Heart disease classification

Luca Bartolomei

27 aprile, 2021

Contents

1 Introduction

This project has the aim to analyze the Heart Disease dataset to build a classifiers to predict whether people have heart disease or not.

The dataset used for project comes from Kaggle website: https://www.kaggle.com/ronitf/heart-disease-uci

Attribute Information:

1. age: The person's age in years
2. sex: The person's sex
   - 1 = male,
   - 0 = female
3. cp: The chest pain experienced
   - Value 1: typical angina,
   - Value 2: atypical angina,
   - Value 3: non-anginal pain,
   - Value 4: asymptomatic
4. trestbps: The person's resting blood pressure (mm Hg on admission to the hospital)
5. chol: The person's cholesterol measurement in mg/dl
6. fbs: The person's fasting blood sugar
   - if > 120 mg/dl, 1 = true; 0 = false
7. restecg: Resting electrocardiographic measurement
   - 0 = normal,
   - 1 = having ST-T wave abnormality,
   - 2 = showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: The person's maximum heart rate achieved
9. exang: Exercise induced angina
   - 1 = yes;
   - 0 = no
10. oldpeak: ST depression induced by exercise relative to rest ('ST' relates to positions on the ECG plot)
11. slope: the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat,
    - Value 3: downsloping
12. ca: The number of major vessels (0-3)
13. thal: A blood disorder called thalassemia
    - 1 = normal;
    - 2 = fixed defect;
    - 3 = reversable defect
14. target: Heart disease
    - 0 = no
    - 1 = yes

In this project "target" are going to be the dependent variable. The others attributes are going to be the predictors.

This project will make a comparison between different machine learning algorithms in order to to assess the correctness in ***classifying*** data with respect to efficiency of each algorithm in terms of ***sensitivity*** and ***specificity***.

Sensitivity and ***specificity*** are widely used in medicine for binary classification test.

We are going to project a ***Machine Learning Classifiers***, more specifically a ***Binary classifiers*** where the two classes are:

1. yes: Heart disease
2. no: No Heart disease

2 Methods

2.1 Key concepts

Classification Classification is the process of predicting the class of given data points. When there are only two classes the problem is known as ***statistical binary classification***.

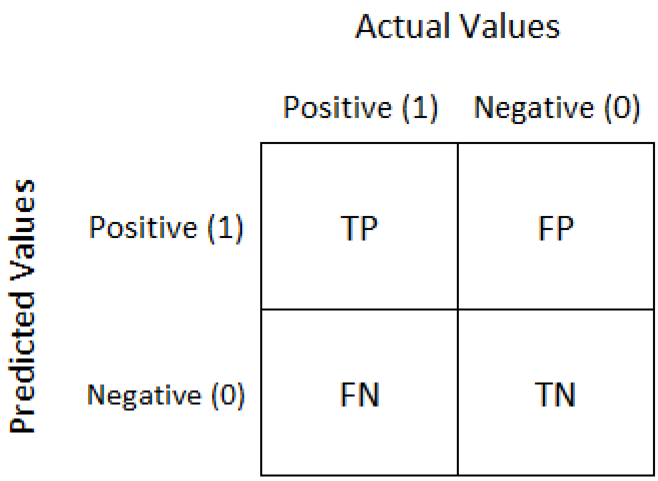
Accuracy and ***Kappa*** These are the default metrics used to evaluate algorithms on binary and multi-class classification datasets in caret.

Accuracy is the percentage of correctly classifies instances out of all instances. It is more useful on a binary classification than multi-class classification problems because it can be less clear exactly how the accuracy breaks down across those classes.

Sensitivity is the true positive rate also called the recall. It is the number instances from the positive (first) class that actually predicted correctly.

Specificity is also called the true negative rate. Is the number of instances from the negative class (second) class that were actually predicted correctly

Confusion Matrix Confusion Matrix is represented by the following table



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 1: Confusion matrix

1. TP: True positive
2. FP: False positive
3. FN: False negative
4. TN: True negative

A true positive is an outcome where the model correctly predicts the positive class.

A false positive is an outcome where the model incorrectly predicts the positive class.

A false negative is an outcome where the model incorrectly predicts the negative class.

A true negative is an outcome where the model correctly predicts the negative class.

Now we consider these rates

$$\text{TPR} = \frac{TP}{TP + FN}$$

$$\text{FPR} = \frac{FP}{TN + FP}$$

TPR (True Positive Rate, that correspond to Sensitivity) means how often does the classifier correctly predict positive.

FPR (False Positive Rate) means how often does the classifier incorrectly predict positive. FPR also correspond to: 100% - Specificity, where Specificity is the proportion of true negative who are correctly identified.

ROC (Receiver Operating Characteristic) Curve is a way to visualize the performance of a binary classifier.

The ROC metric is strictly connected with Confusion Matrix

The ROC metric is better explained by ROC curve where TPR is on y axis and TFR on x axis.

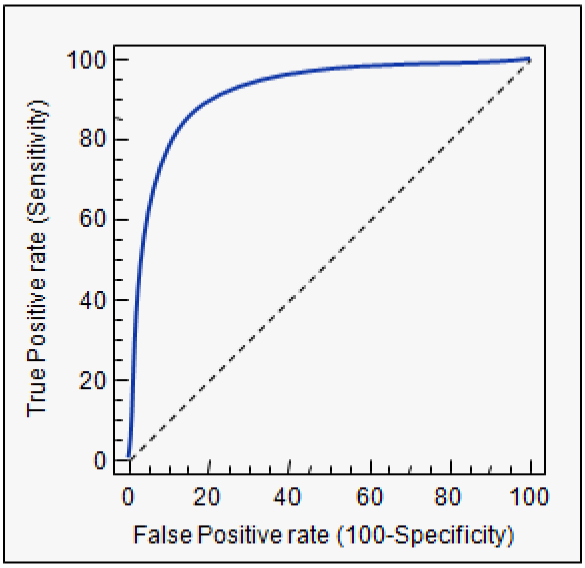


Figure 2: ROC curve

The dotted line correspond to the random classification, so a good classification is showed by any curve (like the blue one) up the dotted line.

The area under the ROC curve is also called: ***AUC*** (Area Under Curve)

Covariance of two statistical variables is a number that provides a measure of how much the two vary together.

Covariance matrix is a generalization of ***covariance*** to the case of dimension greater than two.

Supervised learning is a machine learning technique that aims to train a computer system in order to allow it to autonomously make predictions on the output values of a system versus an input based on a series of ideal examples, consisting of pairs input and output, which are initially provided..

Unsupervised learning is a type of machine learning that looks for previously undetected patterns in a dataset with no pre-existing labels and with a minimum of human supervision. Unlike supervised learning which usually makes use of human-labeled data, unsupervised learning allows for the modeling of probability densities on inputs.

The Bayes theorem describes the probability of an event, based on the preliminary knowledge of the conditions that could be related to the event. It serves as a way to understand ***conditional probability***.

The conditional probability of an event A with respect to an event B is the probability that A will occur, knowing that B has occurred. This probability, indicated $P(A|B)$ or $P_B(A)$, expresses a "correction" of expectations for A, dictated by observation of B.

Naive Bayes is a ***supervised learning algorithm*** suitable for solving binary (two-class) and multi-class classification problems. The main peculiarity of the algorithm, in addition to making use of ***the Bayes theorem***, is that it is based on the fact that all the characteristics are not related to each other.

Ensemble model is a combination of single simple models (called weak learners) that together create a new, more powerful model (called strong learner).

Boosting is an ***ensemble method*** for improving model predictions of a given learning algorithm. The idea behind it is to train weak learners sequentially, each trying to correct its predecessor.

Loss function is a method for estimating the quality of an algorithm in modeling the data provided. If the forecast deviates too much from actual results, the loss function produces a very large number.

Bagging is an ***ensemble model***. Is a general procedure that can be used to reduce the variance of those algorithms that have a high variance.

Collinearity is a linear association between two predictors. ***Multicollinearity*** is a situation where two or more predictors are highly linearly related that it can lead to misleading results when attempting to predict the dependent variable.

2.2 Model evaluation

2.2.1 Metrics To Evaluate Machine Learning Algorithms

The metric used for these project is ROC.

AUC is the area under the ROC curve and represents a models ability to discriminate between positive and negative classes. An area of 1.0 represents a model that made all predicts perfectly. An area of 0.5 represents a model as good as random.

ROC can be broken down into sensitivity and specificity. A binary classification problem is really a trade-off between sensitivity and specificity.

To calculate ROC information, in our trainControl we must set the summaryFunction
to twoClassSummary.

2.2.2 Principal component analysis

Principal component analysis (PCA) is a technique that, starting from a set of numerical variables, obtains a smaller set of “artificial” orthogonal variables. The reduced set of linear orthogonal projections (known as “principal components” or “principal components”, “PC”) is obtained by linearly combining the original variables in an appropriate manner.

In PCA, the term “information” indicates the total variability of the original input variables, ie the sum of the variances of the original variables. The central point of the PCA is the so-called spectral decomposition (also called the decomposition into eigenvalues and eigenvectors, or eigendecomposition) of the sample variance/covariance matrix. This decomposition returns the eigenvalues and eigenvalues of the ***covariance matrix***. The eigenvalues (in decreasing order of value) represent the amount of the total variability observed on the original variables, explained by each main component; the eigenvectors instead represent the corresponding (orthogonal) directions of maximum variability extracted from the principal components.

The hope in the application of the PCA is that the sample variances of the first Main Components (PC) are large, while the variances of the other components are small enough to consider the corresponding PCs negligible. A principal component variable that has little variability (relative to other variables) can be treated roughly as a constant. Omitting PCs with low sample variability and placing all attention on PCs with higher variance can be seen as a simple and “sensible” way to reduce the dimensionality (number of columns) of the dataset.

2.2.3 Adaptive Boosting

AdaBoost, acronym for “Adaptive Boosting” is a ***supervised learning algorithm*** proposed by Freund and Schapire in 1996. It was the first highly successful ***boosting*** algorithm developed for ***binary classification***.

It represents a popular boosting technique that combines multiple “weak classifiers” into one “strong classifier”.

2.2.4 Stochastic Gradient Boosting

Stochastic Gradient Boosting is another ***supervised learning algorithm*** for regression and classification problems.

It represents an ***ensemble model***. In each training cycle, or iteration, the weak learner is built and its predictions are compared with the correct result we expect.

The distance between observation and prediction represents the error rate of our model. These errors are defined using a ***loss function***.

The purpose of the algorithm is to minimize this ***loss function***, using a tool that is called “***gradient***”, which basically represents the partial derivative of the ***loss function***.

2.2.5 Classification and Regression Trees

Classification Trees is another ***supervised learning algorithm*** for regression and classification problems. Is based on concept of ***decision tree***.

A tree is a collection of entities called nodes connected to each other by arrows or lines. Each node contains a value and may or may not have child nodes, while arrows indicate the decisions/rules a tree may or may not make.

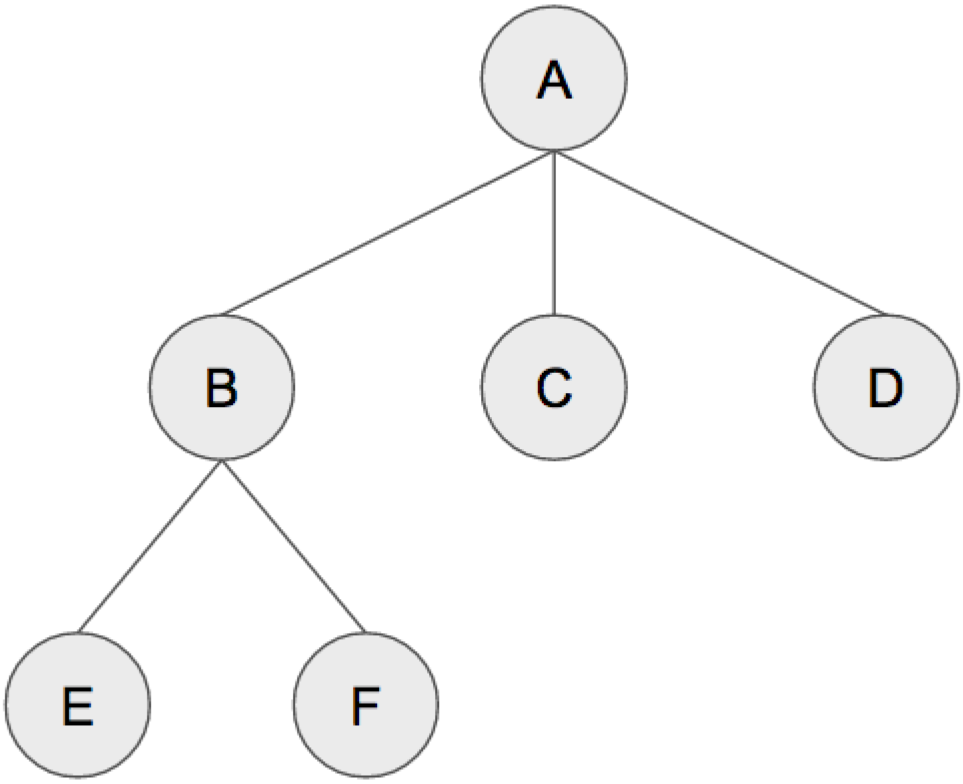


Figure 3: Decision tree

The nodes are A, B, C, D, E and F and are connected by lines that indicate kinship relations between the various nodes.

