Heart Disease Prediction using Support Vector Machine (SVM)

1. Introduction:

Data modelling is the process of identifying the patterns and discovering new insights from large dataset by using statistical methods and machine learning algorithms. There are various machine learning algorithms which are broadly classified into 2 groups - Supervised and Unsupervised learning models based on whether the output variable is labelled or not. In Supervised learning model, if the output variable is discreate or categorical, we should us the classification algorithm like decision tree or random forest to train the model. If the output variable is numeric, we should use the regression algorithm to train the model. Similarly, for Unsupervised learning model, if the output variable is not labelled, K- mean clustering algorithm should be used to divide the dataset into small clusters. The number and types of the parameter we pass while training a model also plays an important role in choosing the right algorithm. Support Vector Machine algorithms is preferred when multiple parameters are required to train the model.

The Support Vector Machine is a classification algorithm used to separate two classes of the target data by a hyperplane that has maximum margin, i.e the maximum distance between data points of each class. These data points that determines the position and the orientation of the hyperplane are called support vectors. SVM works well for high-dimensional dataset where the number of features/variables is greater than the number of observations/records.

In this project, we will analyse heart disease dataset which has 14 variables and 300 records. Since variable not named, it is hard to find the target and prediction variables. All the variables are of integer type except for the variable 10 which is a numeric. As per the instruction, the first 13 variables will be used for predicting the 14th variable which is a target variable for the given dataset. The values in the first 13 columns looks to be a information of a patient which is used to predict the binary value(0 and 1)in the 14th column, where 0 represents the absence of the heart disease and the 1 represents the presence of the heart disease.

In order to better understand the dataset, I looked for the similar dataset in Kaggle.com and found the one in the below link to be matching with our heart disease dataset.

2. Methods & analysis

2.1. Libraries used

The caret package (Classification and Regression Training) contains all necessary functions for building and training complex classification and regression problems like data splitting, pre-processing, feature selection, tuning, etc,.

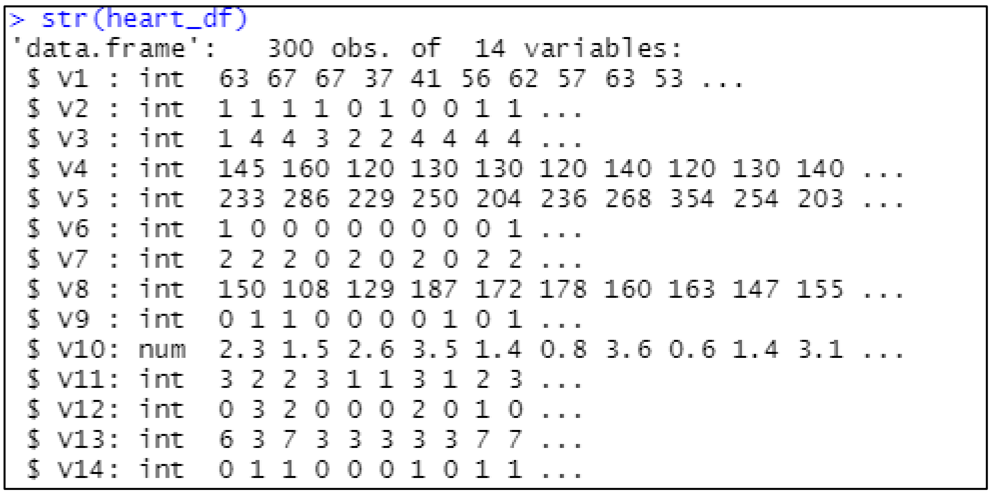
2.2. Loading the dataset

The heart_tidy.csv dataset which we used for this project is a comma separated values file. We have used setwd() function to set the working directory to the path of the dataset location. We then use read.csv() function with filename as the parameter to load the csv file as a dataframe into a variable *heart_df*. The sep parameter tell the interprets which delimiter is used as a separator in the csv file. In our dataset, since the 1st column is not the header, we have passed the value FALSE for the parameter header.

3. Exploratory Data Analysis

3.1. Structure of the dataset

We use the str() function to check the structure of the *heart_df* datframe.

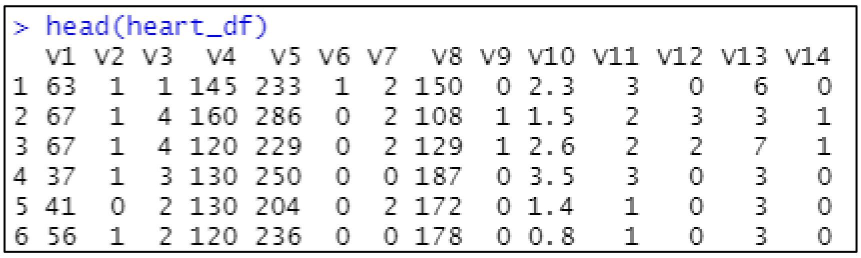


```
> str(heart_df)
'data.frame':	300 obs. of  14 variables:
 $ V1 : int  63 67 67 37 41 56 62 57 63 53 ...
 $ V2 : int  1 1 1 1 0 1 0 0 1 1 ...
 $ V3 : int  1 4 4 3 2 2 4 4 4 4 ...
 $ V4 : int  145 160 120 130 130 120 140 120 130 140 ...
 $ V5 : int  233 286 229 250 204 236 268 354 254 203 ...
 $ V6 : int  1 0 0 0 0 0 0 0 0 1 ...
 $ V7 : int  2 2 2 0 2 0 2 0 2 2 ...
 $ V8 : int  150 108 129 187 172 178 160 163 147 155 ...
 $ V9 : int  0 1 1 0 0 0 0 1 0 1 ...
 $ V10: num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ V11: int  3 2 2 3 1 1 3 1 2 3 ...
 $ V12: int  0 3 2 0 0 0 2 0 1 0 ...
 $ V13: int  6 3 7 3 3 3 3 3 7 7 ...
 $ V14: int  0 1 1 0 0 0 1 0 1 1 ...
```

Figure 1: Represents the structure of the *heart_df* datframe

From fig.1, we can understand that the dataframe has 300 observations and 14 variables. Out of 14 variables, 13 are of integer type and the variable 10 is numeric type. Since, *V14* is our target variable, we can use remaining 13 variables as an input or predictor variable. We can observe that the values of the V14 variable is binary (0 and 1).

We use the head() function to check the first few rows of the *heart_df* datframe.



```
> head(heart_df)
  V1 V2 V3  V4  V5 V6 V7  V8 V9 V10 V11 V12 V13 V14
1 63  1  1 145 233  1  2 150  0 2.3   3   0   6   0
2 67  1  4 160 286  0  2 108  1 1.5   2   3   3   1
3 67  1  4 120 229  0  2 129  1 2.6   2   2   7   1
4 37  1  3 130 250  0  0 187  0 3.5   3   0   3   0
5 41  0  2 130 204  0  2 172  0 1.4   1   0   3   0
6 56  1  2 120 236  0  0 178  0 0.8   1   0   3   0
```

Figure 2: Represents the first few rows of the *heart_df* datframe

From fig.2, we can understand that except for V10 whose values are in decimal, the values of the rest all columns are integers.

3.2 Data Splicing

Data Splicing is the process of splitting the data into a training set and a testing set. We will split the data using two random samples without replacement:

- training dataset (70%) for model building
- test dataset (30%) to validate the efficiency of the model.

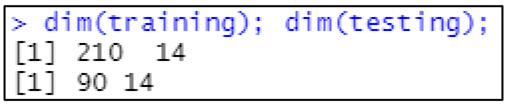
We have used the set.seed() function with the argument 3033 to get the reproducible random numbers. During data partitioning, if the same value is passed in the set.seed() method, we will get the identical split of training and testing data even though it is random.
For partitioning the data into training and testing set, we use the function createDataPartition() which is part of the caret package. The first parameter 'y' takes the value of the target variable which needs to be partitioned. We have passed heart_df$V14, the 14th column on our datframe as it's our target variable.

The "p" parameter indicates the percentage of the split between the training and testing data and it holds a decimal value in the range of 0-1. Since we want to split the training and testing data in 70:30 ratio, we have given p=0.7. Out of 300 observations from the heart_df dataframe, 70% of records (210 observations) are spitted and loaded into a training data frame. The value P= 0.7 determines that 70% of data should be loaded into the training data frame and the remaining 30% loaded into the testing data frame.

