**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

*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 |

A screenshot of text

Description automatically generated

Figure-: 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.

A close up of a device

Description automatically generated A close up of a logo

Description automatically generated

Figure-: % 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.

A close up of text on a white background

Description automatically generated

Figure-4: Histogram of the numerical variables

A close up of text on a white background

Description automatically generated

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.

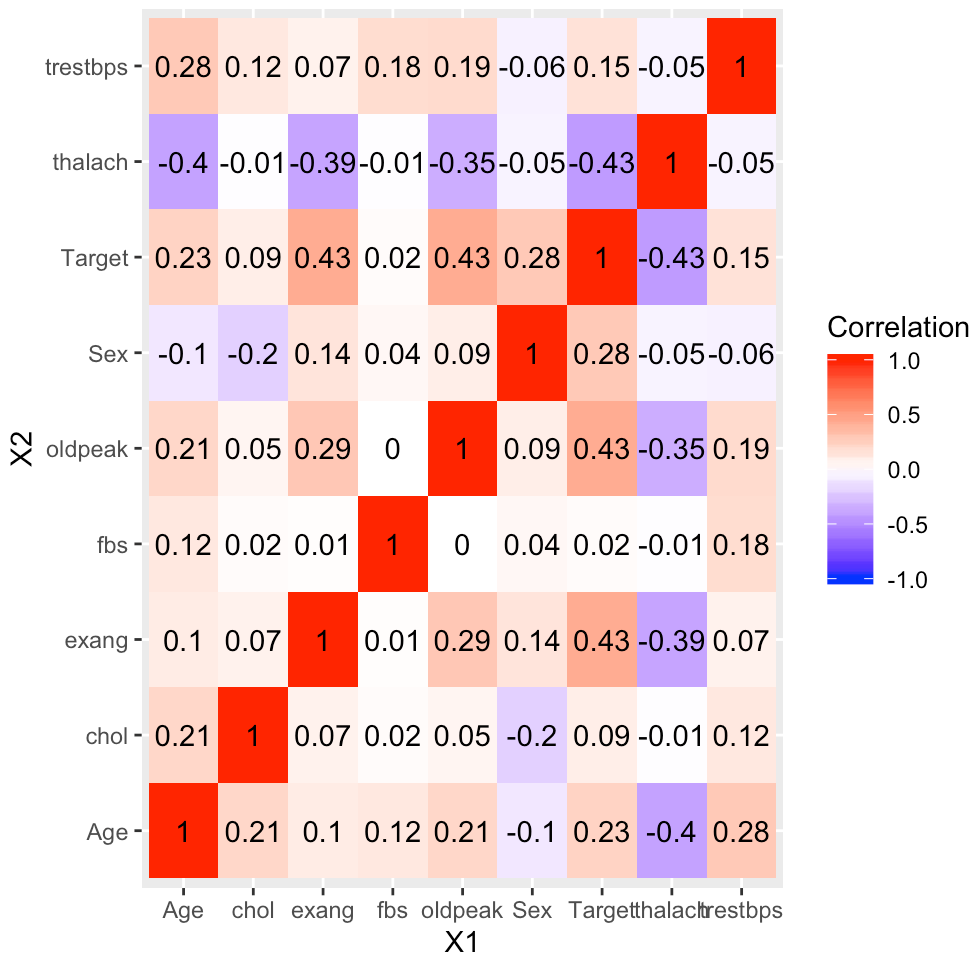


Figure-6: Heatmap for correlation of numerical variables with target

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

A close up of text on a white background

Description automatically generated

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
4. Default Tree
5. Full-Grown Tree
6. Pruned tree
7. Random Forests
8. Logistic Regression
   1. Default Logistic Regression
   2. Stepwise Logistic Regression
9. Neural Networks
10. **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.

A screenshot of a cell phone

Description automatically generated

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.

A screenshot of a cell phone

Description automatically generated A close up of a map

Description automatically generated

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.

A close up of a logo

Description automatically generated

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%.