Node A is called the root node and is the starting point of the tree. It consists of three children: node B, node C and node D. The only node without a parent is the root node.

A tree is a ***binary tree*** when each parent has at most 2 child nodes. Decision trees are ***binary trees***.

An example of a binary tree is shown in the following image.

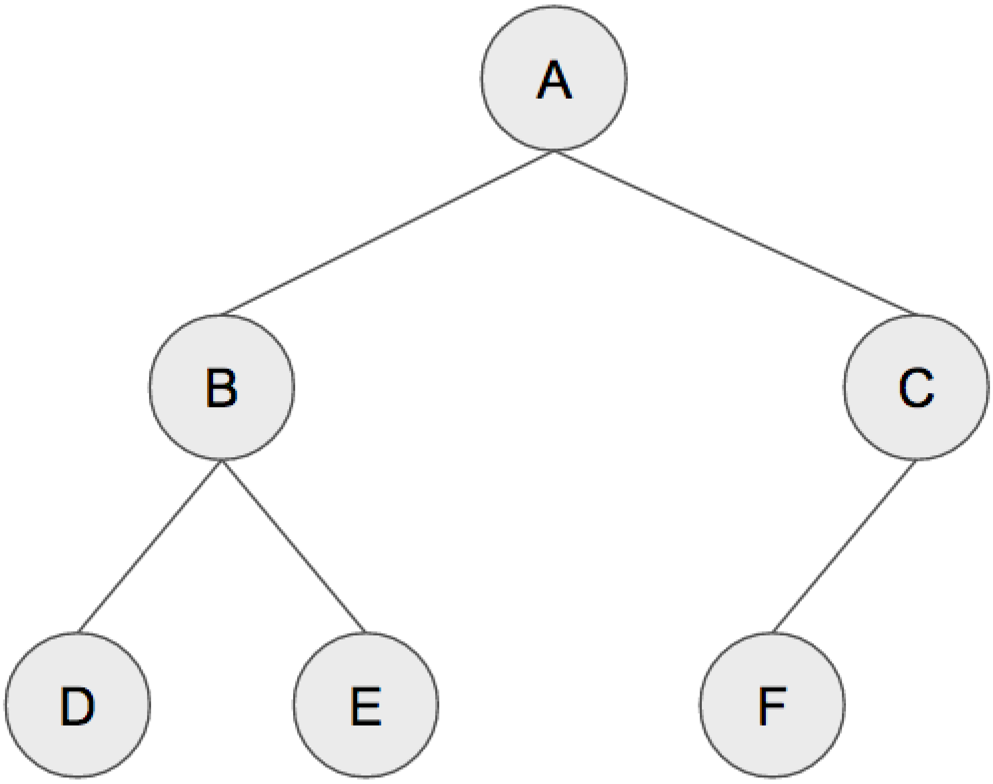


Figure 4: Binary tree

Classification and Regression Trees, also called CART, it is basically a binary tree.

2.2.6 Random Forest

Random Forest is another ***supervised learning algorithm***

It represents a type of ***ensemble model***, which uses ***bagging*** as an ensemble method and the decision tree as an individual model.

This means that a random forest combines many decision trees into one model. Individually, the predictions made by the decision trees may not be accurate, but combined together, the predictions will on average be closer to the outcome.

2.2.7 K Nearest Neighbor (KNN)

KNN is a ***supervised learning algorithm***, whose purpose is to predict a new instance by knowing the data points that are separated into different classes.

Its operation is based on the similarity of characteristics: the closer an instance is to a data point, the more the knn will consider them similar.

Similarity is usually calculated by Euclidean distance. The shorter the distance, the greater the similarity between the data point and the instance to be predicted.

In addition to the distance, the algorithm provides for setting a parameter k, chosen arbitrarily, which identifies the number of closest data points. The algorithm evaluates the k minimum distances thus obtained. The class that obtains the greatest number of these distances is chosen as the prediction.

2.2.8 Neural Network

A neural network is a model composed of artificial “neurons”, vaguely inspired by the simplification of a biological neural network.

This model consists of a group of information interconnections made up of artificial and computational neurons processes. In most cases, an artificial neural network is an adaptive system that changes its own structure based on external or internal information flowing through the network itself during learning phase.

An artificial neural network receives external signals on a layer of input nodes (processing units), each of which is connected with numerous internal nodes, organized in several layers. Each node processes the received signals and transmits the result to subsequent nodes.

A neural Network model it can be both supervised and unsupervised. In this project we will use the ***supervised model***.

2.3 Data exploration

Let’s start by installing all the necessary libraries

```
# Install all the necessary libraries
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(corrplot)) install.packages("corrplot", repos = "http://cran.us.r-project.org")
if(!require(funModeling)) install.packages("funModeling", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(rstudioapi)) install.packages("rstudioapi", repos = "http://cran.us.r-project.org")
if(!require(adabag)) install.packages("adabag", repos = "http://cran.us.r-project.org")
if(!require(plyr)) install.packages("plyr", repos = "http://cran.us.r-project.org")
if(!require(MASS)) install.packages("MASS", repos = "http://cran.us.r-project.org")
if(!require(gbm)) install.packages("gbm", repos = "http://cran.us.r-project.org")
if(!require(pROC)) install.packages("pROC", repos = "http://cran.us.r-project.org")
if(!require(rpart.plot)) install.packages("rpart.plot")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
if(!require(nnet)) install.packages("nnet", repos = "http://cran.us.r-project.org")
if(!require(ggthemes)) install.packages("ggthemes", repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
if(!require(e1071)) install.packages("e1071")



library(tidyverse)
library(dplyr)
library(ggplot2)
library(corrplot)
library(funModeling)
library(caret)
library(rstudioapi)
library(adabag)
library(plyr)
library(MASS)
library(gbm)
library(pROC)
library(rpart.plot)
library(knitr)
library(nnet)
library(ggthemes)
library(randomForest)
library(e1071)
```

We load the data from the file

```
# Read data from file and save in a data frame
data <- read.csv(’https://raw.githubusercontent.com/LucaBarto/Heart-Diseases-classification/main/hea
```

Let’s explore the data structure

```
# Explore structure
str(data)
```

```
## ’data.frame':    303 obs. of  14 variables:
##  $ ï..age  : int  63 37 41 56 57 57 56 44 52 57 ...
##  $ sex     : int  1 1 0 1 0 1 0 1 1 1 ...
##  $ cp      : int  3 2 1 1 0 0 1 1 2 2 ...
##  $ trestbps: int  145 130 130 120 120 140 140 120 172 150 ...
##  $ chol    : int  233 250 204 236 354 192 294 263 199 168 ...
##  $ fbs     : int  1 0 0 0 0 0 0 0 1 0 ...
##  $ restecg : int  0 1 0 1 1 1 0 1 1 1 ...
##  $ thalach : int  150 187 172 178 163 148 153 173 162 174 ...
##  $ exang   : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ oldpeak : num  2.3 3.5 1.4 0.8 0.6 0.4 1.3 0 0.5 1.6 ...
##  $ slope   : int  0 0 2 2 2 1 1 2 2 2 ...
##  $ ca      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ thal    : int  1 2 2 2 2 1 2 3 3 2 ...
##  $ target  : int  1 1 1 1 1 1 1 1 1 1 ...
```

```
# Explore dimension
dim(data)
```

```
## [1] 303  14
```

```
# Summary of data
summary(data)
```

```
##      ï..age           sex               cp           trestbps
##  Min.   :29.00   Min.   :0.0000   Min.   :0.000   Min.   : 94.0
##  1st Qu.:47.50   1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:120.0
##  Median :55.00   Median :1.0000   Median :1.000   Median :130.0
##  Mean   :54.37   Mean   :0.6832   Mean   :0.967   Mean   :131.6
##  3rd Qu.:61.00   3rd Qu.:1.0000   3rd Qu.:2.000   3rd Qu.:140.0
##  Max.   :77.00   Max.   :1.0000   Max.   :3.000   Max.   :200.0
##       chol            fbs            restecg          thalach
##  Min.   :126.0   Min.   :0.0000   Min.   :0.0000   Min.   : 71.0
##  1st Qu.:211.0   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:133.5
##  Median :240.0   Median :0.0000   Median :1.0000   Median :153.0
##  Mean   :246.3   Mean   :0.1485   Mean   :0.5281   Mean   :149.6
##  3rd Qu.:274.5   3rd Qu.:0.0000   3rd Qu.:1.0000   3rd Qu.:166.0
##  Max.   :564.0   Max.   :1.0000   Max.   :2.0000   Max.   :202.0
##      exang           oldpeak         slope             ca
##  Min.   :0.0000   Min.   :0.00   Min.   :0.000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.00   1st Qu.:1.000   1st Qu.:0.0000
##  Median :0.0000   Median :0.80   Median :1.000   Median :0.0000
##  Mean   :0.3267   Mean   :1.04   Mean   :1.399   Mean   :0.7294
##  3rd Qu.:1.0000   3rd Qu.:1.60   3rd Qu.:2.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :6.20   Max.   :2.000   Max.   :4.0000
##       thal           target
##  Min.   :0.000   Min.   :0.0000
##  1st Qu.:2.000   1st Qu.:0.0000
##  Median :2.000   Median :1.0000
##  Mean   :2.314   Mean   :0.5446
##  3rd Qu.:3.000   3rd Qu.:1.0000
##  Max.   :3.000   Max.   :1.0000
```

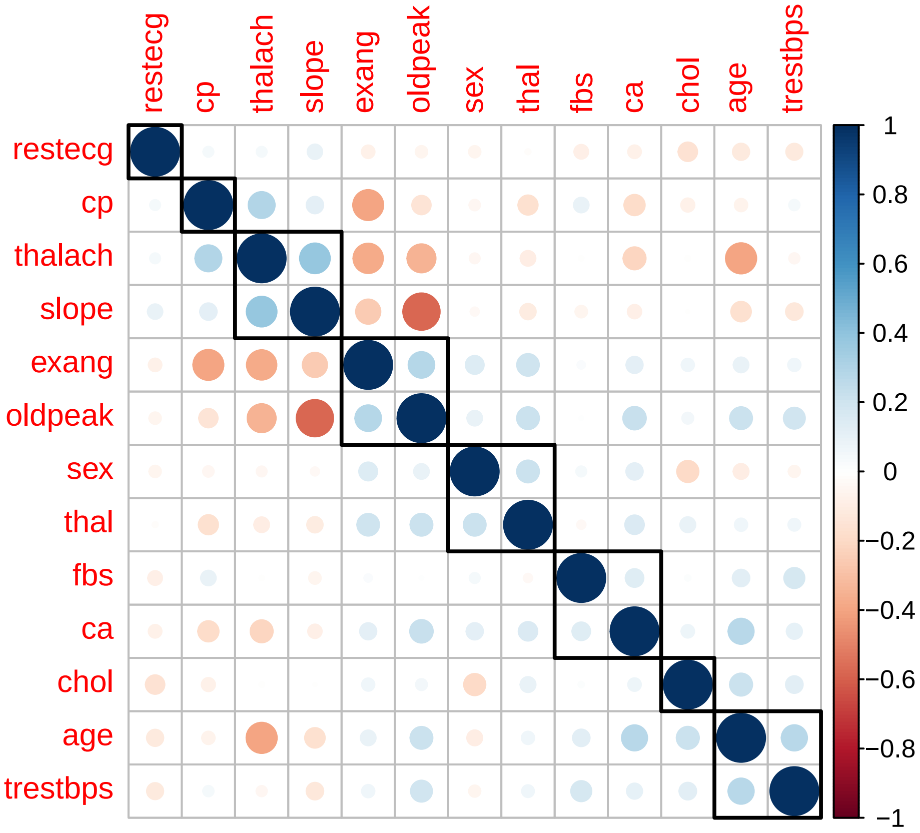
Let's try to understand if they are present missing value and then we are going to analyze the correlation between predictors.

For correlation we use corrplot package that provides a graphical display of a correlation matrix, confidence interval or general matrix.

```r
# Find correlation between predictors

# Change column name
colnames(data)[1] <- "age"

# Set all numeric outputs to 3 digits
options(digits = 3)

# Check for missing values
map_int(data, function(.x) sum(is.na(.x)))
```

```
##      age      sex       cp trestbps     chol      fbs  restecg  thalach
##        0        0        0        0        0        0        0        0
##    exang  oldpeak    slope       ca     thal   target
##        0        0        0        0        0        0
```

```r
# Correlation matrix
correlationMatrix <- cor(data[,1:ncol(data) - 1])

# The corrplot package is a graphical display of a correlation matrix,
# confidence interval or general matrix
corrplot(correlationMatrix, order = "hclust", tl.cex = 1, addrect = 8)
```



```r
# Find attributes that are highly corrected
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.6)

# Print indexes of highly correlated attributes
highlyCorrelated
```

```
## integer(0)
```

There are no missing values.

There is some sort of negative correlation between:

1. cp whit exchang
2. thalach with exchang, oldpeack and age
3. splope with oldpeak

Anyway correllation is below 0.60, therefore well below the cut-off of 0.70 below which it can be said that it does not exist hight ***Collinearity*** between predictors, but it is not enough to be confident in using ***Naive Bayes*** algorithm.