The list parameter returns the output as list if we pass the value TRUE and it returns the output as matrix if we pass the value as FALSE. Since we have pass FALSE in the list parameter, the createDataPartition() function returning a matrix "intrain".

We pass the values of the "intrain" matrix which holds the indices of the observation the heart_df dataframe to split it into training and testing data.

We use the dim() function to check the dimension of the training and testing data



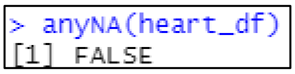
```
> dim(training); dim(testing);
[1] 210  14
[1] 90 14
```

Figure 3: Represents the dimension of the training and testing data

From fig.3, we can understand that the training dataset has 210 observations and 14 variables whereas the testing dataset has 90 observations and 14 variables.

3.3 Finding the missing values in the dataset

In general, the data may contain null values, missing values, unwanted data, outliers that may lead to wrong interpretations. Hence data pre-processing is a critical step where in the irrelevant data can be removed so we can achieve better accuracy while predicting the values using our model. We use anyNA() function to check the presence of any NA(Not Available) values in the *heart_df* dataframe.



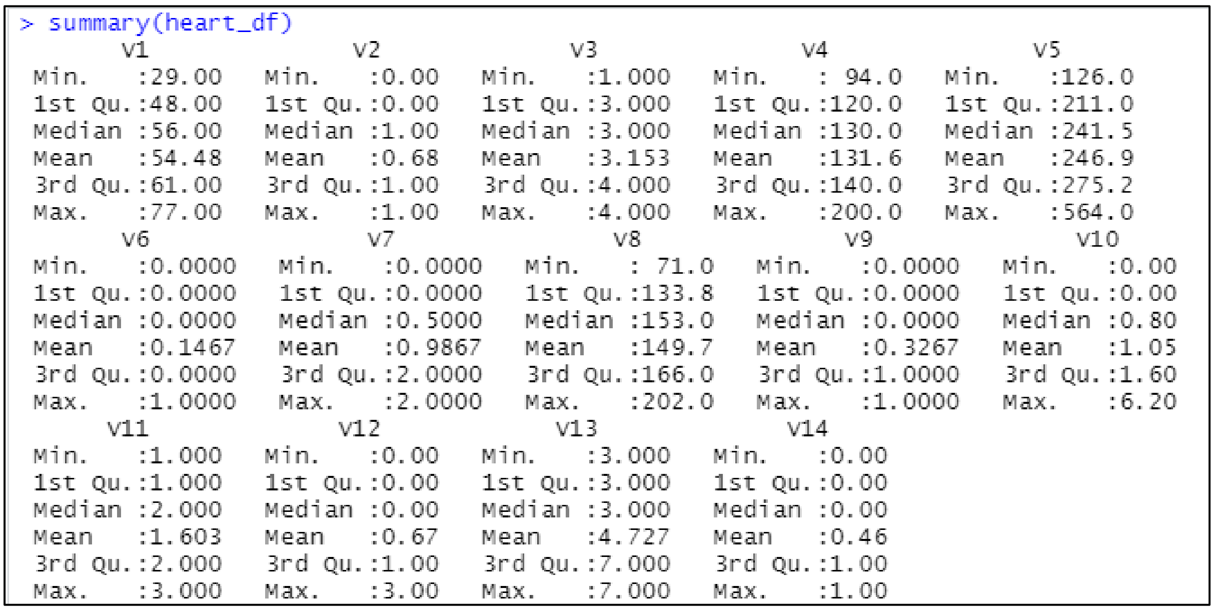
```
> anyNA(heart_df)
[1] FALSE
```

Figure 4: The output of the anyNA() function.

The output FALSE in the fig.4 shows there is no missing values in the *heart_df* dataframe

3.3 Dataset summarized details

We use the summary() function to check the summarized details of our dataframe. It gives the descriptive statistical values such as mean, median, minimum, maximum, 1st and 3rd quantile of each variable.

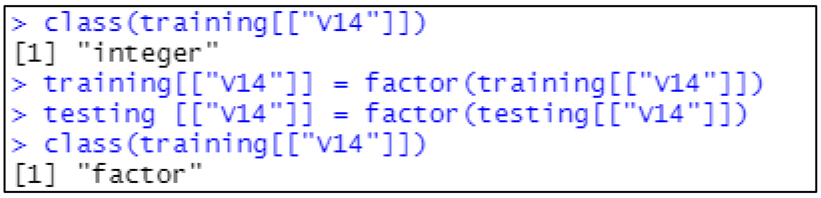


```
> summary(heart_df)
       V1              V2              V3              V4              V5
 Min.   :29.00   Min.   :0.00   Min.   :1.000   Min.   : 94.0   Min.   :126.0
 1st Qu.:48.00   1st Qu.:0.00   1st Qu.:3.000   1st Qu.:120.0   1st Qu.:211.0
 Median :56.00   Median :1.00   Median :3.000   Median :130.0   Median :241.5
 Mean   :54.48   Mean   :0.68   Mean   :3.153   Mean   :131.6   Mean   :246.9
 3rd Qu.:61.00   3rd Qu.:1.00   3rd Qu.:4.000   3rd Qu.:140.0   3rd Qu.:275.2
 Max.   :77.00   Max.   :1.00   Max.   :4.000   Max.   :200.0   Max.   :564.0
       V6                V7               V8               V9               V10
 Min.   :0.0000   Min.   :0.0000   Min.   : 71.0   Min.   :0.0000   Min.   :0.00
 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:133.8   1st Qu.:0.0000   1st Qu.:0.00
 Median :0.0000   Median :0.5000   Median :153.0   Median :0.0000   Median :0.80
 Mean   :0.1467   Mean   :0.9867   Mean   :149.7   Mean   :0.3267   Mean   :1.05
 3rd Qu.:0.0000   3rd Qu.:2.0000   3rd Qu.:166.0   3rd Qu.:1.0000   3rd Qu.:1.60
 Max.   :1.0000   Max.   :2.0000   Max.   :202.0   Max.   :1.0000   Max.   :6.20
      V11             V12             V13             V14
 Min.   :1.000   Min.   :0.00   Min.   :3.000   Min.   :0.00
 1st Qu.:1.000   1st Qu.:0.00   1st Qu.:3.000   1st Qu.:0.00
 Median :2.000   Median :0.00   Median :3.000   Median :0.00
 Mean   :1.603   Mean   :0.67   Mean   :4.727   Mean   :0.46
 3rd Qu.:2.000   3rd Qu.:1.00   3rd Qu.:7.000   3rd Qu.:1.00
 Max.   :3.000   Max.   :3.00   Max.   :7.000   Max.   :1.00
```

Figure 5: Represents the summary statistics of the dataframe

From fig.5, we can observe that all the variables has different range of values and we need to standardize our data before proceeding for model building. We can either use the scale() function in the caTools or use the preProcess() function in the caret package to standardize the data.

Since the values in our target variable is binary with 0 and 1, it should be a categorical variable. We use the factor() function to convert the target variable V14 into factor data structure.

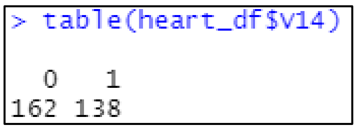


```
> class(training[["V14"]])
[1] "integer"
> training[["V14"]] = factor(training[["V14"]])
> testing [["V14"]] = factor(testing[["V14"]])
> class(training[["V14"]])
[1] "factor"
```

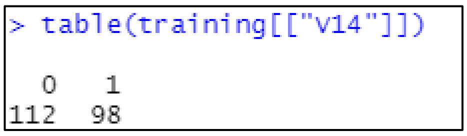
Figure 6: Represents the class of the target variable V14

From Fig.6, we can understand that the class of the target variable V14 has changed from integer to factor.

We can use the table() function to check the number of 0's and 1's in the target variable V14



```
> table(heart_df$V14)

  0   1
162 138
```



```
> table(training[["V14"]])

  0   1
112  98
```

From the above, we can interpret that out of the 300 observations, the absence and the presence of the heart disease are 162 and 138, which is 54% and 46% respectively. Similarly, with the training data of 210 observations, the absence and the presence of the heart disease are 112 and 98, which is 53% and 47% respectively. We can say the distribution of the target variable in the training data almost matches with the total dataset.