1. **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.

A close up of text on a white background

Description automatically generated

Figure-11: Structure of data set for Naive Bayes

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

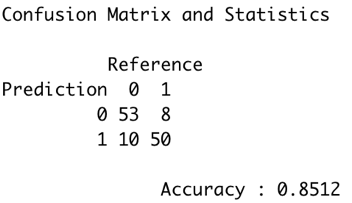
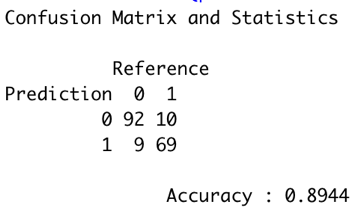


Figure-12: Confusion Matrix for (a) training data and (b) validation data

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

A screenshot of a cell phone

Description automatically generated

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.

1. **Classification Tree**

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

A close up of text on a white background

Description automatically generated

Figure-14: Structure of data set for Classification Tree

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

1. **Default Tree**

A close up of a map

Description automatically generated

Figure-15: Default Tree with leaves = 6

A screenshot of a cell phone

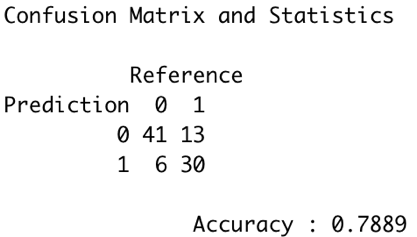
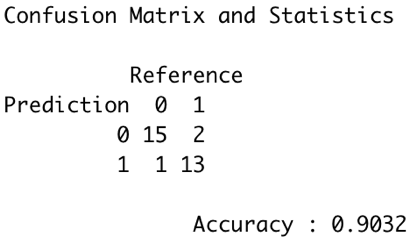
Description automatically generated  

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%.

1. **Full-Grown Tree**

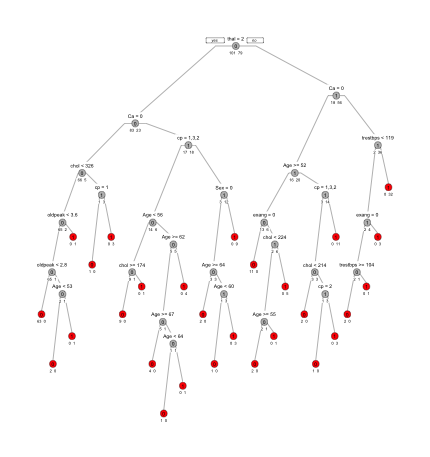


Figure-17: Full-grown Tree with leaves = 28

A screenshot of a cell phone

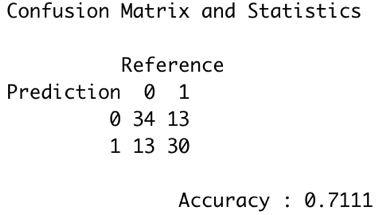
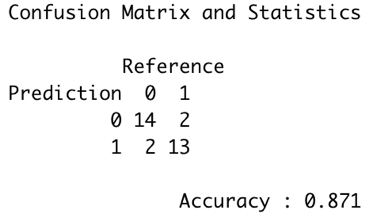
Description automatically generated  

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.

1. **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.

A picture containing outdoor, sky, skiing

Description automatically generated A screenshot of a cell phone

Description automatically generated

Figure-19: Cross-Validation Tree with leaves = 28 Figure-20: cp table for cross-validation

