

raw.heart.df <- read_excel ("Raw Data.xlsx" )
str(raw.heart.df)

# converting the raw data to binary values
# sex (male: 1, fem: 0)
str(raw.heart.df$Sex)
raw.heart.df$Sex <- ifelse (raw.heart.df$Sex == "male", 1, 0)
summary (as.factor (raw.heart.df$Sex))
str(raw.heart.df$Sex)


# cp - chest pain type (4 values from 0 to 3)
# Typical Angina (angina): 3
# Atypical angina (abnang): 1
# Non-typical Angina (notang): 2
# Asymptomatic (asympt): 0

raw.heart.df$cp <- as.factor(raw.heart.df$cp)
summary(raw.heart.df$cp)
levels(raw.heart.df$cp) <- c (1, 3, 0, 2)
summary(raw.heart.df$cp)
str(raw.heart.df$cp)

# checking for any missing values
any(is.na(raw.heart.df$cp))
# no missing values
```

```
# fbs - fasting blood sugar (TRUE: 1 when < 120 mg/dl, fal: 0)
str(raw.heart.df$fbs)
raw.heart.df$fbs <- ifelse (raw.heart.df$fbs == "TRUE", 1, 0)
str(raw.heart.df$fbs)
summary (as.factor (raw.heart.df$fbs))


# restecg- resting electrocardiographic results (values 0,1,2)
# norm: 1, abn: 2, hyp: 0
str(raw.heart.df$restecg)
raw.heart.df$restecg <- as.factor(raw.heart.df$restecg)
summary(raw.heart.df$restecg)
levels (raw.heart.df$restecg) <- c (2, 0, 1)
summary(raw.heart.df$restecg)
str(raw.heart.df$restecg)

# exang - exercise-induced angina (if TRUE: 1, fal: 0)
str(raw.heart.df$exang)
raw.heart.df$exang <- ifelse (raw.heart.df$exang == "TRUE", 1, 0)
str(raw.heart.df$exang)
summary (as.factor (raw.heart.df$exang))

# Slope - the slope of the peak exercise ST segment
# up: 2, flat:1, down: 0
str(raw.heart.df$Slope)
raw.heart.df$Slope <- as.factor(raw.heart.df$Slope)
summary(raw.heart.df$Slope)
levels (raw.heart.df$Slope) <- c (0, 1, 2)
summary(raw.heart.df$Slope)
str(raw.heart.df$Slope)

# Ca
summary(as.factor(raw.heart.df$Ca))
# we see that there are 5 missing values marked with '?'
raw.heart.df$Ca <- as.numeric(raw.heart.df$Ca)
missing_Ca <- which (is.na (raw.heart.df$Ca))
missing_Ca
raw.heart.df [missing_Ca, ]
# We will impute these missing values with the median value for Ca
median_Ca <- median (raw.heart.df$Ca [ - missing_Ca])
raw.heart.df$Ca [missing_Ca] <- median_Ca
summary(raw.heart.df$Ca)
# no missing values remain now
raw.heart.df$Ca <- factor(raw.heart.df$Ca)
summary (raw.heart.df$Ca)

# thal- norm: 2, fix: 1, rev: 3
str(raw.heart.df$thal)
summary(as.factor (raw.heart.df$thal))
# since 2 values are marked with "?", it means that they are missing
missing_thal <- which(raw.heart.df$thal == "?")
missing_thal
# rows = 83, 199
# looking at other values in these rows
raw.heart.df [missing_thal, ]
# we see that the target variable is buff and sick respectively for rows 83 and 199.
# So we will remove these from the dataset altogether
# now remove rows 83 and 199
raw.heart.df <- raw.heart.df [-missing_thal, ]
raw.heart.df$thal <- as.factor(raw.heart.df$thal)
summary(raw.heart.df$thal)
levels (raw.heart.df$thal) <- c (1, 2, 3)
summary(raw.heart.df$thal)

# target variable has values:
# buff: 0 for not having heart disease
# sick: 1 for having heart disease
summary (as.factor (raw.heart.df$Target))
# we see that the data set is balanced
raw.heart.df$Target <- ifelse (raw.heart.df$Target == "buff", 0, 1)
summary (as.factor (raw.heart.df$Target))

str(raw.heart.df)
```

```
# we know that Age, trestbps, chol, thalach, oldpeak are required as numerical variables.
# the others are needed as categorical variables

# Data exploration

# barchart of % target vs every other variable
# 1. vs age
data_1 <- aggregate(raw.heart.df$Target, by = list(raw.heart.df$Age), FUN = mean)
data_1
names(data_1) <- c ("Age", "Mean_Target")
barplot(data_1$Mean_Target * 100, names.arg = data_1$Age, xlab = "Age", ylab = "% of Target")
# we can see that majority of the population get a heart disease after the age of 50

# 2. vs sex
data_2 <- aggregate(raw.heart.df$Target, by = list(raw.heart.df$Sex), FUN = mean)
data_2
names(data_2) <- c ("Sex", "Mean_Target")
barplot(data_2$Mean_Target * 100, names.arg = data_2$Sex, xlab = "Sex", ylab = "% of Target")
# shows that males have higher chance of having a heart disease