4. Model Building

The model building is performed using the train() function in the caret package. This function supports various classification and regression algorithm by setting up sets up a grid of tuning parameters. The fitted model is used for predicting the target variable by inputing predictor values.

To control the computational nuances of the train() method, we use the trainControl() method that generates parameters that further control how models are created. We are setting 3 parameters of trainControl() method. The “method” parameter holds the details about resampling method. We can set “method” with many values like "boot", "cv", "LOOCV", "LGOCV", "repeatedcv", "timeslice", "none" and "oob". For this project, we use repeatedcv, which repeatedly performs X-fold cross-validation on the training data.

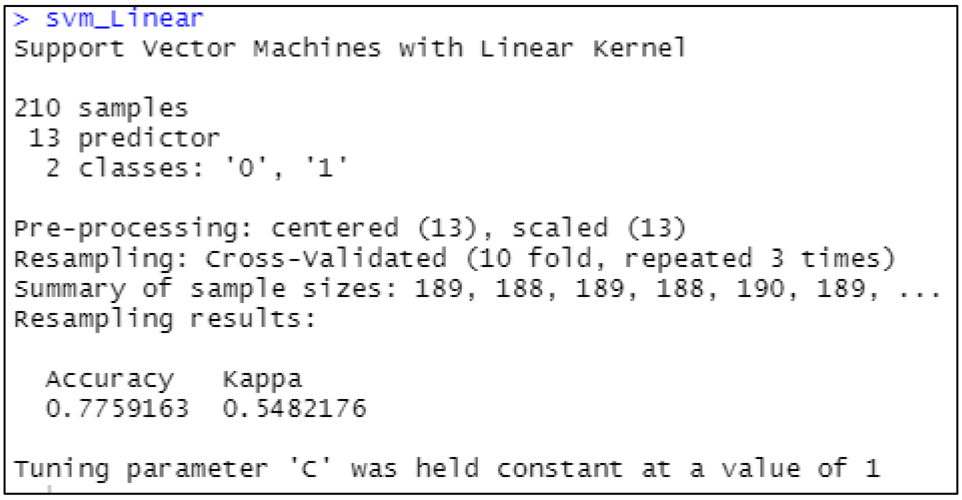
The “number” parameter controls with the number of folds in K-fold cross-validation. The “repeats” parameter contains the complete sets of folds to compute for our repeated cross-validation. Since we are using number =10 and repeats =3, it will perform 10-fold cross-validation on the training data 3 times, using a different set of folds for each cross-validation. This trainControl() methods returns a list which will be pass on our train() method.

4.1 SVM Classifier using Linear Kernel

First, let’s create a model for SVM linear classifier with the assumption that the training data are linearly separable. A hyperplane should be at a maximum distance from the nearest data point of either class. For training SVM linear classifier, we should pass “svmLinear” with “method” parameter in train() function. The “V14~.” denotes a formula for using all attributes in our classifier and V14 as the target variable. The “trControl” parameter holds the list of values that controls the train() function. It should be passed with results from our trainControl() method.

The data pre-processing can be carried out by passing desired values in the preProcess arugument in the train() function of the caret package. The scale transform calculates the standard deviation for every variable and divides each value by it's standard deviation. The center transform calculates the mean for every variable and subtracts it from each value. By combining both scale and center transforms, the varibles are standarding a mean value of 0 and a standard deviation of 1. (Nselvar, 2022)

The "tuneLength" parameter holds an integer value that represents the number of levels for each tuning parameters that should be generated by train. By setting the tuneLength parameter as 10, it allows to pick 10 arbitrary values for the C, the "cost" of the linear kernel. The tuneLength parameter controls the complexity of the boundary between support vectors.



```
> svm_Linear
Support Vector Machines with Linear Kernel

210 samples
 13 predictor
  2 classes: '0', '1'

Pre-processing: centered (13), scaled (13)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 189, 188, 189, 188, 190, 189, ...
Resampling results:

  Accuracy   Kappa
  0.7759163  0.5482176

Tuning parameter 'C' was held constant at a value of 1
```

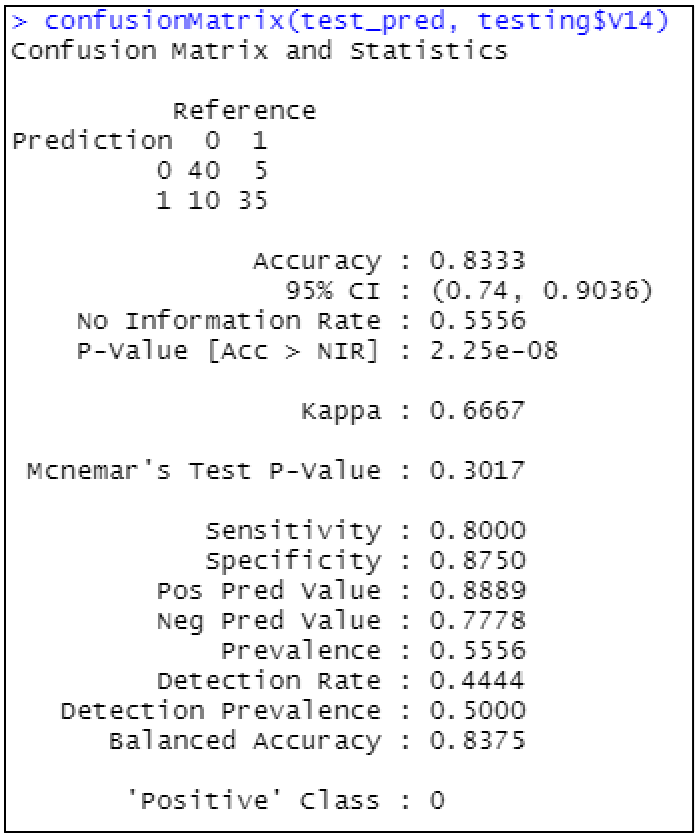
Figure 7: The output of the SVM linear fit model

From fig.7, we understand that the support vector machine model was created with the linear kernel. The testing data used for creating this model contains 210 samples with 13 predictor variables. The target variable has 2 classes which are '0' and '1'. The centre and scale transformation were performed on 13 predictor variables. The resampling was done with10-fold cross-validation on the training data 3 times, using a different set of folds for each cross-validation. The resulting SVM linear model has an accuracy of 77.59% and the kappa value of 0.5482176. By default, the tuning parameter 'C' was held constant at a value of 1. The cost, C is a hypermeter which is set before the training model and used to control error.

4.2 Testing the SVM linear classifier

To test the prediction of the SVM linear model, we use the predict() function which is part of the caret package. We will be passing 2 arguments. It's first parameter is our trained model 'svm_Linear' and second parameter "newdata" holds our testing data frame. The predict() method returns a list which is stored in a test_pred variable. The test_pred has the output of "0" and "1" which is the predicted class for the test dataset.

To check the model's test accuracy, we use the confusionMatrix() function in R which shows a cross-tabulation of the observed and predicted classes..