A close up of a map

Description automatically generated

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

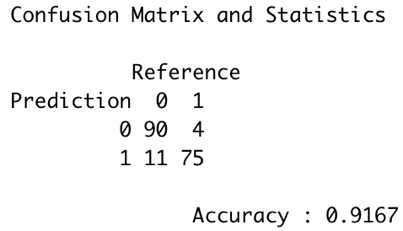
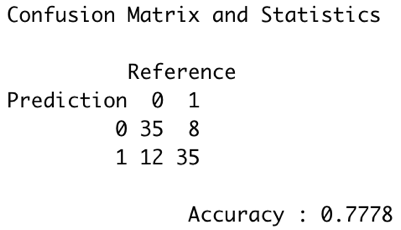
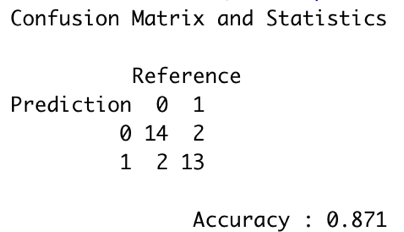
  

Figure-22: Confusion Matrix for (a) training data, (b) validation data and (c) test data

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.

1. **Random Forest**

A screenshot of a cell phone

Description automatically generated

Figure-2: Important Variables for making the forest

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

A screenshot of a cell phone

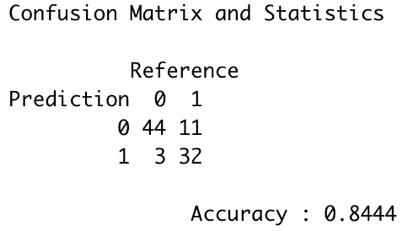
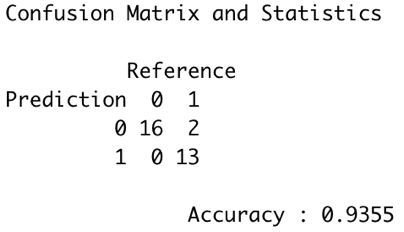
Description automatically generated  

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.

A screenshot of a cell phone

Description automatically generated

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.

1. **Logistic Regression**

**A close up of text on a white background

Description automatically generated**

Figure-26: Structure of data set for Logistic Regression

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

1. **Default Logistic Regression** (using all predictors)

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

A screenshot of a cell phone

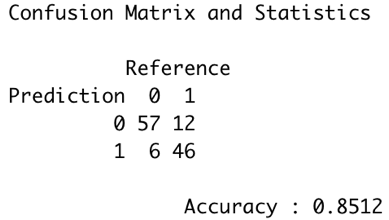
Description automatically generated

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%.

A close up of a map

Description automatically generated

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.

1. **Stepwise Logistic Regression**

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

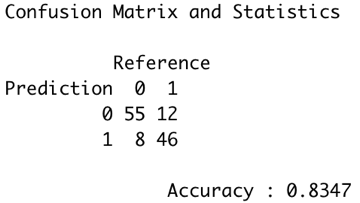
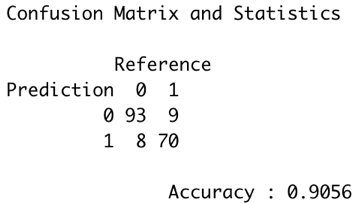


Figure-29: Confusion Matrix for (a) training data, (b) validation data

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

A screenshot of a cell phone

Description automatically generated

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).

1. **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.

A screenshot of a cell phone

Description automatically generated

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.

A close up of a piece of paper

Description automatically generated

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.

A screenshot of a cell phone

Description automatically generated

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.

A screenshot of a cell phone

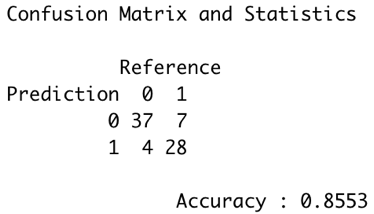
Description automatically generated

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%.

A screenshot of a cell phone

Description automatically generated

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.

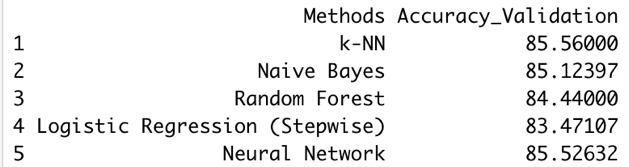


Figure-36: Accuracy on validation data for the different models

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.

A close up of a map

Description automatically generated

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)