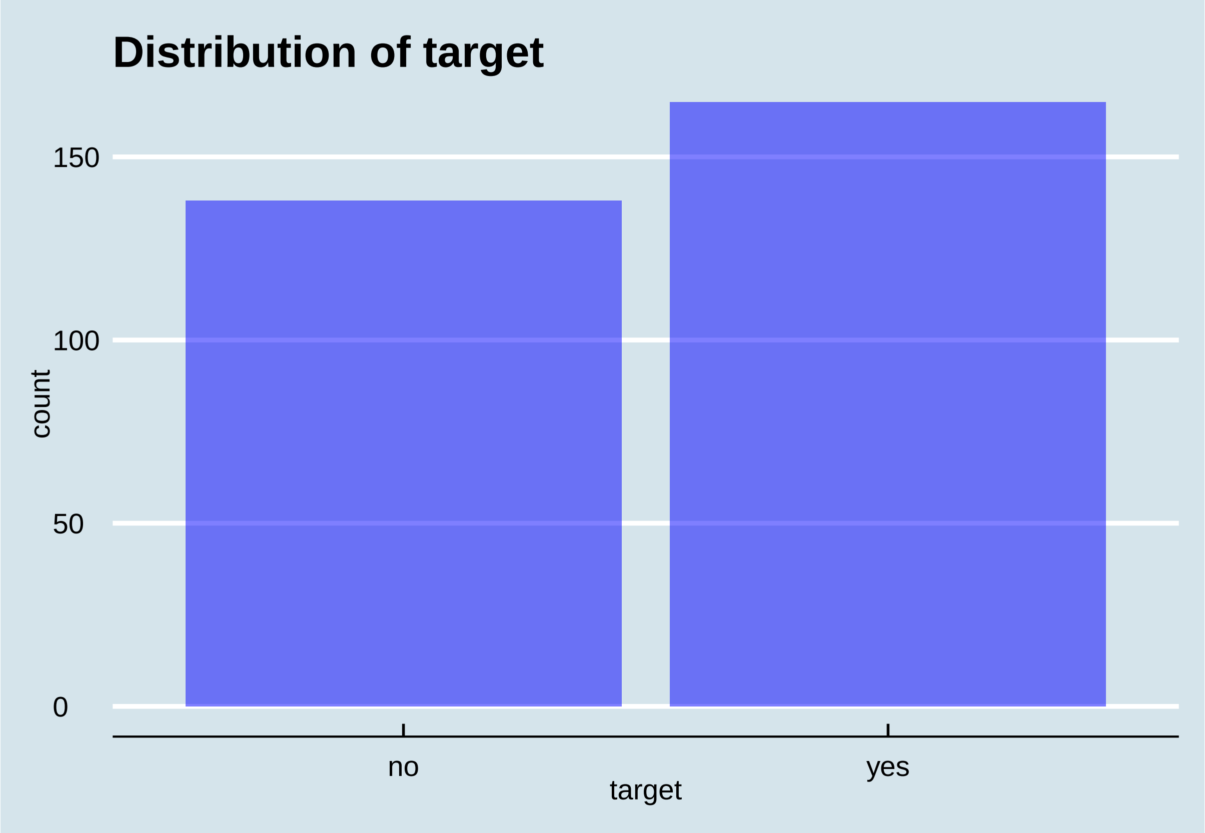
```
# Change target variable to factor
data$target <- as.factor(ifelse(data$target == 1, "yes", "no"))
```

2.4 Visualization

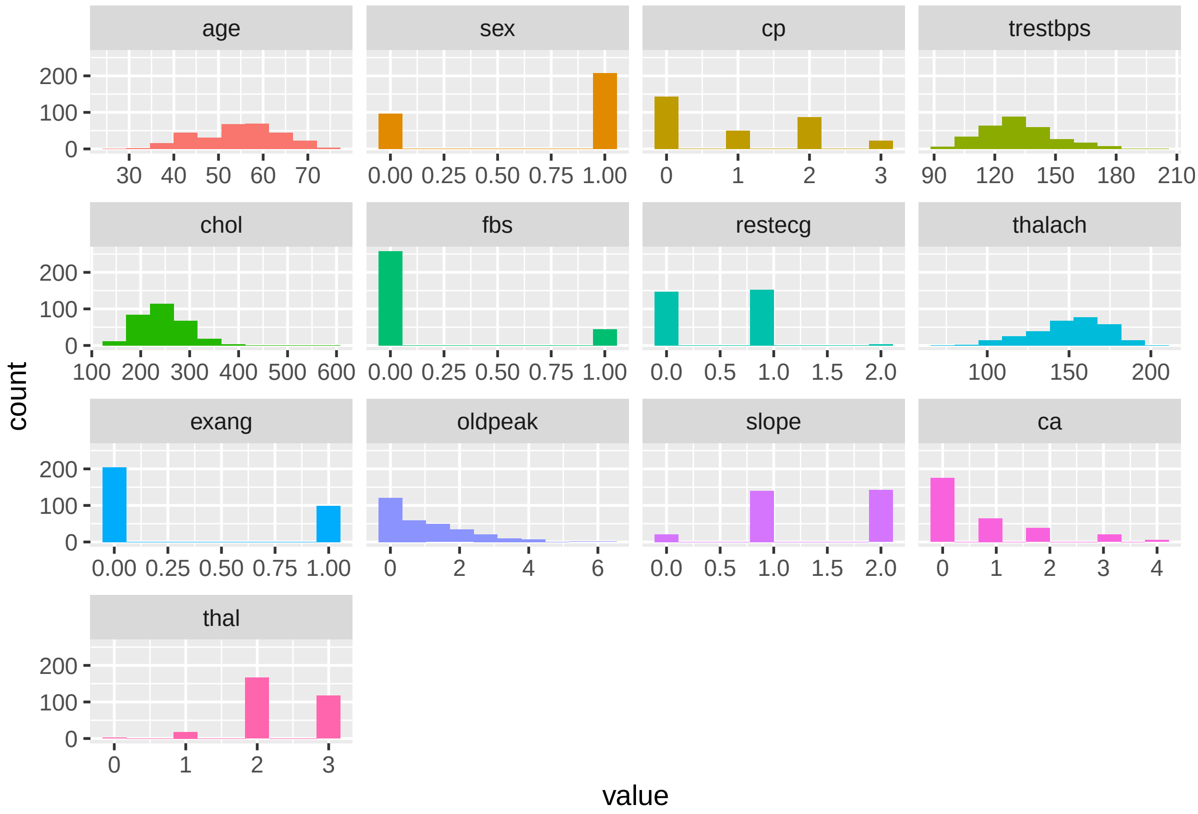
```
# Check proportion of data
prop.table(table(data$target))
```

```
##
##    no   yes
## 0.455 0.545
```

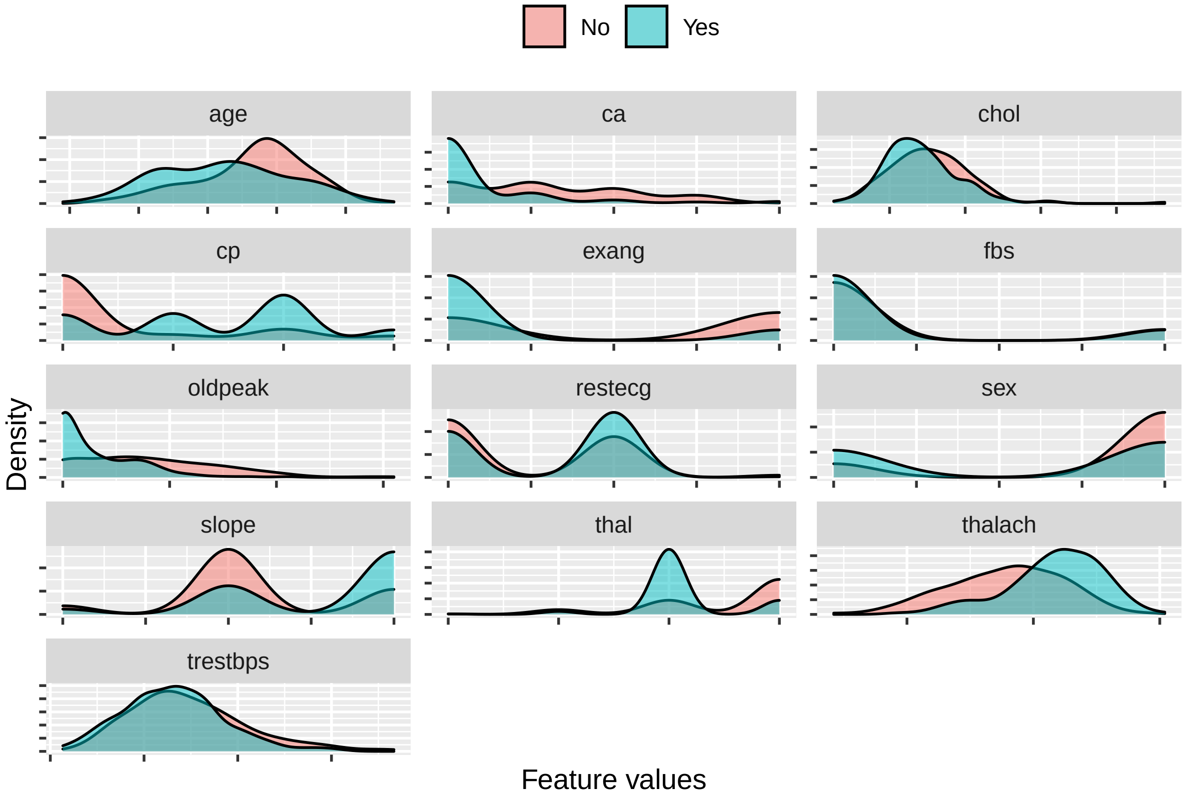
```
# Plot distribution of target
ggplot(data, aes(x=target)) +
  geom_bar(fill="blue",alpha=0.5) +
  theme_economist() +
  labs(title="Distribution of target")
```



```
# Plotting Numerical Data
plot_num(data, bins=10)
```



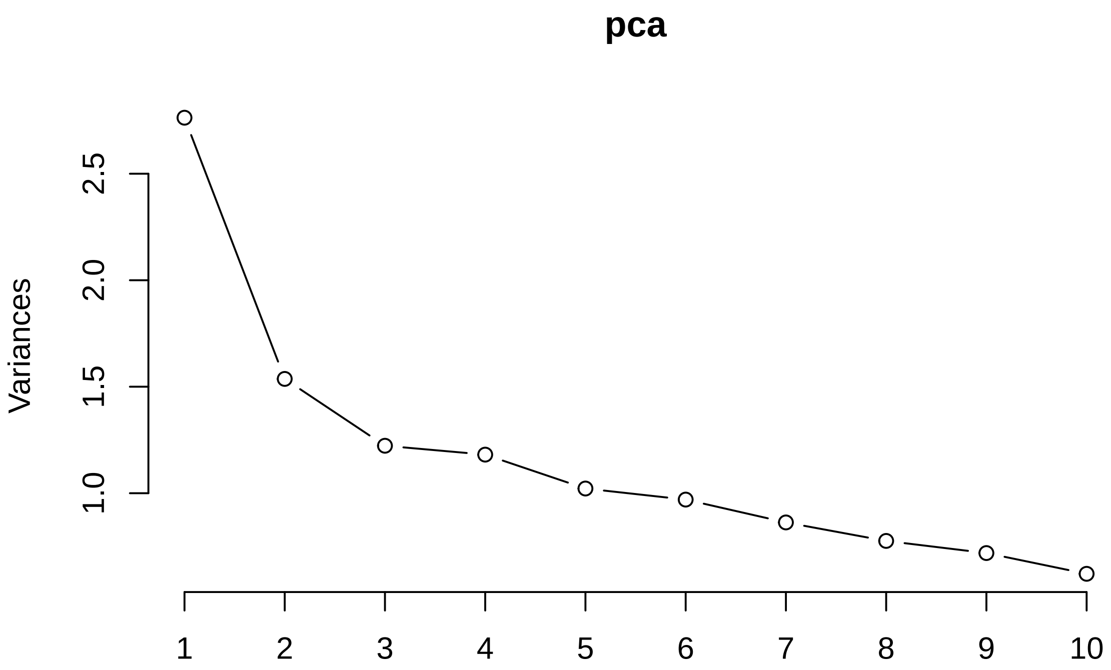
```
# Target correlation with predictors
# Plot and facet wrap density plots
data %>%
  gather("feature", "value", -target) %>%
  ggplot(aes(value, fill = target)) +
  geom_density(alpha = 0.5) +
  xlab("Feature values") +
  ylab("Density") +
  theme(legend.position = "top",
        axis.text.x = element_blank(), axis.text.y = element_blank(),
        legend.title=element_blank()) +
  scale_fill_discrete(labels = c("No", "Yes")) +
  facet_wrap(~ feature, scales = "free", ncol = 3)
```



Only few variables are normally distributed

Now we apply the Principal Component Analysis to understand if it is possible to reduce the number of predictors