# histogram for all numerical variables
par (mfrow = c (3, 2))
hist(raw.heart.df$Age, main = "(a) Histogram of Age", xlab = "Age") # seems to follow a normal distribution
hist(raw.heart.df$trestbps, main = "(b) Histogram of trestbps", xlab = "trestbps") # slightly right skewed
hist(raw.heart.df$chol, main = "(c) Histogram of chol", xlab = "chol")   # outlier
hist(raw.heart.df$thalach, main = "(d) Histogram of thalach", xlab = "thalach")  # slightly left-skewed
hist(raw.heart.df$oldpeak, main = "(e) Histogram of oldpeak", xlab = "oldpeak")  # heavily right-skewed. See log transformation
hist (log (raw.heart.df$oldpeak), main = "(f) Histogram of log (oldpeak)", xlab = "oldpeak")
par (mfrow = c (1, 1))

# boxplots
par (mfrow = c (2, 3))
boxplot(raw.heart.df$Age ~ raw.heart.df$Target, ylab = "Age", xlab = "Target")
# one outlier for someone with a heart disease. Age < 40. However, this doesn't
# seem like an error since the person might have a heart disease due to some other factors.

boxplot(raw.heart.df$trestbps ~ raw.heart.df$Target, ylab = "trestbps", xlab = "Target")

boxplot(raw.heart.df$chol ~ raw.heart.df$Target, ylab = "chol", xlab = "Target")
boxplot(raw.heart.df$thalach ~ raw.heart.df$Target, ylab = "thalach", xlab = "Target")
boxplot(raw.heart.df$oldpeak ~ raw.heart.df$Target, ylab = "oldpeak", xlab = "Target")
par (mfrow = c (1, 1))

# heatmap for correlation for numerical variable with target
cor.mat <- round(cor(raw.heart.df [, c(1,2,4,5,6,8,9,10,14)]), 2)
melted.cor.mat <- melt (cor.mat)
ggplot(melted.cor.mat, aes (x = X1, y = X2, fill = value)) + geom_tile() +
  geom_text(aes(x = X1, y = X2, label = value))

cor (raw.heart.df[, c(1,2,4,5,6,8,9,10,14)])

# Converting Target into categorical
# Target
raw.heart.df$Target <- as.factor (raw.heart.df$Target)
summary (raw.heart.df$Target)

# Now we have connverted all variables into categorical variables
str(raw.heart.df)




# Model-1: k-NN
heart.df_knn <- as.data.frame (raw.heart.df)
str(heart.df_knn)

# to use k-NN, all categorical predictors with m classes should have m dummies
# so we need to create dummies for predictors: cp, restecg, Slope, Ca, thal
# for cp
heart.df_knn [,c("cp0", "cp1", "cp2", "cp3")] <- model.matrix( ~ cp - 1, data = heart.df_knn)
# for restecg
heart.df_knn [,c("restecg0", "restecg1", "restecg2")] <- model.matrix( ~ restecg - 1,
                                                 data = heart.df_knn)

# for Slope
```

```
heart.df_knn [,c("Slope0", "Slope1", "Slope2")] <- model.matrix( ~ Slope - 1, data = heart.df_knn)

# for Ca
heart.df_knn [, c("Ca0", "Ca1", "Ca2", "Ca3")] <- model.matrix(~ Ca - 1, data = heart.df_knn)

# for thal
heart.df_knn [,c("thal1", "thal2", "thal3")] <- model.matrix( ~ thal - 1, data = heart.df_knn)


str(heart.df_knn)
# deleting cp, restecg, Slope, Ca, thal
heart.df_knn <- heart.df_knn [, -c (3, 7, 11:13)]
str (heart.df_knn)

# rearranging variables such that Target variable is the last column
heart.df_knn <- heart.df_knn [, c(1:8, 10:26, 9)]
str (heart.df_knn)

# Partitioning the data into training (50%), validation (30%) and test (20%) sets.
set.seed(110)
train1.index <- sample(row.names(heart.df_knn), 0.5*dim(heart.df_knn)[1])
valid1.index <- sample(setdiff(row.names(heart.df_knn), train1.index), 0.3*dim(heart.df_knn)[1])
test1.index <- setdiff(row.names(heart.df_knn), union(train1.index, valid1.index))
train1.df <- heart.df_knn[train1.index, ]
valid1.df <- heart.df_knn[valid1.index, ]
test1.df <- heart.df_knn[test1.index, ]

# Normalizing the data
heart.norm.df <- heart.df_knn
train1.norm.df <- train1.df
valid1.norm.df <- valid1.df
test1.norm.df <- test1.df