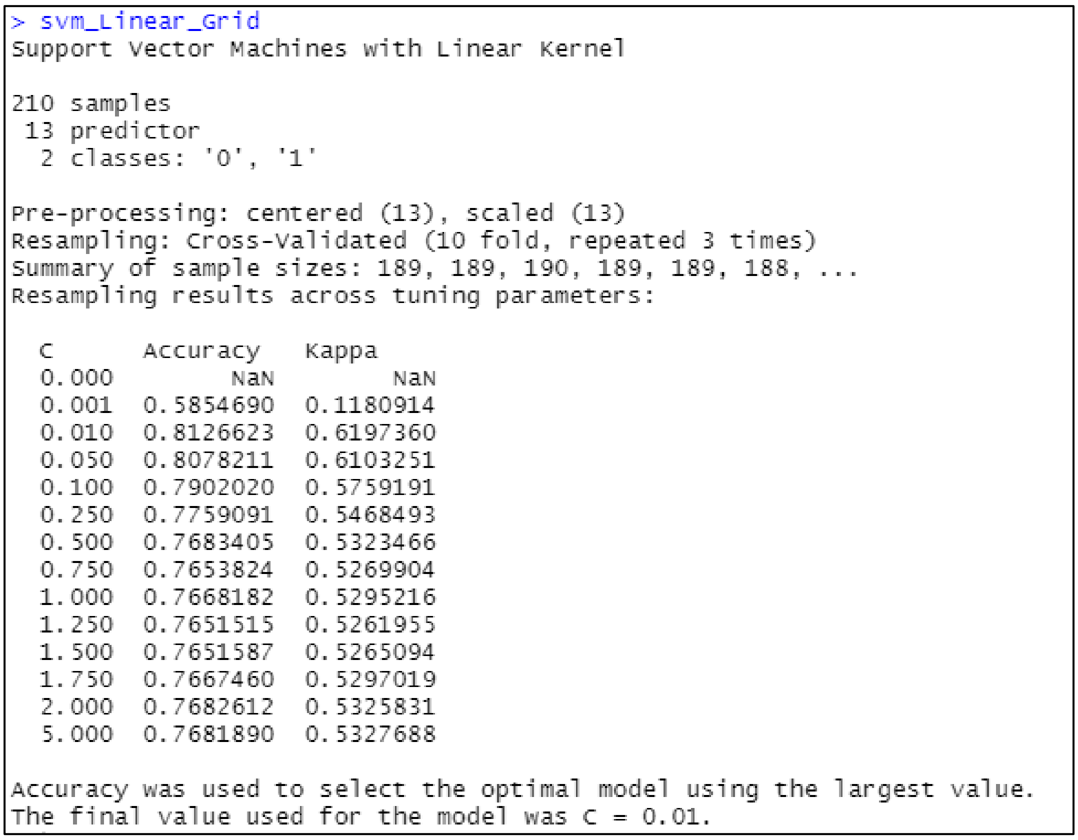
```
> confusionMatrix(test_pred, testing$V14)
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 40  5
         1 10 35

               Accuracy : 0.8333
                 95% CI : (0.74, 0.9036)
    No Information Rate : 0.5556
    P-Value [Acc > NIR] : 2.25e-08

                  Kappa : 0.6667

 Mcnemar's Test P-Value : 0.3017

            Sensitivity : 0.8000
            Specificity : 0.8750
         Pos Pred Value : 0.8889
         Neg Pred Value : 0.7778
             Prevalence : 0.5556
         Detection Rate : 0.4444
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.8375

       'Positive' Class : 0
```

Figure 8: The confusion matrix output of the test data for SVM linear fit model

From fig.8, we can observe that out of the 90-test observation, the absence and presence of the heart disease is 45 observations each. In the absence of the heart disease class, 89% data are correctly classified whereas in presence of the heart disease class only 78% data are correctly classified. Th model accuracy is 83.33%. The 95% confidence interval lies between 0.74 and 0.9036. The The "no-information rate" which is the largest proportion of the observed classes is 0.5556. McNemar's P-value test indicates that the accuracy of the model is higher than NIR. The Kappa value is 0.6667.

4.3 Tuning the SVM linear classifier

We can tune the model for better accuracy by using the tuneGrid parameter to do some sensitivity analysis around the values C value(Cost) between 0 and 5 in Linear classifier. This can be done by inputting values in grid search. The next code snippet will show you, building & tuning of an SVM classifier with different values of C. The expand.grid() function is used to build a dataframe that contain all the combinations of C we want to look at. Next step is to use this data frame for testing our classifier at specific C values. It needs to be put in train() method with tuneGrid parameter(Revolutions, 2022).



```
> svm_Linear_Grid
Support Vector Machines with Linear Kernel

210 samples
 13 predictor
  2 classes: '0', '1'

Pre-processing: centered (13), scaled (13)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 189, 189, 190, 189, 189, 188, ...
Resampling results across tuning parameters:

  C      Accuracy   Kappa
  0.000        NaN        NaN
  0.001  0.5854690  0.1180914
  0.010  0.8126623  0.6197360
  0.050  0.8078211  0.6103251
  0.100  0.7902020  0.5759191
  0.250  0.7759091  0.5468493
  0.500  0.7683405  0.5323466
  0.750  0.7653824  0.5269904
  1.000  0.7668182  0.5295216
  1.250  0.7651515  0.5261955
  1.500  0.7651587  0.5265094
  1.750  0.7667460  0.5297019
  2.000  0.7682612  0.5325831
  5.000  0.7681890  0.5327688

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was C = 0.01.
```

Figure 9: The output of the SVM linear grid fit model

Since, we have used the dataframe grid to impute the values of C in the range of 0 to 5, we obtain the accuracy and kappa value for each of the cost value. The optimal model is obtained for the cost value with highest accuracy. From fig.9, we can observe that for the C value of 0.01, the maximum accuracy of 81.26% is attained.

The plot of the svm_Linear_Grid shows that the maximum accuracy is reached when c=0.01.

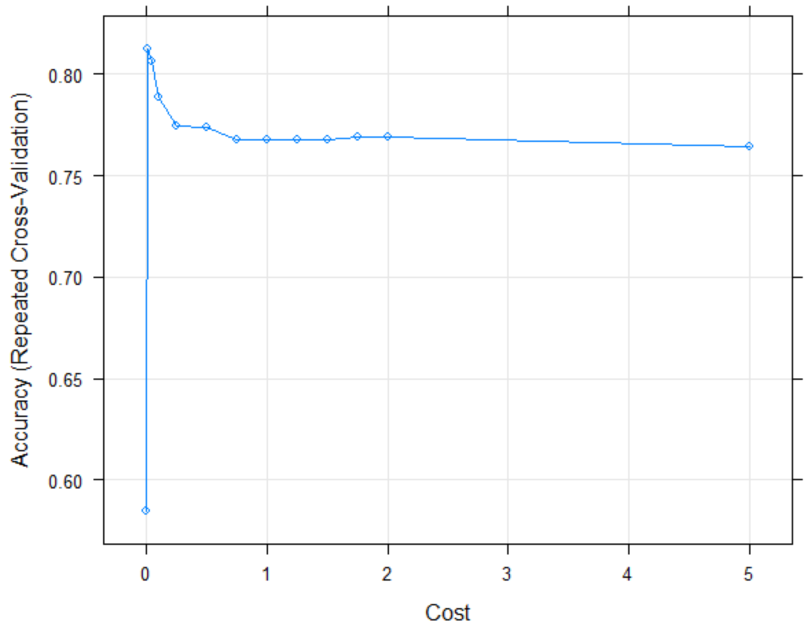
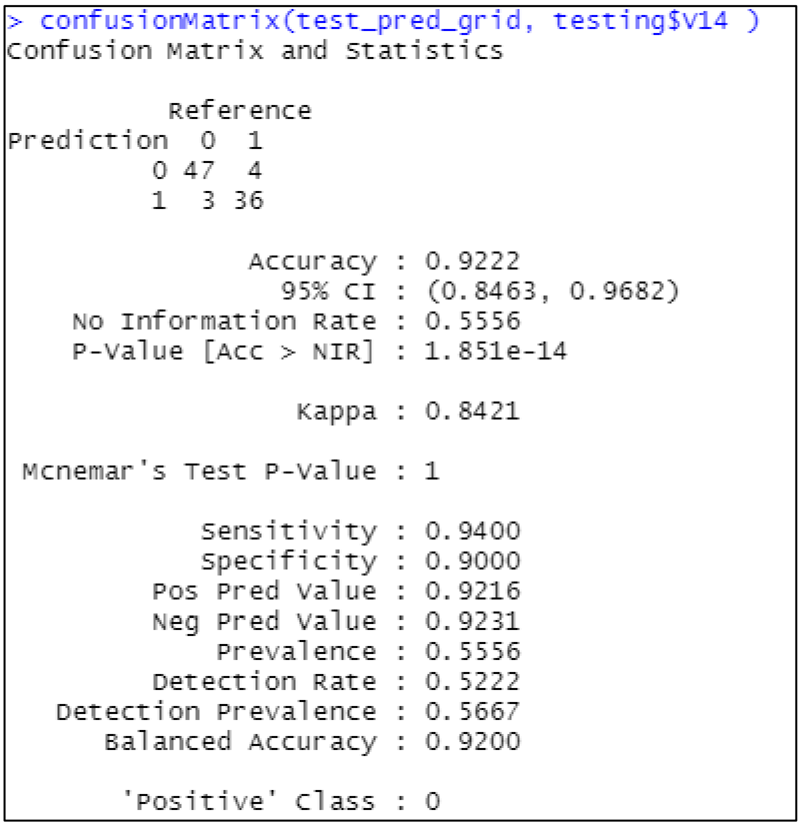