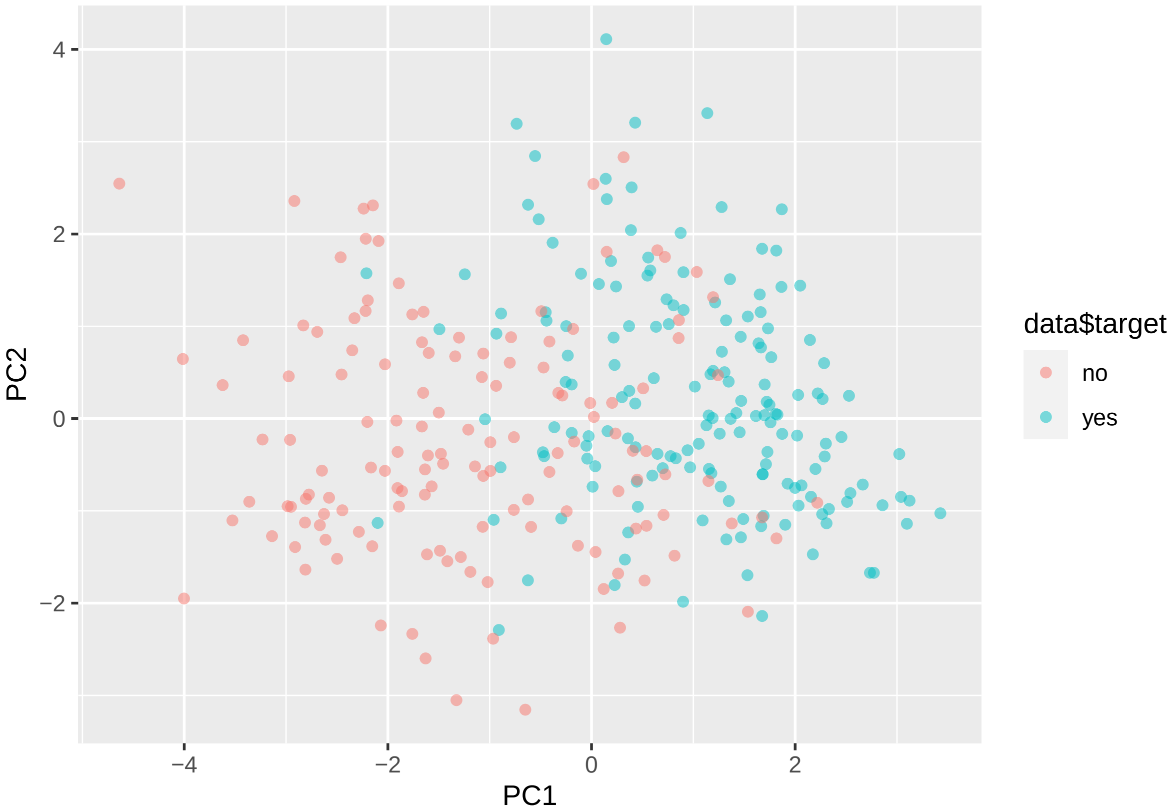
```
# Principal Component Analysis (PCA)
pca <- prcomp(data[,1:ncol(data) - 1], center = TRUE, scale = TRUE)
plot(pca, type="l")
```



```
# Summary of data after PCA
summary(pca)
```

```
## Importance of components:
##                          PC1   PC2    PC3    PC4    PC5    PC6    PC7    PC8
## Standard deviation     1.662 1.240 1.1058 1.0868 1.0109 0.9849 0.9289 0.8809
## Proportion of Variance 0.213 0.118 0.0941 0.0909 0.0786 0.0746 0.0664 0.0597
## Cumulative Proportion  0.213 0.331 0.4248 0.5157 0.5943 0.6689 0.7353 0.7950
##                           PC9   PC10   PC11   PC12   PC13
## Standard deviation     0.8479 0.7884 0.7281 0.6505 0.6098
## Proportion of Variance 0.0553 0.0478 0.0408 0.0326 0.0286
## Cumulative Proportion  0.8503 0.8981 0.9388 0.9714 1.0000
```

```
# We need 12 variables to reach 95% of the variance

pca_df <- as.data.frame(pca$x)
ggplot(pca_df, aes(x=PC1, y=PC2, col=data$target)) + geom_point(alpha=0.5)
```



```
# The data of the first 2 components cannot be easly separated into two classes.
```

The data of the first 2 components cannot be easly separated into two classes.

We need 12 variables to reach 95% of the variance, so there is no point to implement PCA to reduce the number of predictors.

We are going to use all predictors on dataset.

3 Results

We begin the implementation of the various algorithms described above.

For the implementation of the algorithms we will use the Caret package

We start creating the partition 80% and 20% to create respectively the training dataset and test dataset.

```r
# Creation of the partition 80% and 20%
set.seed(1, sample.kind="Rounding")
target_index <- createDataPartition(data$target, times=1, p=0.8, list = FALSE)
train_data <- data[target_index, ]
test_data <- data[-target_index, ]
```

The function trainControl generates parameters that further control how models are created

```r
# Define train control
fitControl <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 10,
                           ## Estimate class probabilities
                           classProbs = TRUE,
                           ## Evaluate performance using
                           ## the following function
                           summaryFunction = twoClassSummary)
```

For each algorithm, the following operations will be performed:

1. set up a grid of adjustment parameters which, by adapting to the model and calculating its performance, will allow us to determine the values that provide optimal performance;
2. Train model;
3. Exploration and visualization of the model, with indication of the best values for tuning parameters;
4. Perform the prediction;
5. Evaluation of confusion matrix will show the best values for parameters and the relative performance obtained;
6. Show the 10 most important predictors;
7. Show the ROC cure with highlighted the best value for Specificity and Sensitivity.

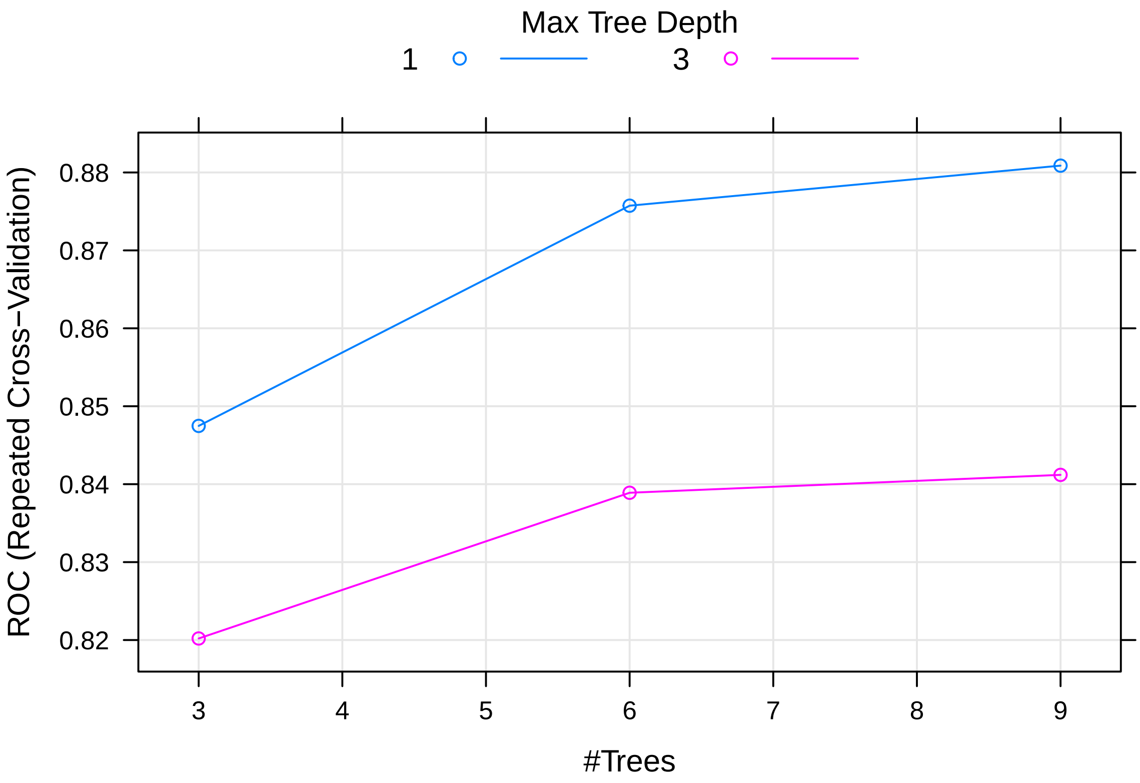
3.1 Adaptive Boosting

```
# Set up tuning grid
am1_grid <- expand.grid(mfinal = (1:3)*3,
                         maxdepth = c(1, 3),
                         coeflearn = c("Zhu"))
```

Tuning parameters:

1. mfinal (#Trees)
2. maxdepth (Max Tree Depth)
3. coeflearn (Coefficient Type)

```
# Train model
set.seed(1, sample.kind="Rounding")
am1_model <- train(target~., data=train_data,
                   method = "AdaBoost.M1",
                   trControl = fitControl,
                   verbose = FALSE,
                   tuneGrid = am1_grid,
                   # center, scale - centering and scaling data
                   preProcess = c("center", "scale"),
                   metric = "ROC")

plot(am1_model)
```

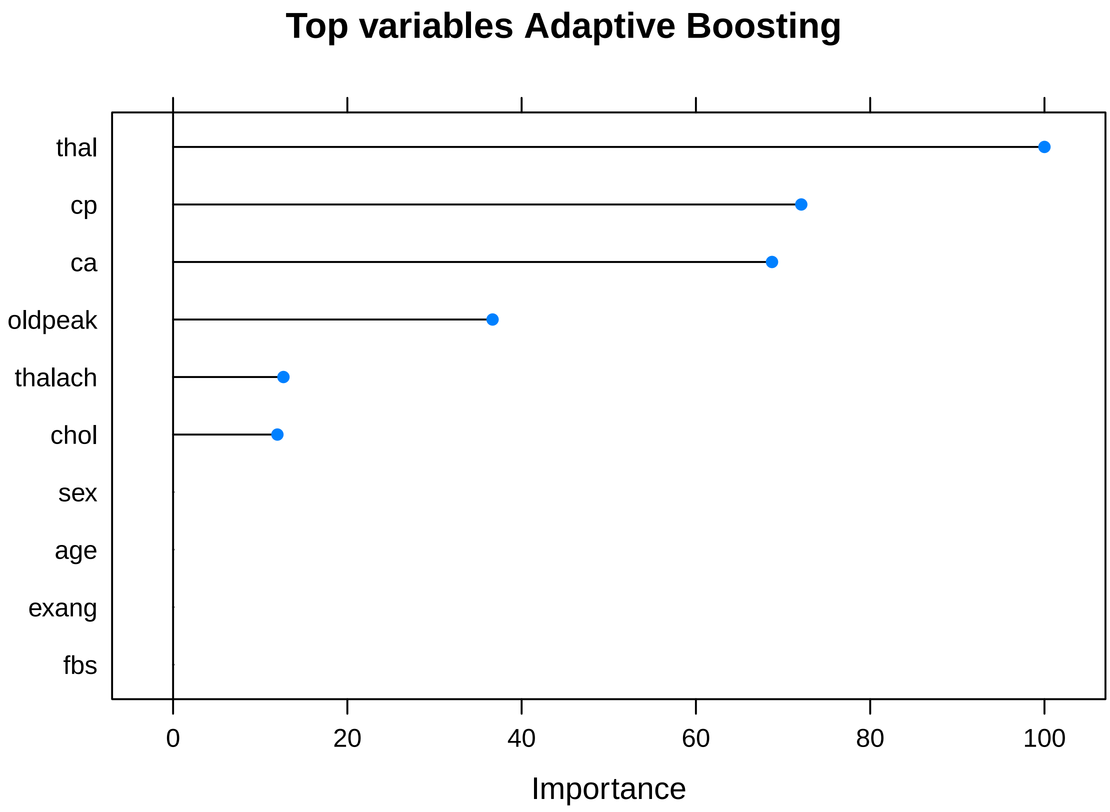


```
# Predict data
set.seed(1, sample.kind="Rounding")
am1_pred <- predict(am1_model, newdata = test_data)
```