# normalizing using preProcess
norm1.values <- preProcess(train1.df [, -26], method = c ("center", "scale"))
# preProcess: normalizes the data -> (x - mean(x))/ sd(x)
# method = "center" subtracts the mean of the predictor's data (again
# from the data in x) from the predictor values while method = "scale"
# divides by the standard deviation
heart.norm.df [,-26] <- predict (norm1.values, heart.df_knn [, -26])
train1.norm.df [, -26] <- predict (norm1.values, train1.df [, -26])
valid1.norm.df [, -26] <- predict (norm1.values, valid1.df [, -26])
test1.norm.df [, -26] <- predict (norm1.values, test1.norm.df [, -26])

# initialize a data frame with two columns: k, accuracy and error.
accuracy.df <- data.frame(k = seq(1, 20, 1), accuracy = rep(0, 20), error = rep(0, 20))

# compute knn for different k on validation data.
for(i in 1:20) {
  knn.pred <- knn(train = train1.norm.df [, -26], test= valid1.norm.df [, -26],
                  cl = train1.norm.df$Target, k = i)
  accuracy.df[i, 2] <- round (100 * confusionMatrix(table(knn.pred, valid1.norm.df$Target))$overall[1],
                              2)
  accuracy.df[i, 3] <- round (100 * (1 - confusionMatrix(table(knn.pred, valid1.norm.df$Target))$overall[1]),
                              2)
}
accuracy.df
plot(x = accuracy.df$k, y = accuracy.df$accuracy, type = "b", xlab = "k",
     ylab = "Accuracy (in %)", main = "Accuracy for Different values of k")

# As we know that a low k value would result in overfitting. To reduce overfiiting,
# we choose k such that the accuracy is still high. The accuracy is highest for k = 9, 10 and 11.
# The second highest accuracy is for k = 8. Thus, we choose k = 8 whose accuracy is worse only to k = 9.
# (which would result in overfitting).

# for k = 8
# on test data
knn_8 <- knn(train = train1.norm.df [, -26], test = test1.norm.df [, -26],
             cl = train1.norm.df$Target, k = 8)
confusionMatrix(data = knn_8, reference = test1.norm.df$Target)
knn.accuracy_test <- confusionMatrix(data = knn_8, reference = test1.norm.df$Target)$overall [1]
knn.accuracy_test * 100
# Accuracy on test data = 83.60656 %

# on validation data
```

```
knn.accuracy_valid <- accuracy.df$accuracy [8]
knn.accuracy_valid
# Accuracy on validation data = 85.55556 %

knn_8_prob <- knn(train = train1.norm.df [, -26], test = test1.norm.df [, -26],
                  cl = train1.norm.df$Target, k = 8, prob = TRUE)
str(knn_8_prob)
# lift.knn <- gains(actual = ifelse (valid.norm.df$Target == "1", 1, 0),
#                   predicted = ifelse (knn_8 == "1", 1, 0), groups = 10)




# # Model-2: Naives Bayes

heart.df_nb <- as.data.frame (raw.heart.df)
str (heart.df_nb)

# all predictors should be categorical
# Age: numerical predictor. Thus binning it in age of 5.
heart.df_nb$Age <- factor (round (heart.df_nb$Age/5))
heart.df_nb$Sex <- factor (heart.df_nb$Sex)

# trestbps: numerical predictor. Thus binning it in groups of 5.
heart.df_nb$trestbps <- factor(round(heart.df_nb$trestbps/5))

# chol: numerical predictor. Thus binning it in groups of 10.
heart.df_nb$chol <- factor(round(heart.df_nb$chol/10))
heart.df_nb$fbs <- factor(heart.df_nb$fbs)

# thalach: numerical predictor. Thus binning it in groups of 5.
heart.df_nb$thalach <- factor(round(heart.df_nb$thalach/5))
heart.df_nb$exang <- factor(heart.df_nb$exang)

# oldpeak: numerical predictor. Thus binning it.
summary(heart.df_nb$oldpeak)
heart.df_nb$oldpeak <- factor(round (heart.df_nb$oldpeak))

str (heart.df_nb)

# partition into training  (60%) and validation (40%) data
set.seed (110)
train2.index <- sample (row.names(heart.df_nb), 0.6*dim(heart.df_nb)[1])
valid2.index <- setdiff(row.names(heart.df_nb), train2.index)
train2.df <- heart.df_nb [train2.index, ]
valid2.df <- heart.df_nb [valid2.index, ]

nb <- naiveBayes(Target ~ ., data = train2.df)
nb

# perdict probabilities
pred.prob <- predict (nb, newdata = valid2.df, type = "raw")
head (pred.prob)

# predict class membership
pred.class <- predict(nb, newdata = valid2.df)
head (pred.class)

# create a data frame
df <- data.frame(actual = valid2.df$Target, predicted = pred.class, pred.prob)
names(df) <- c ("Actual", "Predicted", "P(Target = 0 / all predictors)", "P(Target = 1/ all predictors)")
head (df)
# this data frame gives the actual and predicted outcome. Also, it gives the
# Posterior probabilities

# for training data
pred.class_train <- predict (nb, newdata = train2.df)
confusionMatrix(pred.class_train, train2.df$Target)
nb.accuracy_train <- 100 * confusionMatrix(pred.class_train, train2.df$Target)$overall [1]
nb.accuracy_train