Figure 10: Plot of Cost versus accuracy (repeated cross-validation) for SVM Linear

We tested the prediction model of svm_Linear_Grid with our test data using the predict() function.



```
> confusionMatrix(test_pred_grid, testing$V14 )
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 47  4
         1  3 36

               Accuracy : 0.9222
                 95% CI : (0.8463, 0.9682)
    No Information Rate : 0.5556
    P-Value [Acc > NIR] : 1.851e-14

                  Kappa : 0.8421

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9400
            Specificity : 0.9000
         Pos Pred Value : 0.9216
         Neg Pred Value : 0.9231
             Prevalence : 0.5556
         Detection Rate : 0.5222
   Detection Prevalence : 0.5667
      Balanced Accuracy : 0.9200

       'Positive' Class : 0
```

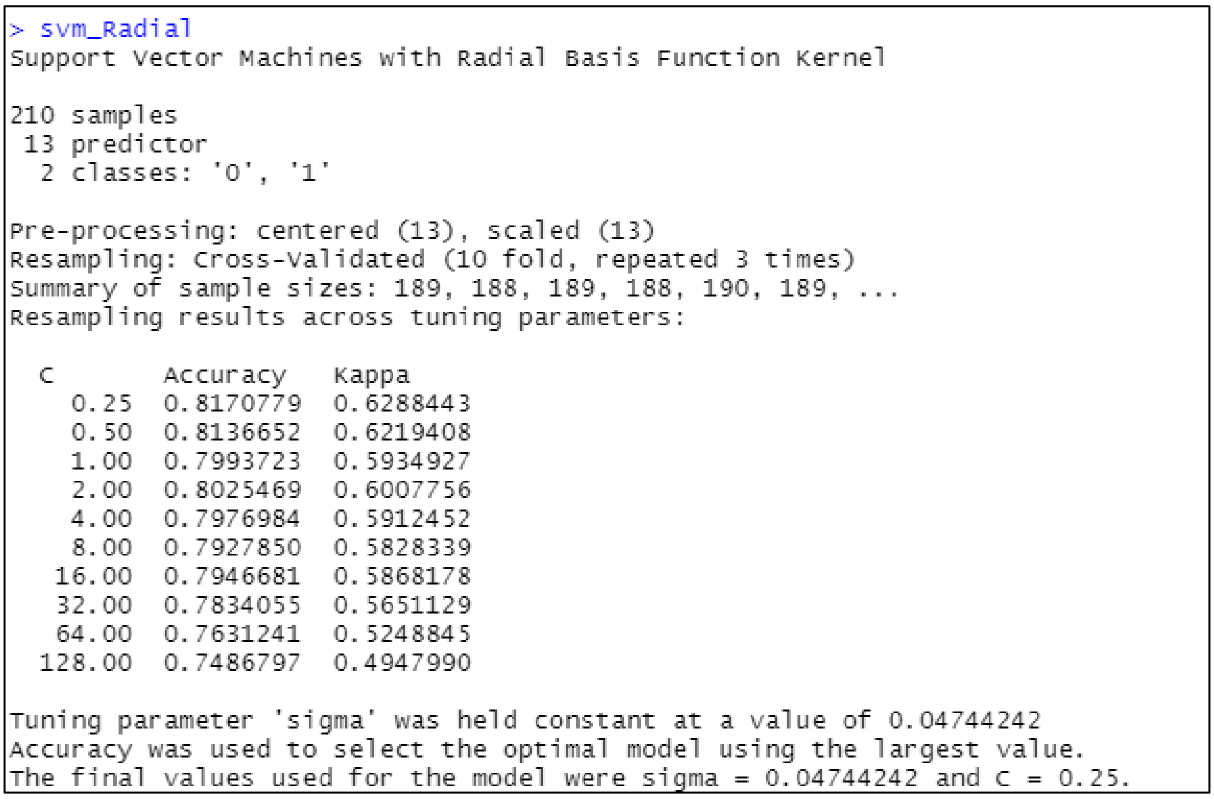
Figure 11: The confusion matrix output of the test data for SVM linear grid fit model

The confusion matrix calculated for SVM Linear Grid model shows an accuracy of 92.22%. we can observe that out of the 90-test observation, the absence and presence of the heart disease are 51 and 39 observations respectively. In both absence and presence of the heart disease class, 92% data are correctly classified. The 95% confidence interval lies between 0.8463 and 0.9682. The Kappa value obtained is 0.8421.

We can observe that the accuracy of the SVM linear model has increased from 83.33% to 92.22% by using the Cost value of 0.01 instead of the default value 1.

5. SVM Classifier using Non-Linear Kernel

We will now try to build a SVM classifier model using Non-Linear Kernel like Radial Basis and Polynomial Function. For radial basis function, we draw completely non-linear hyperplanes between the 2 classes. For distance metric, squared euclidean distance is used. For using RBF kernel, we need to pass "svmRadial" as "method" parameter in train() function. Selecting the right Cost "C" and "sigma" parameter is critical for Radial kernel.



```
> svm_Radial
Support Vector Machines with Radial Basis Function Kernel

210 samples
 13 predictor
  2 classes: '0', '1'

Pre-processing: centered (13), scaled (13)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 189, 188, 189, 188, 190, 189, ...
Resampling results across tuning parameters:

  C       Accuracy   Kappa
    0.25  0.8170779  0.6288443
    0.50  0.8136652  0.6219408
    1.00  0.7993723  0.5934927
    2.00  0.8025469  0.6007756
    4.00  0.7976984  0.5912452
    8.00  0.7927850  0.5828339
   16.00  0.7946681  0.5868178
   32.00  0.7834055  0.5651129
   64.00  0.7631241  0.5248845
  128.00  0.7486797  0.4947990

Tuning parameter 'sigma' was held constant at a value of 0.04744242
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.04744242 and C = 0.25.
```

Figure 12: The output of SVM Radial fit model

From fig.12, we can understand that we have used the same training dataset, pre-processing and traincontrol parameters that we used for SVM linear. The SVM RBF kernel calculates the variation and provides the optimal values of cost(C) and sigma. From the above output, we can say the best value of tuning parameter, sigma is 0.04744242 and it was held constant for the various values of the cost ranging from 0.25 to 128. The optimal model is chosen for the cost value whose accuracy is the largest. Since the accuracy of 81.70% is obtained when the cost value is 0.25, we can say the best cost value, C = 0.25.

We can plot the output of the SVM radial using the plot() function. The cost versus accuracy plot shows when the cost value, C = 0.25.