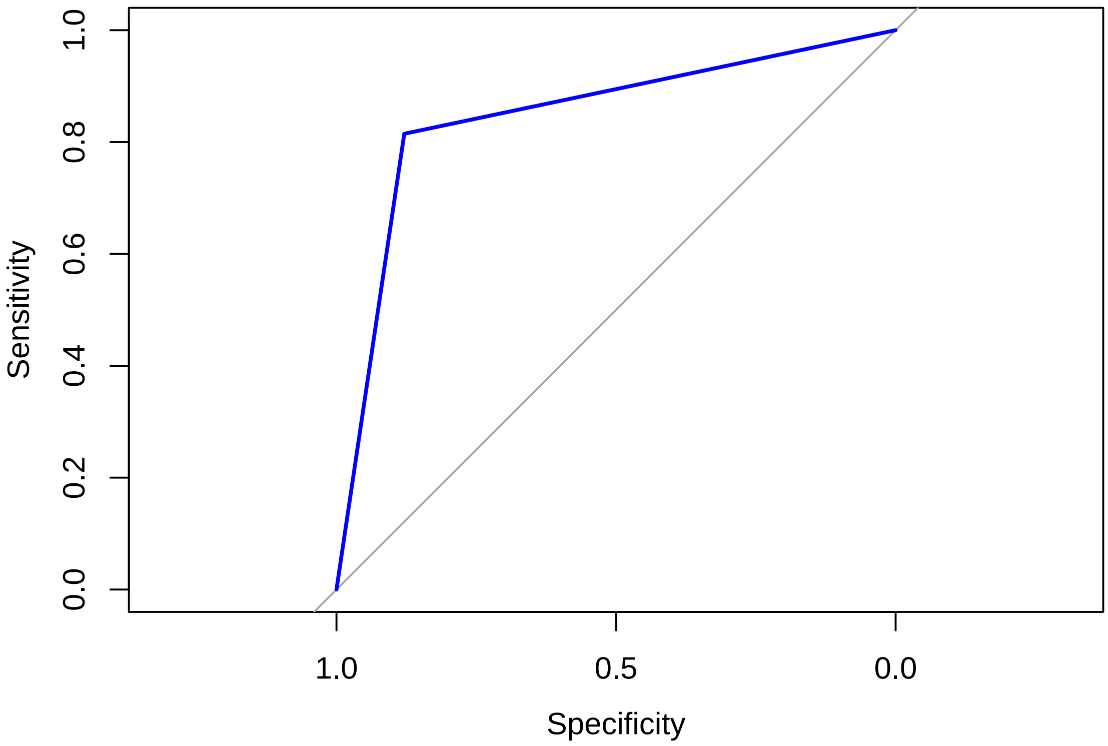
```
# Evaluate confusion matrix
am1_confusionMatrix <- confusionMatrix(am1_pred, test_data$target)

am1_confusionMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no  22   4
##        yes  5  29
##
##                Accuracy : 0.85
##                  95% CI : (0.734, 0.929)
##     No Information Rate : 0.55
##     P-Value [Acc > NIR] : 8.07e-07
##
##                   Kappa : 0.696
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.815
##             Specificity : 0.879
##          Pos Pred Value : 0.846
##          Neg Pred Value : 0.853
##              Prevalence : 0.450
##          Detection Rate : 0.367
##    Detection Prevalence : 0.433
##       Balanced Accuracy : 0.847
##
##        'Positive' Class : no
##
```

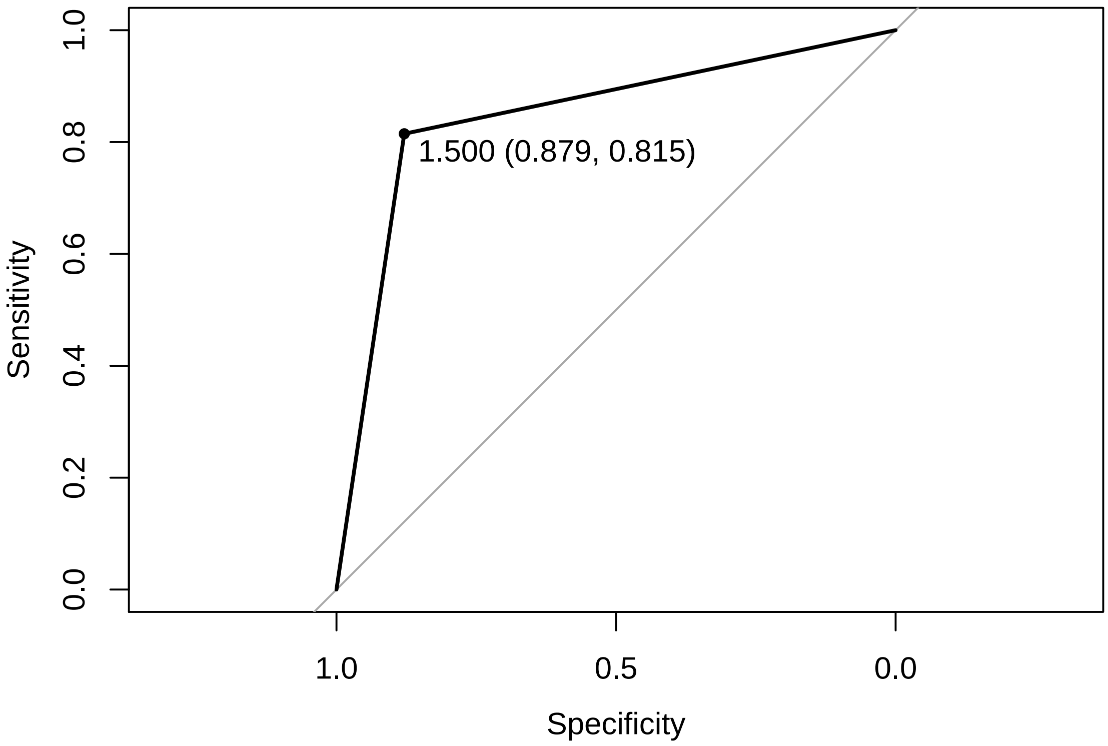
```
# Plot 10 most important variables
plot(varImp(am1_model), top=10, main="Top variables Adaptive Boosting")
```



```r
# Define ROC Curve
am1_rocCurve   <- roc(response = test_data$target,
                      predictor = as.numeric(am1_pred),
                      levels = rev(levels(test_data$target)),
                      plot = TRUE, col = "blue", auc = TRUE)
```



```
# Plot ROC curve
plot(am1_rocCurve, print.thres = "best")
```



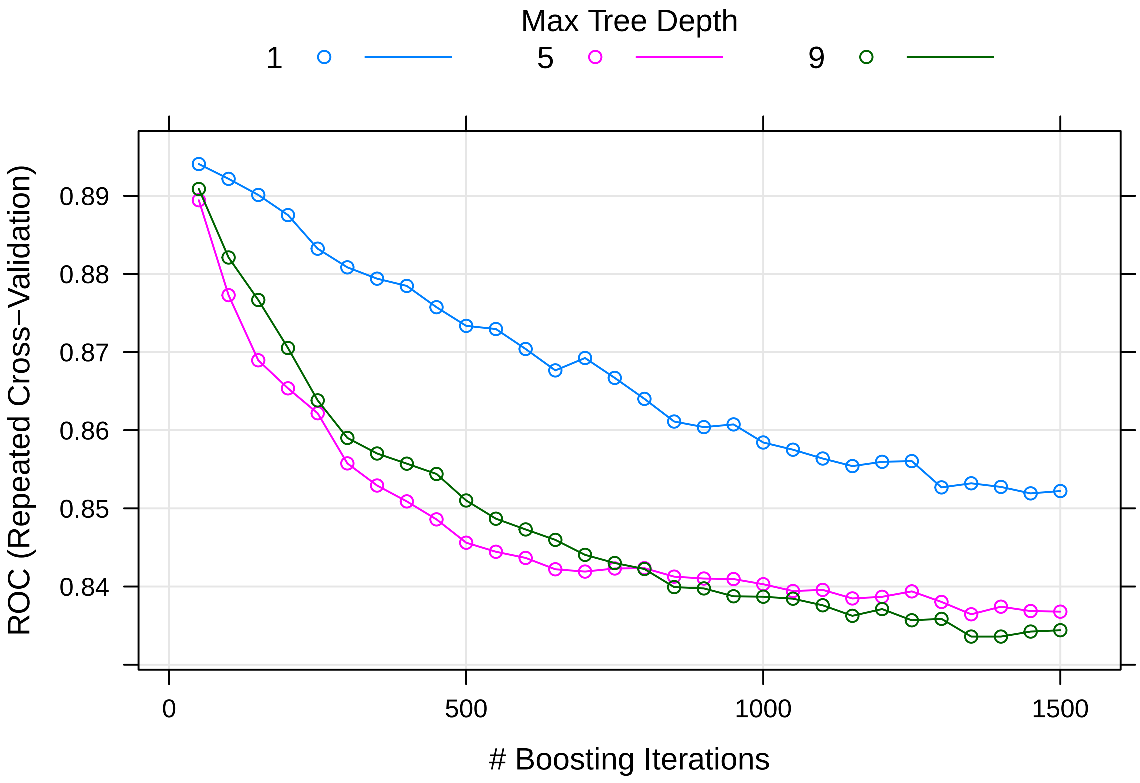
3.2 Stochastic Gradient Boosting

```
# Set up tuning grid
gbm_grid <-  expand.grid(interaction.depth = c(1, 5, 9),
                         n.trees = (1:30)*50,
                         shrinkage = 0.1,
                         n.minobsinnode = 20)
```

Tuning parameters:

1. interaction.depth (Max Tree Depth)
2. n.trees (Boosting Iterations)
3. shrinkage (Bandwidth Adjustment)
4. n.minobsinnode (Min. Terminal Node Size)

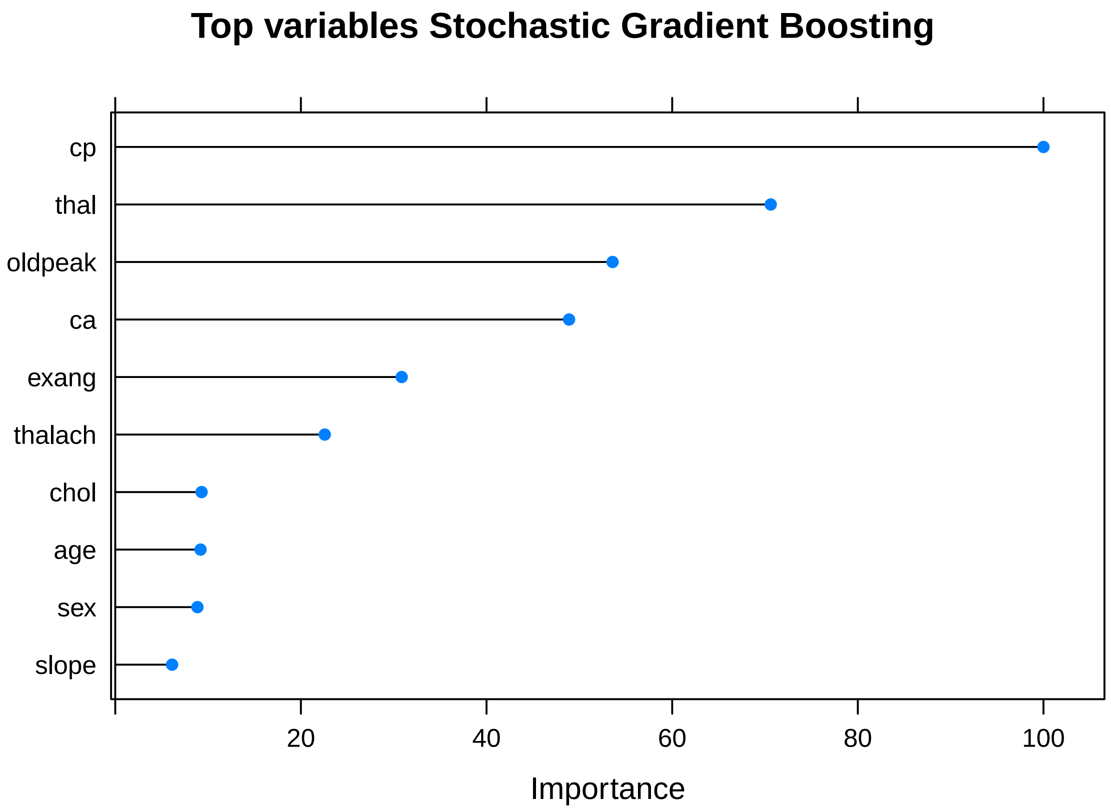
```
# Train model
set.seed(1, sample.kind="Rounding")
gbm_model <- train(target~., data=train_data,
                   method = "gbm",
                   trControl = fitControl,
                   verbose = FALSE,
                   tuneGrid = gbm_grid,
                   # center, scale - centering and scaling data
                   preProcess = c("center", "scale"),
                   ## Specify which metric to optimize
                   metric = "ROC")

plot(gbm_model)
```