# for validation data
pred.class_valid <- predict (nb, newdata = valid2.df)
confusionMatrix(pred.class_valid, valid2.df$Target)
```

```
nb.accuracy_valid <- 100 * confusionMatrix(pred.class_valid, valid2.df$Target)$overall [1]
nb.accuracy_valid

# lift chart
lift.nb <- gains(actual = ifelse (valid2.df$Target == "1", 1, 0),
                 predicted = pred.prob [, 2], groups = 10)
str(lift.nb)

plot (y = c (0, lift.nb$cume.pct.of.total * sum (valid2.df$Target == "1")),
       x = c (0, lift.nb$cume.obs), xlab = "# cases", ylab = "Cumulative",
      main = "Lift Chart for Naive Bayes", type = "l")
# plotting naive rule
lines(y = c (0, sum (valid2.df$Target == "1")), x = c(0, dim(valid2.df)[1]), lty = 2)
```

```
# Model-3: Classification Tree
heart.df_ct <- as.data.frame(raw.heart.df)
str (heart.df_ct)

# Sex, fbs and exang are categorical variables.

# for Sex
summary (as.factor (heart.df_ct$Sex))
heart.df_ct$Sex <- as.factor(heart.df_ct$Sex)
summary (heart.df_ct$Sex)

# for fbs
summary (as.factor (heart.df_ct$fbs))
heart.df_ct$fbs <- as.factor(heart.df_ct$fbs)
summary (heart.df_ct$fbs)

# for exang
summary (as.factor (heart.df_ct$exang))
heart.df_ct$exang <- as.factor(heart.df_ct$exang)
summary (heart.df_ct$exang)

str(heart.df_ct)

# Partitioning the data into training (60%), validation (30%) and test (10%) sets.
set.seed(110)
train3.index <- sample(row.names(heart.df_ct), 0.6*dim(heart.df_ct)[1])
valid3.index <- sample(setdiff(row.names(heart.df_ct), train3.index), 0.3*dim(heart.df_ct)[1])
test3.index <- setdiff(row.names(heart.df_ct), union(train3.index, valid3.index))
train3.df <- heart.df_ct[train3.index, ]
valid3.df <- heart.df_ct[valid3.index, ]
test3.df <- heart.df_ct[test3.index, ]

# 1. default tree
default.tree <- rpart (Target ~ ., data = train3.df, method = "class")

# plotting the tree
prp (default.tree, type = 1, split.font = 1, varlen = -10, extra = 1, under = TRUE)

# number of leaves
leaves_default <- length (default.tree$frame$var [default.tree$frame$var == "<leaf>"])
leaves_default

# prediction on training data
default.tree.pred_train <- predict (default.tree, train3.df, type = "class")
confusionMatrix(data = default.tree.pred_train, reference = train3.df$Target)

Accuracy.default_train <- round (100 * confusionMatrix(data = default.tree.pred_train,
                                    reference = train3.df$Target)$overall [1], 2)
Accuracy.default_train

# prediction on validation data
default.tree.pred_valid <- predict (default.tree, valid3.df, type = "class")
confusionMatrix(data = default.tree.pred_valid, reference = valid3.df$Target)

Accuracy.default_valid <- round (100 * confusionMatrix(data = default.tree.pred_valid,
                                    reference = valid3.df$Target)$overall [1], 2)
```

```
Accuracy.default_valid

# prediction using test data
default.tree.pred_test <- predict (default.tree, test3.df, type = "class")
confusionMatrix(data = default.tree.pred_test, reference = test3.df$Target)

Accuracy.default_test <- round (100 * confusionMatrix(data = default.tree.pred_test,
                                reference = test3.df$Target)$overall [1], 2)
Accuracy.default_test

# 2. full grown tree
full.tree <- rpart (Target ~ ., data = train3.df, method = "class", cp = 0, minsplit = 1)

# plotting the tree
prp (full.tree, type = 1, split.font = 1, varlen = -10, extra = 1, under = TRUE,
    box.col = ifelse (full.tree$frame$var == "<leaf>", "red", "gray"))

# number of leaves
leaves_full <- length (full.tree$frame$var [full.tree$frame$var == "<leaf>"])
leaves_full

# prediction on training data
full.tree.pred_train <- predict (full.tree, train3.df, type = "class")
confusionMatrix(data = full.tree.pred_train, reference = train3.df$Target)

Accuracy.full_train <- round (100 * confusionMatrix(data = full.tree.pred_train,
                              reference = train3.df$Target)$overall [1], 2)
Accuracy.full_train

# prediction on validation data
full.tree.pred_valid <- predict (full.tree, valid3.df, type = "class")
confusionMatrix(data = full.tree.pred_valid, reference = valid3.df$Target)

Accuracy.full_valid <- round (100 * confusionMatrix(data = full.tree.pred_valid,
                              reference = valid3.df$Target)$overall [1], 2)
Accuracy.full_valid

# prediction using test data
full.tree.pred_test <- predict (full.tree, test3.df, type = "class")
confusionMatrix(data = full.tree.pred_test, reference = test3.df$Target)