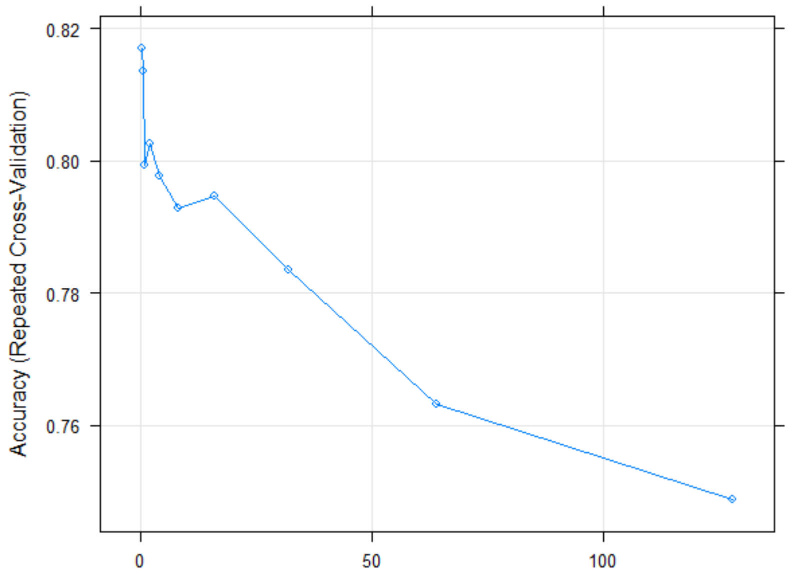
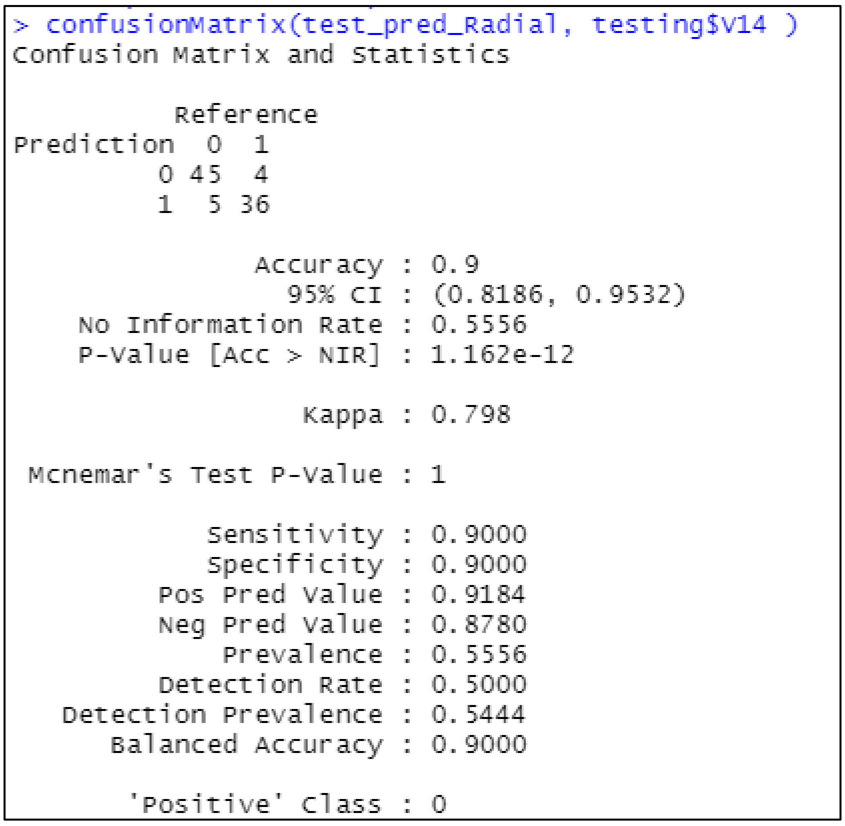


Figure 13: Plot of Cost versus accuracy (repeated cross-validation) for SVM Radial

We tested the prediction model of SVM Radial model with our test data using the predict() function. Confusion matrix is used to check the model's test accuracy and it shows a cross-tabulation of the observed and predicted classes.

The confusion matrix calculated for SVM Radial model shows an accuracy of 90%. We can observe that out of the 90-test observation, the absence and presence of the heart disease are 49 and 41 observations respectively. In the absence of the heart disease class, 92% data are correctly classified whereas in presence of the heart disease class only 88% data are correctly classified.. The 95% confidence interval lies between 0.8186and 0.9532. The Kappa value obtained is 0.798.



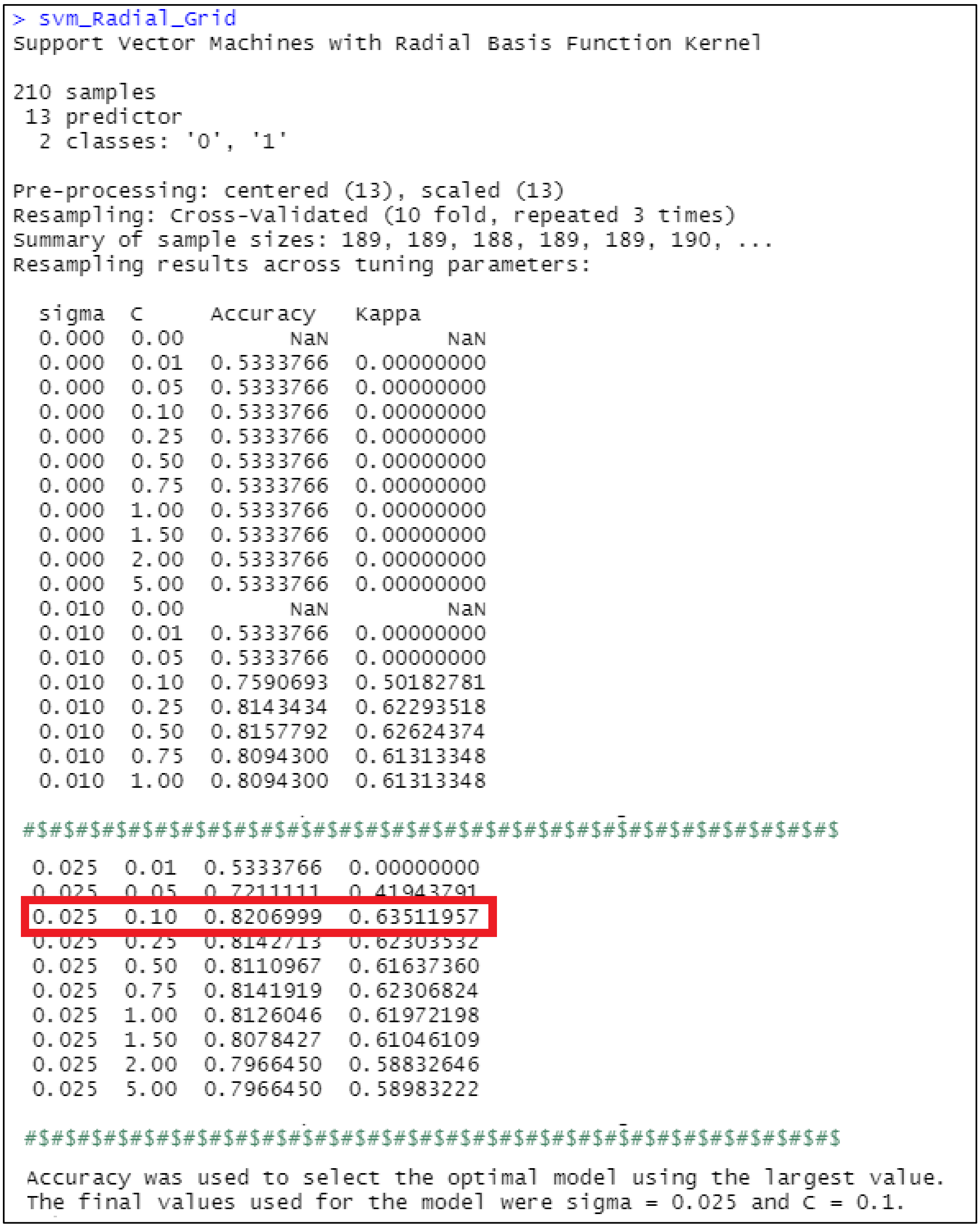
```
> confusionMatrix(test_pred_Radial, testing$V14 )
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 45  4
         1  5 36

               Accuracy : 0.9
                 95% CI : (0.8186, 0.9532)
    No Information Rate : 0.5556
    P-Value [Acc > NIR] : 1.162e-12

                  Kappa : 0.798

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9000
            Specificity : 0.9000
         Pos Pred Value : 0.9184
         Neg Pred Value : 0.8780
             Prevalence : 0.5556
         Detection Rate : 0.5000
   Detection Prevalence : 0.5444
      Balanced Accuracy : 0.9000

       'Positive' Class : 0
```

Figure 14: The confusion matrix output of the test data for SVM Radial fit model

We use train()'s tuneGrid parameter to test and tune our classifier with different values C and sigma to produce the model with the best accuracy. The expand.grid() function is used to build a dataframe contain all the combinations of C and sigma that we want to check on the model. The variable grid_radial dataframe will hold values of sigma & C. Value of grid_radial will be passed to train() method's tuneGrid parameter.