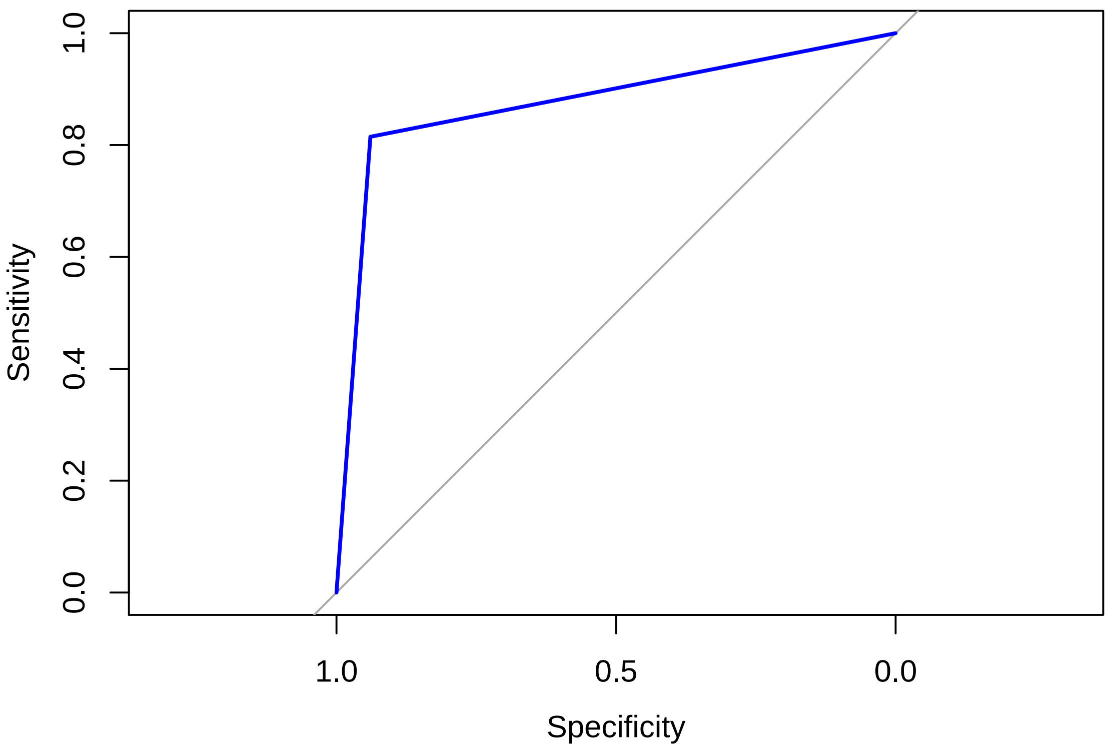
```r
# Predict data
set.seed(1, sample.kind="Rounding")
gbm_pred <- predict(gbm_model, newdata = test_data)

# Evaluate confusion matrix
gbm_confusionMatrix <- confusionMatrix(gbm_pred, test_data$target)

gbm_confusionMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no  22   2
##        yes  5  31
##
##                Accuracy : 0.883
##                  95% CI : (0.774, 0.952)
##     No Information Rate : 0.55
##     P-Value [Acc > NIR] : 2.96e-08
##
##                   Kappa : 0.762
##
##  Mcnemar's Test P-Value : 0.45
##
##             Sensitivity : 0.815
##             Specificity : 0.939
##          Pos Pred Value : 0.917
##          Neg Pred Value : 0.861
##              Prevalence : 0.450
##          Detection Rate : 0.367
##    Detection Prevalence : 0.400
##       Balanced Accuracy : 0.877
##
##        'Positive' Class : no
##
```

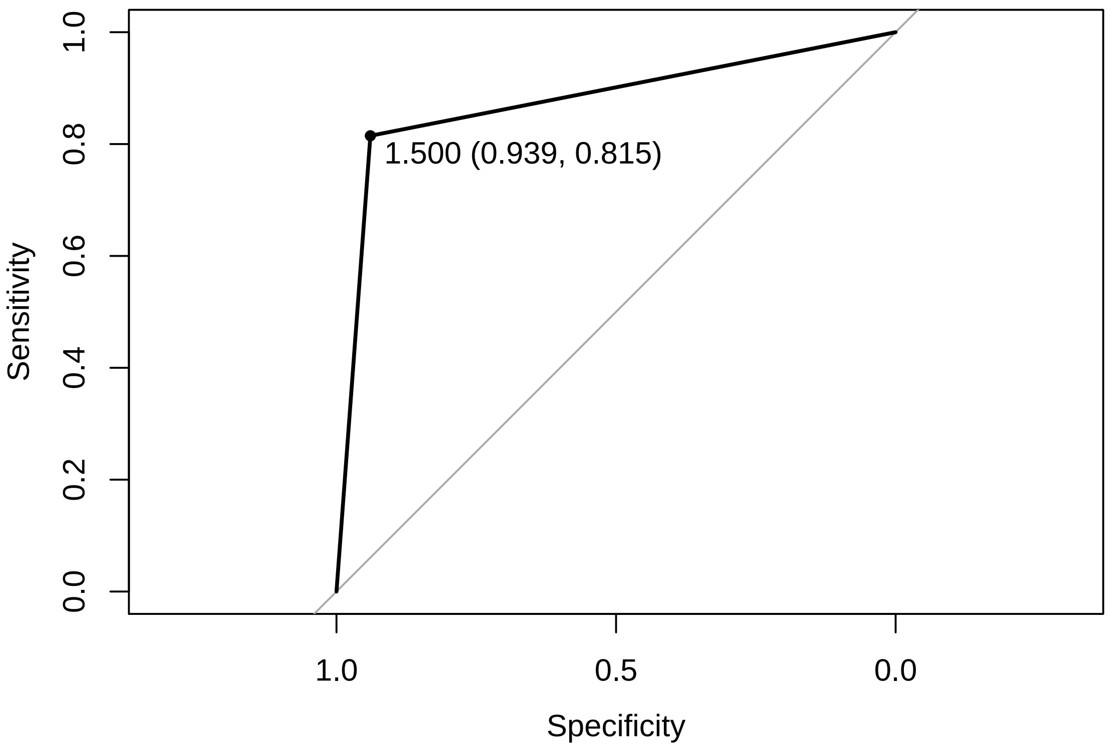
```r
# Plot 10 most important variables
plot(varImp(gbm_model), top=10, main="Top variables Stochastic Gradient Boosting")
```



```r
# Define ROC Curve
gbm_rocCurve   <- roc(response = test_data$target,
                     predictor = as.numeric(gbm_pred),
                     levels = rev(levels(test_data$target)),
                     plot = TRUE, col = "blue", auc = TRUE)
```



```r
# Plot ROC curve
plot(gbm_rocCurve, print.thres = "best")
```



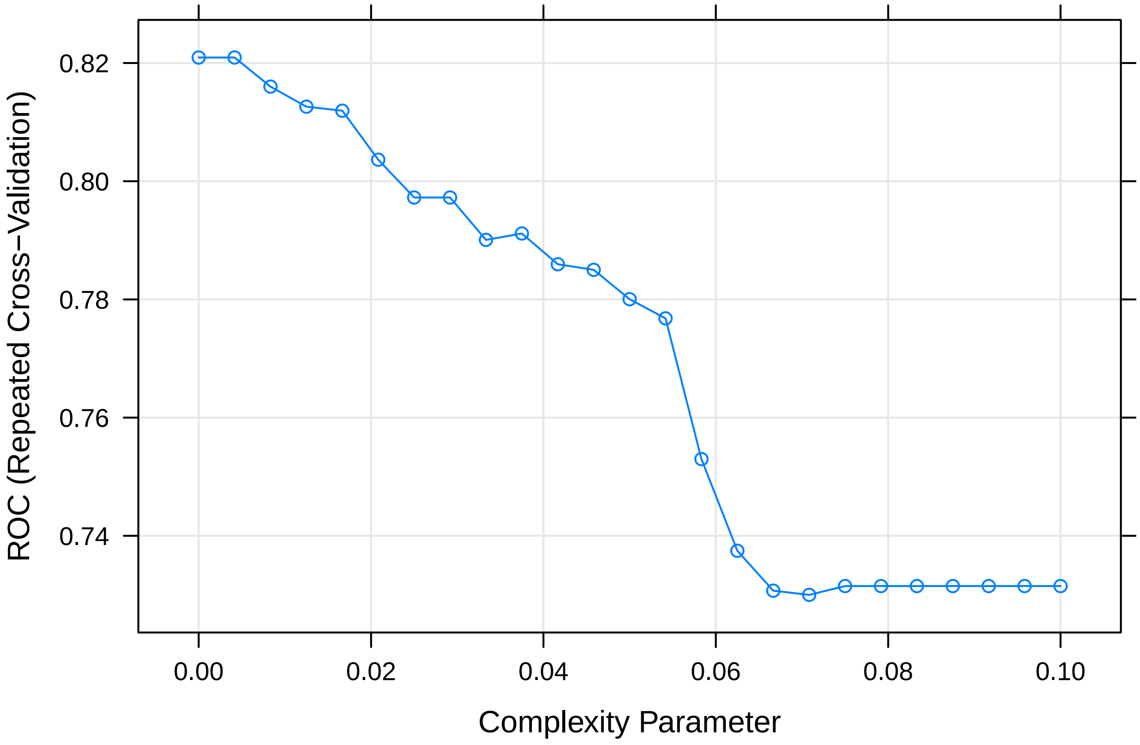
3.3 Classification Trees

```
# Set up tuning grid
ct_grid <-  data.frame(cp = seq(0.0, 0.1, len = 25))
```

Tuning parameters:

1. cp (Complexity Parameter)

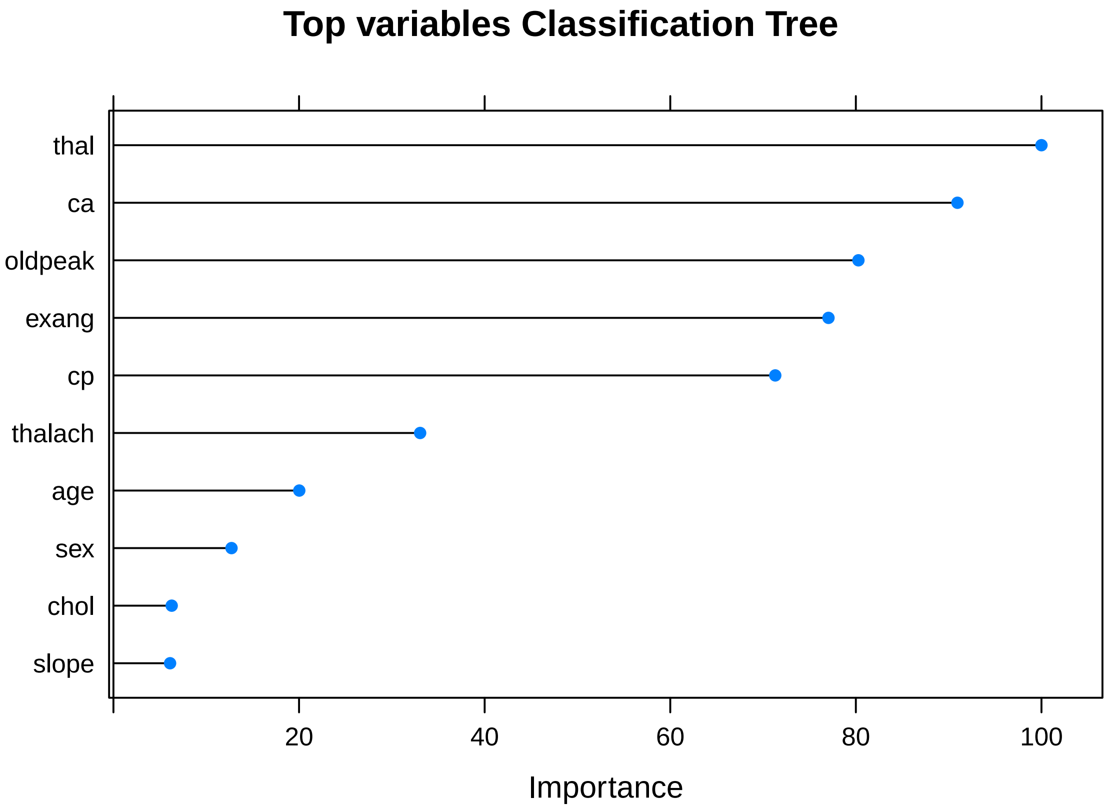
```
# Train model
set.seed(1, sample.kind="Rounding")
ct_model <- train(target~., data=train_data,
                  method = "rpart",
                  metric="ROC",
                  # center, scale - centering and scaling data
                  preProcess = c("center", "scale"),
                  tuneGrid = ct_grid,
                  trControl = fitControl)

plot(ct_model)
```