Accuracy.full_test <- round (100 * confusionMatrix(data = full.tree.pred_test,
                             reference = test3.df$Target)$overall [1], 2)
Accuracy.full_test

# 3. Using cross validation to make the best-pruned tree

cross.validation <- rpart(Target ~ ., data = train3.df, method = "class", cp = 0.00001, xval = 5,
                    minsplit = 2)
# plotting the tree
prp (cross.validation, type = 1, split.font = 1, varlen = -10, extra = 1, under = TRUE)

# printcp: Displays the cp table for fitted rpart object.
printcp (cross.validation)

# number of leaves
leaves_cross.validation <- length (cross.validation$frame$var [cross.validation$frame$var == "<leaf>"])
leaves_cross.validation

# prediction on training data
cross.validation.pred_train <- predict (cross.validation, train3.df, type = "class")
confusionMatrix(data = cross.validation.pred_train, reference = train3.df$Target)

Accuracy.cross.validation_train <- round (100 * confusionMatrix(data = cross.validation.pred_train,
                                          reference = train3.df$Target)$overall [1], 2)
Accuracy.cross.validation_train

# prediction on validation data
cross.validation.pred_valid <- predict (cross.validation, valid3.df, type = "class")
confusionMatrix(data = cross.validation.pred_valid, reference = valid3.df$Target)

Accuracy.cross.validation_valid <- round (100 * confusionMatrix(data = cross.validation.pred_valid,
                                          reference = valid3.df$Target)$overall [1], 2)
Accuracy.cross.validation_valid
```

```
# prediction using test data
cross.validation.pred_test <- predict (cross.validation, test3.df, type = "class")
confusionMatrix(data = cross.validation.pred_test, reference = test3.df$Target)

Accuracy.cross.validation_test <- round (100 * confusionMatrix(data = cross.validation.pred_test,
                                        reference = test3.df$Target)$overall [1], 2)
Accuracy.cross.validation_test

# 4. Best-pruned tree
prune.tree <- prune(cross.validation,
                cp = cross.validation$cptable [which.min(cross.validation$cptable [, "xerror"]), "CP"])

# plotting the best-pruned tree
prp (prune.tree, type = 1, split.font = 1, varlen = -10, extra = 1, under = TRUE)

# number of leaves
leaves_prune <- length (prune.tree$frame$var [prune.tree$frame$var == "<leaf>"])
leaves_prune

# prediction on training data
prune.tree.pred_train <- predict (prune.tree, train3.df, type = "class")
confusionMatrix(data = prune.tree.pred_train, reference = train3.df$Target)

Accuracy.prune_train <- round (100 * confusionMatrix(data = prune.tree.pred_train,
                                    reference = train3.df$Target)$overall [1], 2)
Accuracy.prune_train

# prediction on validation data
prune.tree.pred_valid <- predict (prune.tree, valid3.df, type = "class")
confusionMatrix(data = prune.tree.pred_valid, reference = valid3.df$Target)

Accuracy.prune_valid <- round (100 * confusionMatrix(data = prune.tree.pred_valid,
                                    reference = valid3.df$Target)$overall [1], 2)
Accuracy.prune_valid

# prediction using test data
prune.tree.pred_test <- predict (prune.tree, test3.df, type = "class")
confusionMatrix(data = prune.tree.pred_test, reference = test3.df$Target)

Accuracy.prune_test <- round (100 * confusionMatrix(data = prune.tree.pred_test,
                                    reference = test3.df$Target)$overall [1], 2)
Accuracy.prune_test

# 5. Random forest
rf <- randomForest(Target ~ ., data = train3.df, ntree = 500, mtry = 4, nodesize = 5, importance = TRUE)

# variable importance plot
varImpPlot(rf, type = 1)
# we see that the top 4 predictors are thal, Ca, oldpeak, cp.

# prediction on training data
rf.pred_train <- predict (rf, train3.df, type = "class")
confusionMatrix(data = rf.pred_train, reference = train3.df$Target)

Accuracy.rf_train <- round (100 * confusionMatrix(data = rf.pred_train,
                                   reference = train3.df$Target)$overall [1], 2)
Accuracy.rf_train

# prediction on validation data
rf.pred_valid <- predict (rf, valid3.df, type = "class")
confusionMatrix(data = rf.pred_valid, reference = valid3.df$Target)

Accuracy.rf_valid <- round (100 * confusionMatrix(data = rf.pred_valid,
                                   reference = valid3.df$Target)$overall [1], 2)
Accuracy.rf_valid

# prediction on test data
rf.pred_test <- predict (rf, test3.df, type = "class")
confusionMatrix(data = rf.pred_test, reference = test3.df$Target)

Accuracy.rf_test <- round (100 * confusionMatrix(data = rf.pred_test,
                                  reference = test3.df$Target)$overall [1], 2)
Accuracy.rf_test