We have checked the SVM Radial model with below values of cost and sigma.
sigma = 0,0.01, 0.02, 0.025, 0.03, 0.04, 0.05, 0.06, 0.07,0.08, 0.09, 0.1, 0.25, 0.5, 0.75,0.9),
cost = 0,0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.5, 2,5

The optimal model is chosen for the cost ad sigma value that has high accuracy. Based on the below output the final SVM Radial Grid classifier was modelled by using the values sigma = 0.025 and C = 0.1.



```
> svm_Radial_Grid
Support Vector Machines with Radial Basis Function Kernel

210 samples
 13 predictor
  2 classes: '0', '1'

Pre-processing: centered (13), scaled (13)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 189, 189, 188, 189, 189, 190, ...
Resampling results across tuning parameters:

  sigma  C     Accuracy   Kappa
  0.000  0.00        NaN         NaN
  0.000  0.01  0.5333766  0.00000000
  0.000  0.05  0.5333766  0.00000000
  0.000  0.10  0.5333766  0.00000000
  0.000  0.25  0.5333766  0.00000000
  0.000  0.50  0.5333766  0.00000000
  0.000  0.75  0.5333766  0.00000000
  0.000  1.00  0.5333766  0.00000000
  0.000  1.50  0.5333766  0.00000000
  0.000  2.00  0.5333766  0.00000000
  0.000  5.00  0.5333766  0.00000000
  0.010  0.00        NaN         NaN
  0.010  0.01  0.5333766  0.00000000
  0.010  0.05  0.5333766  0.00000000
  0.010  0.10  0.7590693  0.50182781
  0.010  0.25  0.8143434  0.62293518
  0.010  0.50  0.8157792  0.62624374
  0.010  0.75  0.8094300  0.61313348
  0.010  1.00  0.8094300  0.61313348

 #$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$

  0.025  0.01  0.5333766  0.00000000
  0.025  0.05  0.7211111  0.41943791
  0.025  0.10  0.8206999  0.63511957
  0.025  0.25  0.8142713  0.62303532
  0.025  0.50  0.8110967  0.61637360
  0.025  0.75  0.8141919  0.62306824
  0.025  1.00  0.8126046  0.61972198
  0.025  1.50  0.8078427  0.61046109
  0.025  2.00  0.7966450  0.58832646
  0.025  5.00  0.7966450  0.58983222

 #$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$

 Accuracy was used to select the optimal model using the largest value.
 The final values used for the model were sigma = 0.025 and C = 0.1.
```

Figure 15: The confusion matrix output of the test data for SVM Radial Grid fit model

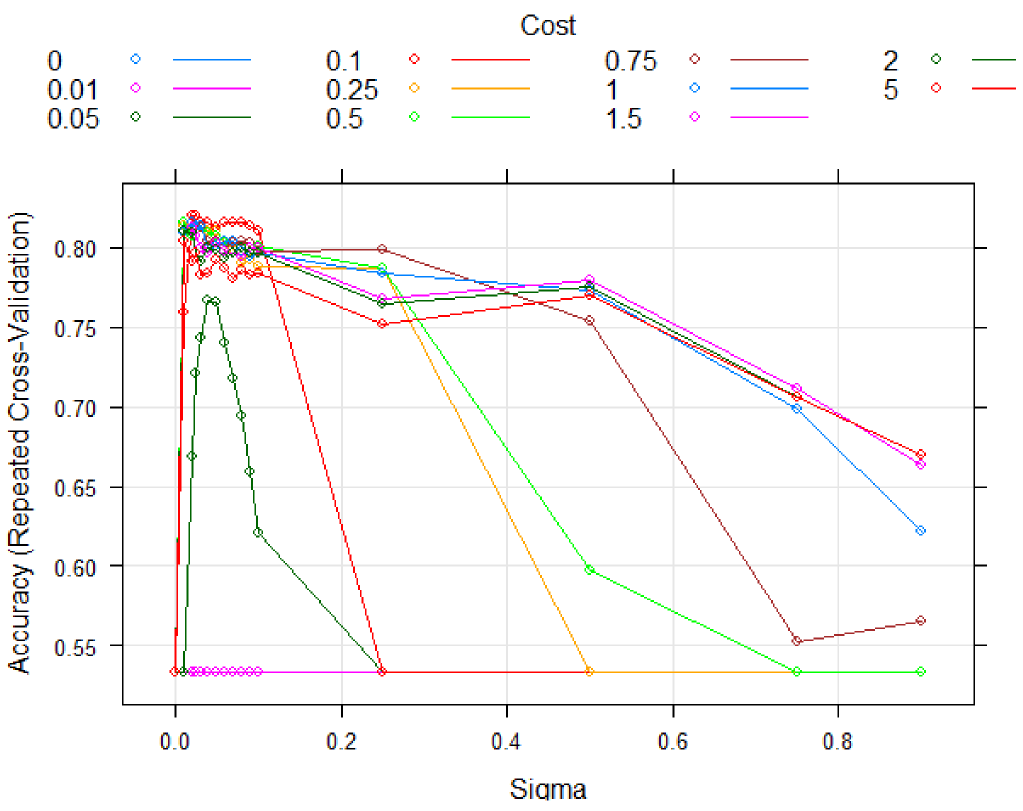


Figure 15: Plot of Cost versus accuracy (repeated cross-validation) for SVM Radial Grid model.

In the fig.15, the colour coded lines represent the individual cost value. We can observe that the maximum accuracy is attained for most of the cost value when the sigma is between 0.01 and 0.1. When sigma is 0.0, the accuracy remains low and constant at 0.5333766 for each of the cost value. Similarly, when the cost is 0.05, the accuracy remains constant at 0.5333766 when the sigma value is above 0.2.

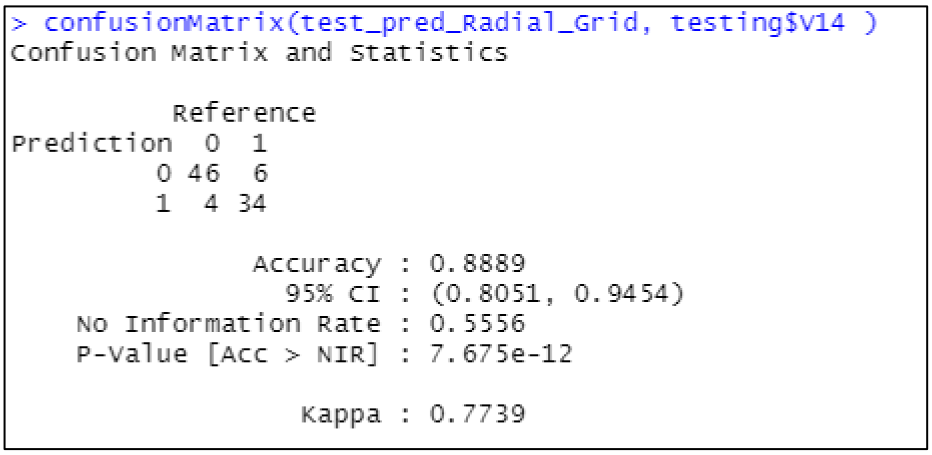
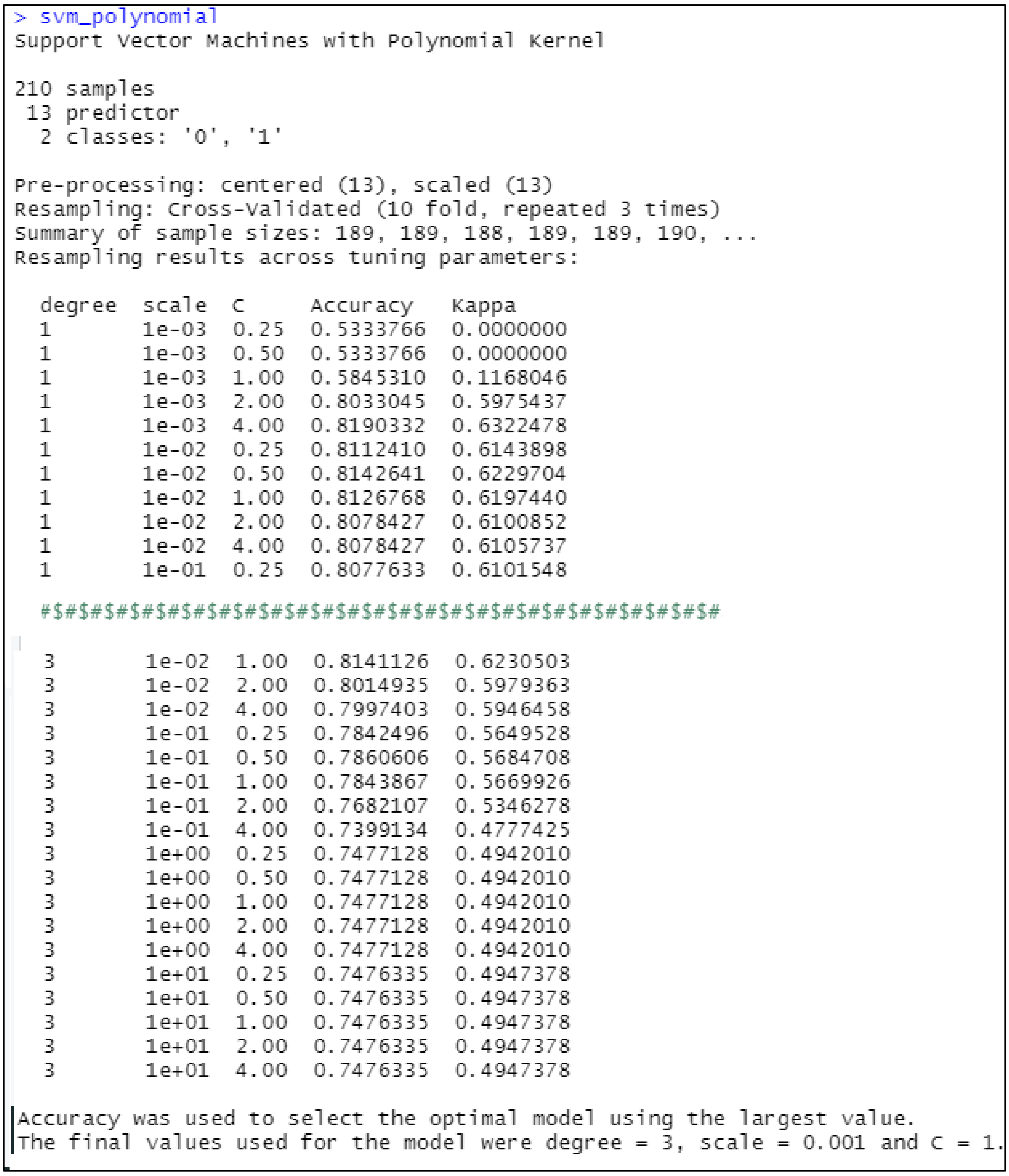

```
> confusionMatrix(test_pred_Radial_Grid, testing$V14 )
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 46  6
         1  4 34
                                          
               Accuracy : 0.8889
                 95% CI : (0.8051, 0.9454)
    No Information Rate : 0.5556
    P-Value [Acc > NIR] : 7.675e-12
                                          
                  Kappa : 0.7739
```