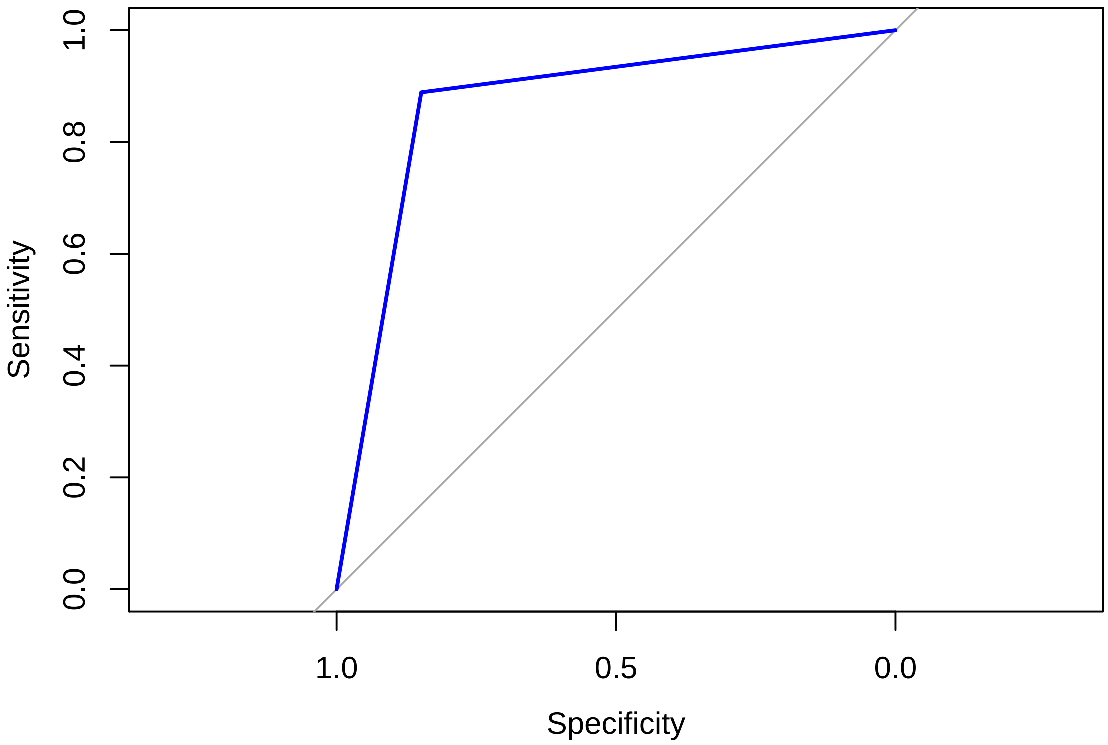
```
# Predict data
set.seed(1, sample.kind="Rounding")
ct_pred <- predict(ct_model, newdata = test_data)

# Evaluate confusion matrix
ct_confusionMatrix <- confusionMatrix(ct_pred, test_data$target)

ct_confusionMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no  24   5
##        yes  3  28
##
##                Accuracy : 0.867
##                  95% CI : (0.754, 0.941)
##     No Information Rate : 0.55
##     P-Value [Acc > NIR] : 1.65e-07
##
##                   Kappa : 0.732
##
##  Mcnemar's Test P-Value : 0.724
##
##             Sensitivity : 0.889
##             Specificity : 0.848
##          Pos Pred Value : 0.828
##          Neg Pred Value : 0.903
##              Prevalence : 0.450
##          Detection Rate : 0.400
##    Detection Prevalence : 0.483
##       Balanced Accuracy : 0.869
##
##        'Positive' Class : no
##
```

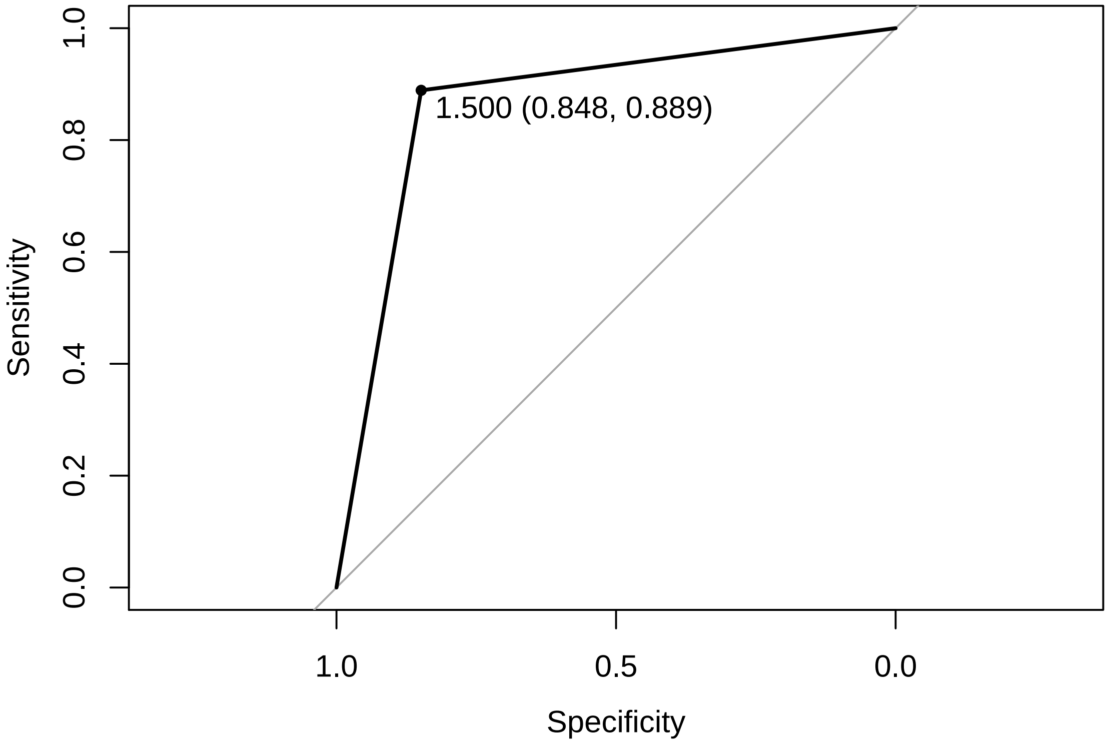
```
# Plot 10 most important variables
plot(varImp(ct_model), top=10, main="Top variables Classification Tree")
```



```
# Define ROC Curve
ct_rocCurve   <- roc(response = test_data$target,
                     predictor = as.numeric(ct_pred),
                     levels = rev(levels(test_data$target)),
                     plot = TRUE, col = "blue", auc = TRUE)
```

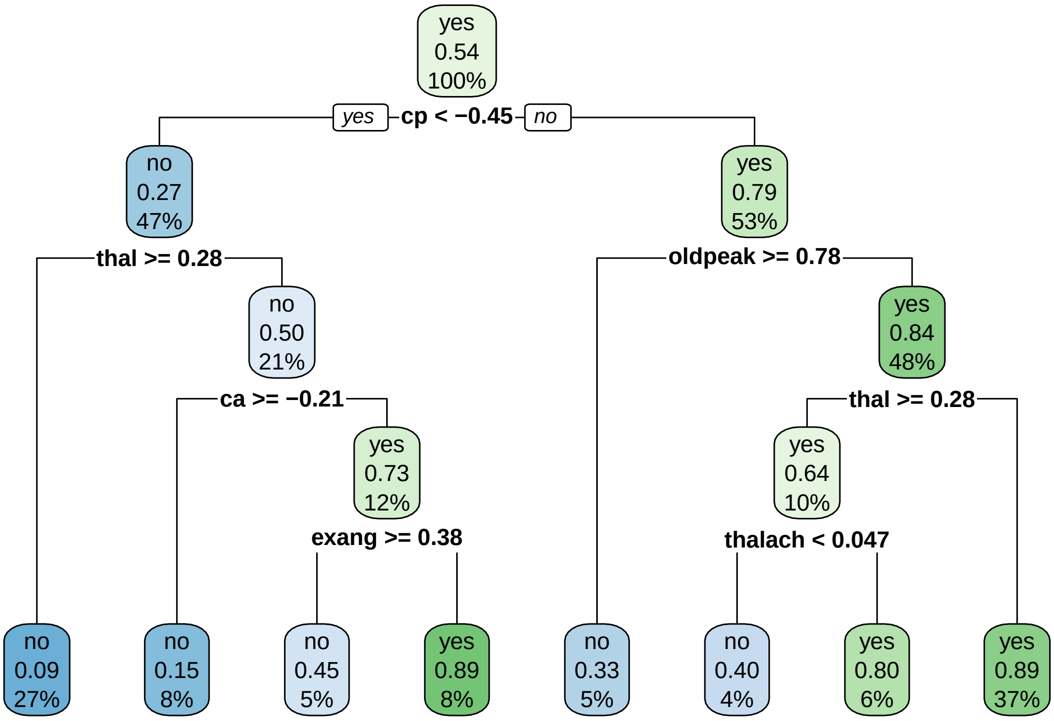


```
# Plot ROC curve
plot(ct_rocCurve, print.thres = "best")
```



The graph below shows the decision tree

```
rpart.plot(ct_model$finalModel)
```



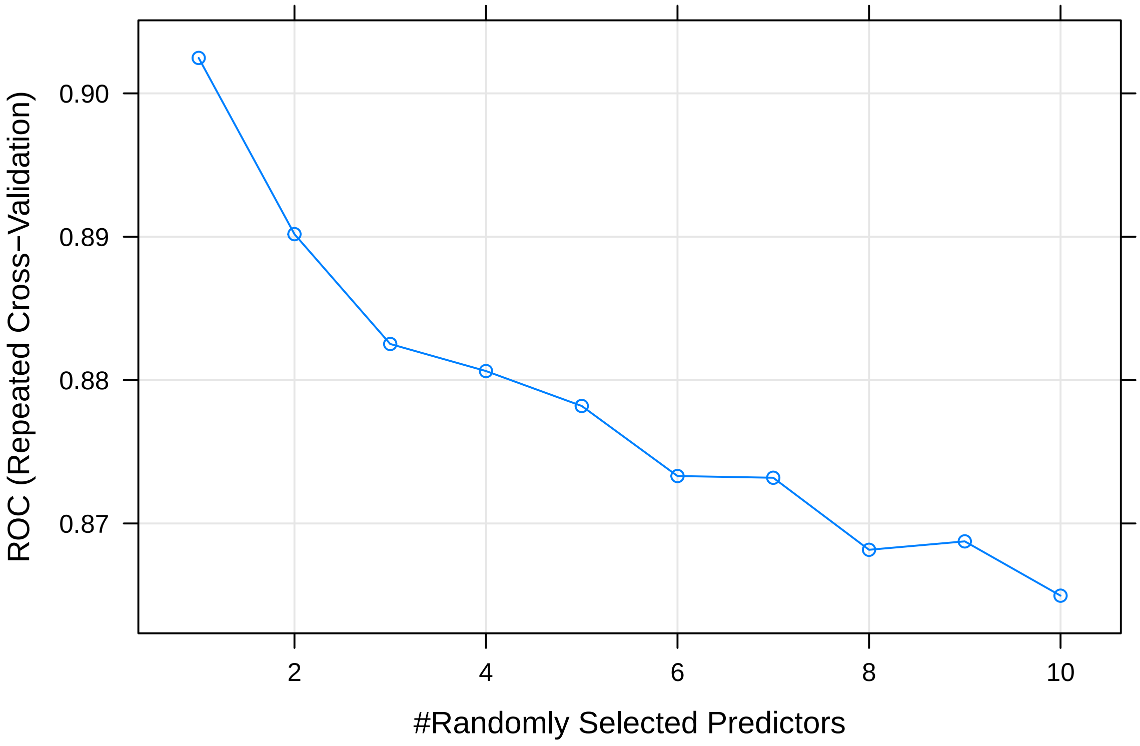
3.4 Random Forest

```
# Set up tuning grid
rf_grid <-  data.frame(mtry = seq(1, 10))
```

Tuning parameters:

1. mtry (#Randomly Selected Predictors)

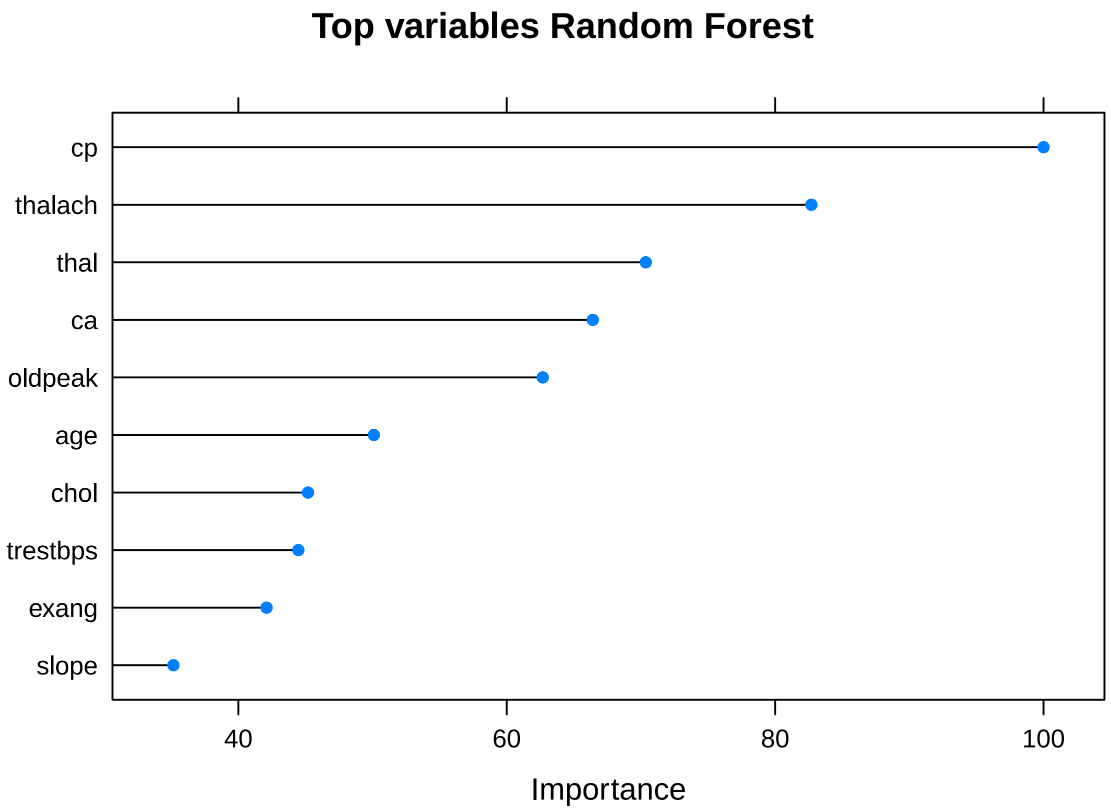
```
# Train model
set.seed(1, sample.kind="Rounding")
rf_model <- train(target~., data=train_data,
                  method = "rf",
                  metric = "ROC",
                  # center, scale - centering and scaling data
                  preProcess = c("center", "scale"),
                  tuneGrid = rf_grid,
                  ntree = 100,
                  trControl = fitControl)

plot(rf_model)
```