#create data frame for number of leaves and accuracy for the different trees
```

```
output_table <- data.frame(Leaves = c (leaves_default, leaves_full, leaves_prune, "NA"),
                           Accuracy_Training = c (Accuracy.default_train, Accuracy.full_train,
                                                  Accuracy.prune_train, Accuracy.rf_train),
                           Accuracy_Validation = c (Accuracy.default_valid, Accuracy.full_valid,
                                                    Accuracy.prune_valid, Accuracy.rf_valid),
                           Accuracy_Test = c (Accuracy.default_test, Accuracy.full_test,
                                              Accuracy.prune_test, Accuracy.rf_test))
row.names(output_table) <- c ("Default Tree", "Full Grown Tree", "Best-Pruned Tree", "Random Forest")
output_table
# from this table, we can see that the model with the highest accuracy on validation set
# is "Random Forest".

tree.accuracy_valid <- Accuracy.rf_valid
tree.accuracy_test <- Accuracy.rf_test




# Model-4: Logistic Regression

heart.df_lg <- as.data.frame (raw.heart.df)
str (heart.df_lg)

# partioning into 60% training and 40% validation data
set.seed (110)
train4.index <- sample (row.names(heart.df_lg), 0.6 * dim (heart.df_lg) [1])
valid4.index <- setdiff(row.names(heart.df_lg), train4.index)
train4.df <- heart.df_lg [train4.index, ]
valid4.df <- heart.df_lg [valid4.index, ]

# running logistic regression using all predictors
n <- names (heart.df_lg)
f <- as.formula (paste ("Target ~ ", paste (n [!n %in% "Target"], collapse = "+")))
f

# glm function automatically takes m-1 dummies from m categories
lg_1 <- glm(f, data = train4.df, family = "binomial")
summary (lg_1)
# total number pf predictors including dummies = 20

# predict on training data
# this gives us P (Y = 1)
lg_1.train_pred <- predict (lg_1, train4.df [, -14], type = "response")
head (lg_1.train_pred)

# converting into class
lg_1.train_pred_class <- ifelse (lg_1.train_pred > 0.5, 1, 0)

# comparing using confusion matrix
confusionMatrix(data = as.factor (lg_1.train_pred_class), reference = as.factor(train4.df$Target))
# accuracy = 90 %

# predict on validation data
# this gives us P (Y = 1)
lg_1.valid_pred <- predict (lg_1, valid4.df [, -14], type = "response")
head (lg_1.valid_pred)

# converting into class
lg_1.valid_pred_class <- ifelse (lg_1.valid_pred > 0.5, 1, 0)

# comparing using confusion matrix
confusionMatrix(data = as.factor (lg_1.valid_pred_class), reference = as.factor(valid4.df$Target))
# accuracy = 85.12 %

# lift chart
lift.lg <- gains(actual = ifelse (valid4.df$Target == "1", 1, 0), predicted = lg_1.valid_pred)

plot (y = c (0, lift.lg$cume.pct.of.total * sum (ifelse (valid4.df$Target == "1", 1, 0))),
      x = c (0, lift.lg$cume.obs), xlab = "Number of Cases", ylab = "Cumulative",
      main = "Default Logistic Regression using all Predictors", type = "l")
lines (y = c (0, sum (ifelse (valid4.df$Target == "1", 1, 0))), x = c (0, dim (valid4.df)[1]), lty = 2)
```

```
# selecting predictors using stepwise backward selection

lg_2.step <- step(lg_1, direction = "backward")
summary (lg_2.step)
# total number pf predictors including dummies = 16

# prediction on training data
lg_2.step.train_pred <- predict (lg_2.step, train4.df [, -14], type = "response")
lg_2.step.train_pred_class <- ifelse (lg_2.step.train_pred > 0.5, 1, 0)
confusionMatrix(as.factor (lg_2.step.train_pred_class), as.factor (train4.df$Target))
# accuracy = 90.56%

# prediction on validation data
lg_2.step.valid_pred <- predict (lg_2.step, valid4.df [, -14], type = "response")
lg_2.step.valid_pred_class <- ifelse (lg_2.step.valid_pred > 0.5, 1, 0)
confusionMatrix(as.factor (lg_2.step.valid_pred_class), as.factor (valid4.df$Target))
lg.accuracy_valid <- 100 * (confusionMatrix(as.factor (lg_2.step.valid_pred_class),
                                as.factor (valid4.df$Target))$overall [1])
lg.accuracy_valid
# accuracy = 83.47%

# The accuracy is slightly lower than using all the predictors.
# However, we have chosen a parsimonious model.

# lift chart
lift.lg_step <- gains(actual = ifelse (valid4.df$Target == "1", 1, 0), predicted = lg_2.step.valid_pred)
plot (y = c (0, lift.lg_step$cume.pct.of.total * sum (ifelse (valid4.df$Target == "1", 1, 0))),
     x = c (0, lift.lg_step$cume.obs), xlab = "Number of Cases", ylab = "Cumulative",
     main = "Stepwise Logistic Regression", type = "l")
lines (y = c (0, sum (ifelse (valid4.df$Target == "1", 1, 0))), x = c (0, dim (valid4.df)[1]), lty = 2)