Figure 15: The output of confusion matrix for SVM Radial Grid model

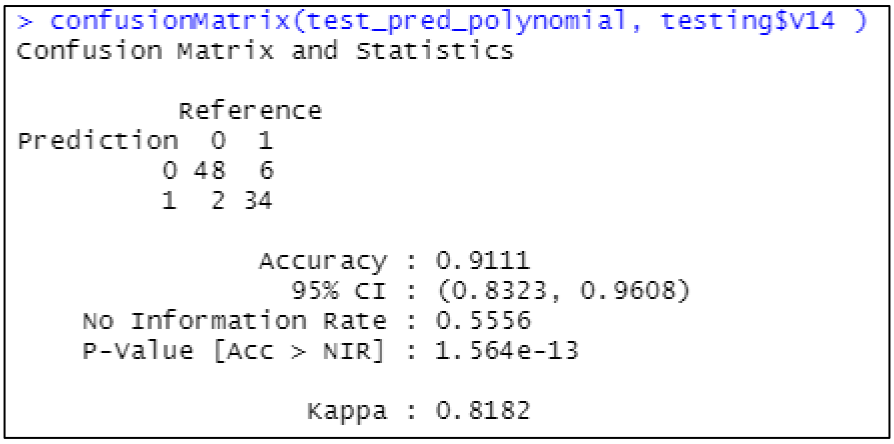
5.2. SVM Classifier using Polynomial Kernel

Polynomial Kernel is a general representation of kernels with a degree of more than one. For using polynomial kernel, we need to pass “svmPoly” as “method” parameter in train() function.



```
> svm_polynomial
Support Vector Machines with Polynomial Kernel

210 samples
 13 predictor
  2 classes: '0', '1'

Pre-processing: centered (13), scaled (13)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 189, 189, 188, 189, 189, 190, ...
Resampling results across tuning parameters:

  degree  scale  C     Accuracy   Kappa
  1       1e-03  0.25  0.5333766  0.0000000
  1       1e-03  0.50  0.5333766  0.0000000
  1       1e-03  1.00  0.5845310  0.1168046
  1       1e-03  2.00  0.8033045  0.5975437
  1       1e-03  4.00  0.8190332  0.6322478
  1       1e-02  0.25  0.8112410  0.6143898
  1       1e-02  0.50  0.8142641  0.6229704
  1       1e-02  1.00  0.8126768  0.6197440
  1       1e-02  2.00  0.8078427  0.6100852
  1       1e-02  4.00  0.8078427  0.6105737
  1       1e-01  0.25  0.8077633  0.6101548

  #$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#

  3       1e-02  1.00  0.8141126  0.6230503
  3       1e-02  2.00  0.8014935  0.5979363
  3       1e-02  4.00  0.7997403  0.5946458
  3       1e-01  0.25  0.7842496  0.5649528
  3       1e-01  0.50  0.7860606  0.5684708
  3       1e-01  1.00  0.7843867  0.5669926
  3       1e-01  2.00  0.7682107  0.5346278
  3       1e-01  4.00  0.7399134  0.4777425
  3       1e+00  0.25  0.7477128  0.4942010
  3       1e+00  0.50  0.7477128  0.4942010
  3       1e+00  1.00  0.7477128  0.4942010
  3       1e+00  2.00  0.7477128  0.4942010
  3       1e+00  4.00  0.7477128  0.4942010
  3       1e+01  0.25  0.7476335  0.4947378
  3       1e+01  0.50  0.7476335  0.4947378
  3       1e+01  1.00  0.7476335  0.4947378
  3       1e+01  2.00  0.7476335  0.4947378
  3       1e+01  4.00  0.7476335  0.4947378

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were degree = 3, scale = 0.001 and C = 1.
```

The final SVM Polynomial classifier is modelled using the values degree = 3, scale = 0.001 and C = 1.



```
> confusionMatrix(test_pred_polynomial, testing$V14 )
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 48  6
         1  2 34

               Accuracy : 0.9111
                 95% CI : (0.8323, 0.9608)
    No Information Rate : 0.5556
    P-Value [Acc > NIR] : 1.564e-13

                  Kappa : 0.8182
```

Figure 16: The output of confusion matrix for SVM Ploynomial model

The confusion matrix of SVM Ploynomial model shows the accuracy of 91.11%.

5.3. Comparison between SVM models

Let's compare the results of the SVM Linear, Radial and Polynomial models. The test accuracy of each model after tuning is tabled below

SVM Model	**Accuracy in %**
SVM Linear	92.22%
SVM Radial	90%
SVM Polynomial	91.11%

We can understand that for the given heart disease dataset, SVM Linear classifer can be used as a best prediction model since it has the accuracy of 92.22%.

6. Conclusion

Based on the model output, we can say the support vector machine (SVM) is a promising classifier approach for predicting the presence of heart disease with the given prediction variables. Majority of the variables in the dataset contribute significantly for building of a SVM predictive model with good accuracy rate (>= 90%). SVM Linear classifer has the highest accuracy when compared to Radial and Polynomial classifer which may be due to overfitting.

References

1. The 5th Tribe, Support Vector Machines and caret. (n.d.). Revolutions. Retrieved June 22, 2022, from https://blog.revolutionanalytics.com/2015/10/the-5th-tribe-support-vector-machines-and-caret.html
2. Bank Telemarketing NN & SVM. (n.d.). Nselvar.github.io. Retrieved June 22, 2022, from https://nselvar.github.io/bank-telemarketing/