# capturing nonlinear interaction
# n <- names (heart.df_lg)
# f <- paste (n [!n %in% "Target"], collapse = " + ")
# f

# lg_all <- glm (Target ~ (f)^2, data = train.df, family = "binomial")




# Model-5: Neural Networks

heart.df_nn <- as.data.frame(raw.heart.df)
str (heart.df_nn)

# to use neural network, all categorical predictors with m classes should have 'm-1' dummies
# so we need to create dummies for predictors: cp, restecg, Slope, Ca, thal
# for cp
heart.df_nn [,c("cp0", "cp1", "cp2", "cp3")] <- model.matrix( ~ cp - 1, data = heart.df_nn)

# for restecg
heart.df_nn [,c("restecg0", "restecg1", "restecg2")] <- model.matrix( ~ restecg - 1,
                                                data = heart.df_nn)

# for Slope
heart.df_nn [,c("Slope0", "Slope1", "Slope2")] <- model.matrix( ~ Slope - 1, data = heart.df_nn)

# Ca
heart.df_nn [,c("Ca0", "Ca1", "Ca2", "Ca3")] <- model.matrix( ~ Ca - 1, data = heart.df_nn)

# for thal
heart.df_nn [,c("thal1", "thal2", "thal3")] <- model.matrix( ~ thal - 1, data = heart.df_nn)

str(heart.df_nn)
# selecting 'm-1' dummies, deleting the original variables, and putting Target variable at last.
deleted.var <- - c (3, 7, 11:13, 18, 21, 24, 28, 31, 14)
heart.df_nn <- heart.df_nn [, deleted.var]
heart.df_nn$Target <- raw.heart.df$Target
str (heart.df_nn)

# We only want 1 output node. Thus, we change Target variable into numeric
```

```
# This is equivalent to creating 2 dummies
heart.df_nn$Target <- as.numeric(heart.df_nn$Target) - 1
str(heart.df_nn)

# Partitioning the data into training (75%) and validation (25%) sets.
set.seed(110)
train5.index <- sample(row.names(heart.df_nn), 0.75*dim(heart.df_nn)[1])
valid5.index <- setdiff(row.names(heart.df_nn), train5.index)
train5.df <- heart.df_nn[train5.index, ]
valid5.df <- heart.df_nn[valid5.index, ]

# Normalizing the data
heart.norm.df <- heart.df_nn
train5.norm.df <- train5.df
valid5.norm.df <- valid5.df

# Scaling using preProcess
norm5.values <- preProcess(train5.df, method = "range")
# method = "range": (X - min(X))/ (max(X) - min(X))
heart.norm.df <- predict (norm5.values, heart.df_nn)
train5.norm.df <- predict (norm5.values, train5.df)
valid5.norm.df <- predict (norm5.values, valid5.df)

# running neural network model
n <- names (heart.df_nn)
f <- as.formula (paste ("Target ~ ", paste (n [!n %in% "Target"], collapse = "+")))
f

# Using 1 hidden layer with changing number of nodes from 5 to 20
hidden <- seq (5, 20, 1)
accuracy.df <- data.frame(Hidden_Nodes = rep (0, length (hidden)),
                          Accuracy_Train = rep (0, length (hidden)),
                          Accuracy_Valid = rep (0,length (hidden)))
row.count <- 0

for (i in hidden){
  nn <- neuralnet(f, data = train5.norm.df, linear.output = FALSE, hidden = i)

  # prediction on training data
  nn_train_pred <- compute (nn, train5.norm.df [, -21])
  # Converting to class
  nn_train_pred.class <- ifelse (nn_train_pred$net.result > 0.5, 1, 0)

  # prediction on validation data
  nn_valid_pred <- compute (nn, valid5.norm.df [, -21])
  # Converting to class
  nn_valid_pred.class <- ifelse (nn_valid_pred$net.result > 0.5, 1, 0)

  row.count = row.count + 1
  accuracy.df [row.count, 1] <- i
  accuracy.df [row.count, 2] <- round (100 * confusionMatrix(as.factor (nn_train_pred.class),
                                       as.factor (train5.norm.df$Target))$overall [1], 2)
  accuracy.df [row.count, 3] <- round (100 * confusionMatrix(as.factor (nn_valid_pred.class),
                                       as.factor (valid5.norm.df$Target))$overall [1], 2)
}
accuracy.df

# We can see that the training data accuracy is increases as number of hidden nodes increases.
# However, the validation data accuracy is maximum at nodes = 9.

# Using 9 hidden nodes with changing number of layers from 1 to 8.
layers <- seq (1, 8, 1)
hidden_layers <- c()
accuracy.df_layers <- data.frame(Hidden_layers = rep (0, length (layers)),
                          Accuracy_Train = rep (0, length (layers)),
                          Accuracy_Valid = rep (0,length (layers)))
row.count <- 0

for (i in layers){
  hidden_layers [[i]] <- 9
  nn <- neuralnet(f, data = train5.norm.df, linear.output = FALSE, hidden = hidden_layers)

  # prediction on training data
  nn_train_pred <- compute (nn, train5.norm.df [, -21])
  # Converting to class
```

```
  nn_train_pred.class <- ifelse (nn_train_pred$net.result > 0.5, 1, 0)

  # prediction on validation data
  nn_valid_pred <- compute (nn, valid5.norm.df [, -21])
  # Converting to class
  nn_valid_pred.class <- ifelse (nn_valid_pred$net.result > 0.5, 1, 0)

  row.count = row.count + 1
  accuracy.df_layers [row.count, 1] <- i
  accuracy.df_layers [row.count, 2] <- round (100 * confusionMatrix(as.factor (nn_train_pred.class),
                                      as.factor (train5.norm.df$Target))$overall [1], 2)
  accuracy.df_layers [row.count, 3] <- round (100 * confusionMatrix(as.factor (nn_valid_pred.class),
                                      as.factor (valid5.norm.df$Target))$overall [1], 2)
}
accuracy.df_layers
# We can see that the training data accuracy increases as hidden layers increase.
# The validation data accuracy is maximum at hidden layers = 7. Validation Accuracy remains the same
# for the hidden layers from 3 to 6.

#Thus, for parsimonious model, we use 3 hidden layer with 9 nodes

nn <- neuralnet(f, data = train5.norm.df, linear.output = FALSE, hidden = c(9, 9, 9))

# prediction on training data
nn_train_pred <- compute (nn, train5.norm.df [, -21])
# Converting to class
nn_train_pred.class <- ifelse (nn_train_pred$net.result > 0.5, 1, 0)

# prediction on validation data
nn_valid_pred <- compute (nn, valid5.norm.df [, -21])
# Converting to class
nn_valid_pred.class <- ifelse (nn_valid_pred$net.result > 0.5, 1, 0)

confusionMatrix(as.factor (nn_train_pred.class), as.factor (train5.norm.df$Target))
# Training accuracy = 98.22%
confusionMatrix(as.factor (nn_valid_pred.class), as.factor (valid5.norm.df$Target))
# Validation Accuracy = 85.53%

nn.accuracy_train <- 100 * (confusionMatrix(as.factor (nn_train_pred.class),
                                as.factor (train5.norm.df$Target))$overall [1])
nn.accuracy_train

nn.accuracy_valid <- 100 * (confusionMatrix(as.factor (nn_valid_pred.class),
                                as.factor (valid5.norm.df$Target))$overall [1])
nn.accuracy_valid

# Genreally, neural net has high order of accuracy. This is not the case here. This could be due to the
# low amount of data since neural network requires large amount of data.

# lift chart
lift.nn <- gains (actual = valid5.norm.df$Target, predicted = nn_valid_pred$net.result)
str(lift.nn)

plot (y = c (0, lift.nn$cume.pct.of.total * sum (valid5.norm.df$Target)),
      x = c (0, lift.nn$cume.obs), xlab = "# cases", ylab = "Cumulative",
      main = "Lift Chart for Neural Network", type = "l")
# plotting naive rule
lines(y = c (0, sum (valid5.norm.df$Target)), x = c(0, dim(valid5.norm.df)[1]), lty = 2)




# Comparing accuracy on validation data for all methods
compare_accuracy.df <- data.frame(Methods = c ("k-NN", "Naive Bayes", "Random Forest",
                                  "Logistic Regression (Stepwise)", "Neural Network"),
                        Accuracy_Validation = c (knn.accuracy_valid, nb.accuracy_valid,
                                    tree.accuracy_valid, lg.accuracy_valid,
                                    nn.accuracy_valid))
compare_accuracy.df

# Highest accuracy on validation data = 85.56 % by k-NN, closely followed by Neural Network with
# 85.52 %. We choose Neural Network since k-NN works better when we have a large data set.
```

```
# lift chart in 1 plot
# for nb
plot (y = c (0, lift.nb$cume.pct.of.total * sum (valid2.df$Target == "1")),
      x = c (0, lift.nb$cume.obs), xlab = "# cases", ylab = "Cumulative",
      main = "Lift Chart", type = "l", col = "red")
# plotting naive rule
lines(y = c (0, sum (valid2.df$Target == "1")), x = c(0, dim(valid2.df)[1]), lty = 2)

# for lg
lines (y = c (0, lift.lg$cume.pct.of.total * sum (ifelse (valid4.df$Target == "1", 1, 0))),
       x = c (0, lift.lg$cume.obs), type = "l", col = "blue")

# for lg_step
lines (y = c (0, lift.lg_step$cume.pct.of.total * sum (ifelse (valid4.df$Target == "1", 1, 0))),
       x = c (0, lift.lg_step$cume.obs), type = "l", col = "orange")

# for nn
#lines (y = c (0, lift.nn$cume.pct.of.total * sum (valid5.norm.df$Target)),
 #       x = c (0, lift.nn$cume.obs), type = "l", col = "gray", xlim = c (0, 120))


legend ("bottomright", c ("Naive Bayes", "Logistic Regression",
                          "Logistic Regression with Stepwise"),
        fill = c("red", "blue", "orange"), cex = 0.5)
```