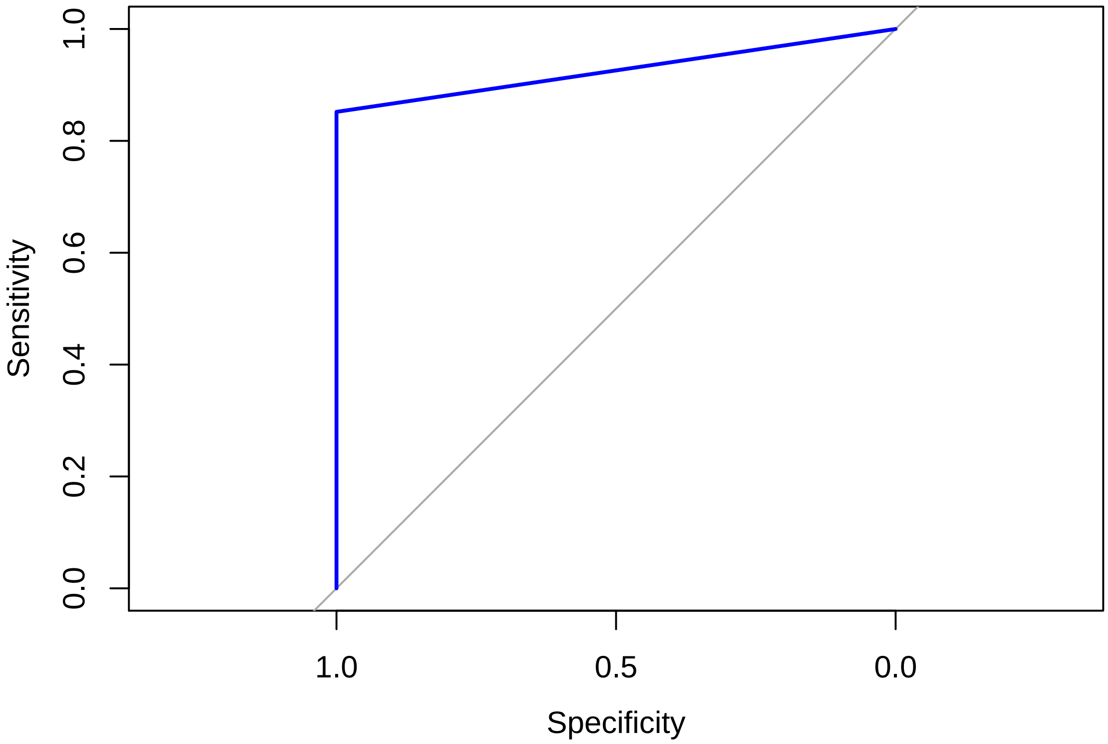
```
# Predict data
set.seed(1, sample.kind="Rounding")
rf_pred <- predict(rf_model, newdata = test_data)

# Evaluate confusion matrix
rf_confusionMatrix <- confusionMatrix(rf_pred, test_data$target)

rf_confusionMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no  23   0
##        yes  4  33
##
##                Accuracy : 0.933
##                  95% CI : (0.838, 0.982)
##     No Information Rate : 0.55
##     P-Value [Acc > NIR] : 6.3e-11
##
##                   Kappa : 0.863
##
##  Mcnemar's Test P-Value : 0.134
##
##             Sensitivity : 0.852
##             Specificity : 1.000
##          Pos Pred Value : 1.000
##          Neg Pred Value : 0.892
##              Prevalence : 0.450
##          Detection Rate : 0.383
##    Detection Prevalence : 0.383
##       Balanced Accuracy : 0.926
##
##        'Positive' Class : no
##
```

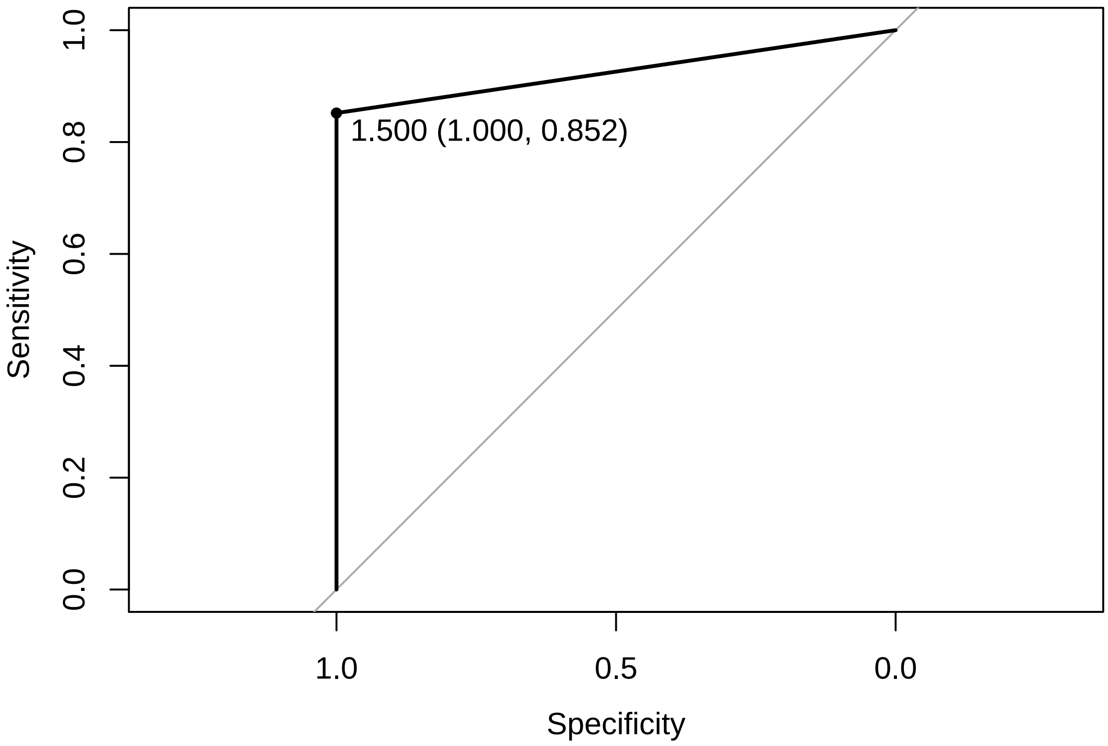
```
# Plot 10 most important variables
plot(varImp(rf_model), top=10, main="Top variables Random Forest")
```



```
# Define ROC Curve
rf_rocCurve   <- roc(response = test_data$target,
                     predictor = as.numeric(rf_pred),
                     levels = rev(levels(test_data$target)),
                     plot = TRUE, col = "blue", auc = TRUE)
```



```
# Plot ROC curve
plot(rf_rocCurve, print.thres = "best")
```


1.500 (1.000, 0.852)
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
Specificity
1.0
0.5
0.0

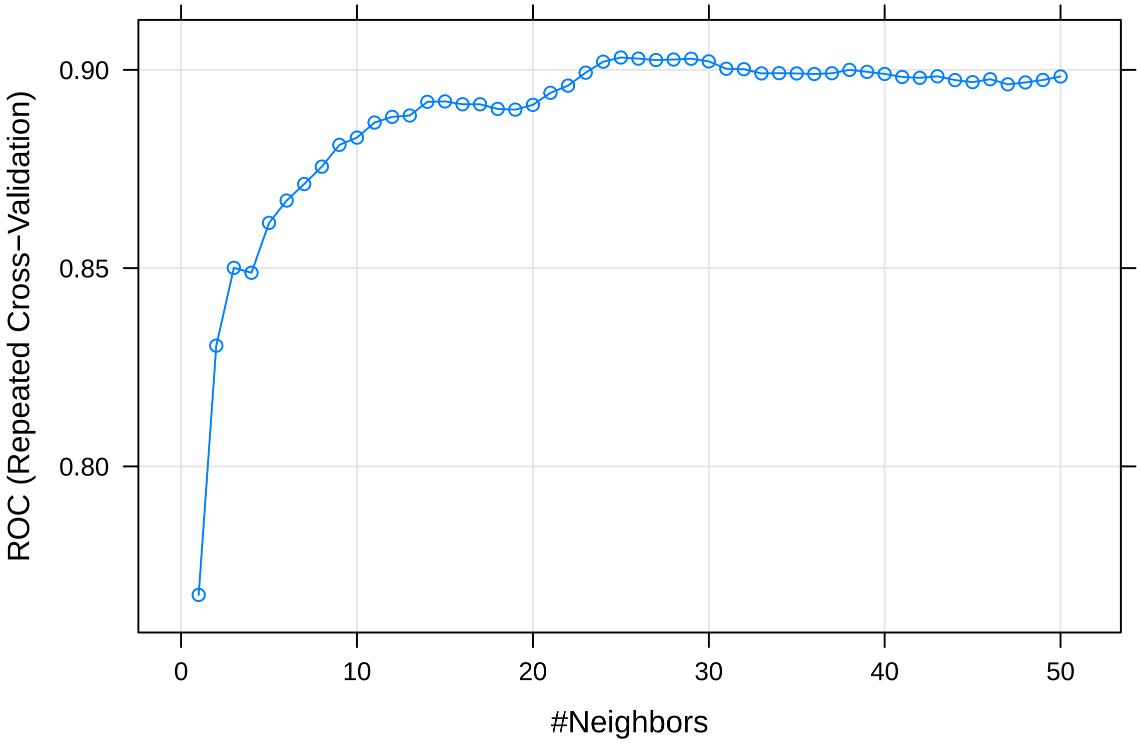
3.5 K Nearest Neighbor (KNN)

```
# Set up tuning grid
knn_grid <-  data.frame(k = seq(1, 50, 1))
```

Tuning parameters:

1. k (#Neighbors)

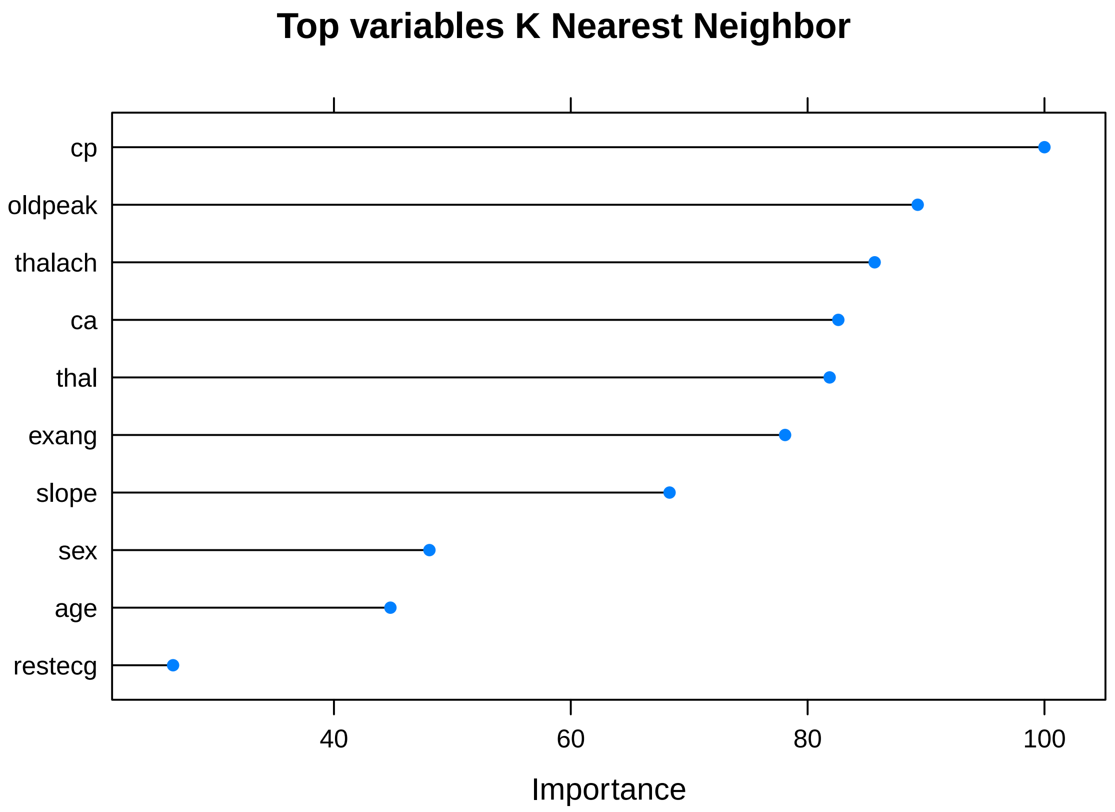
```
# Train model
set.seed(1, sample.kind="Rounding")
knn_model <- train(target~., data=train_data,
                   method="knn",
                   metric="ROC",
                   # center, scale - centering and scaling data
                   preProcess = c("center", "scale"),
                   tuneGrid = knn_grid,
                   trControl=fitControl)

plot(knn_model)
```



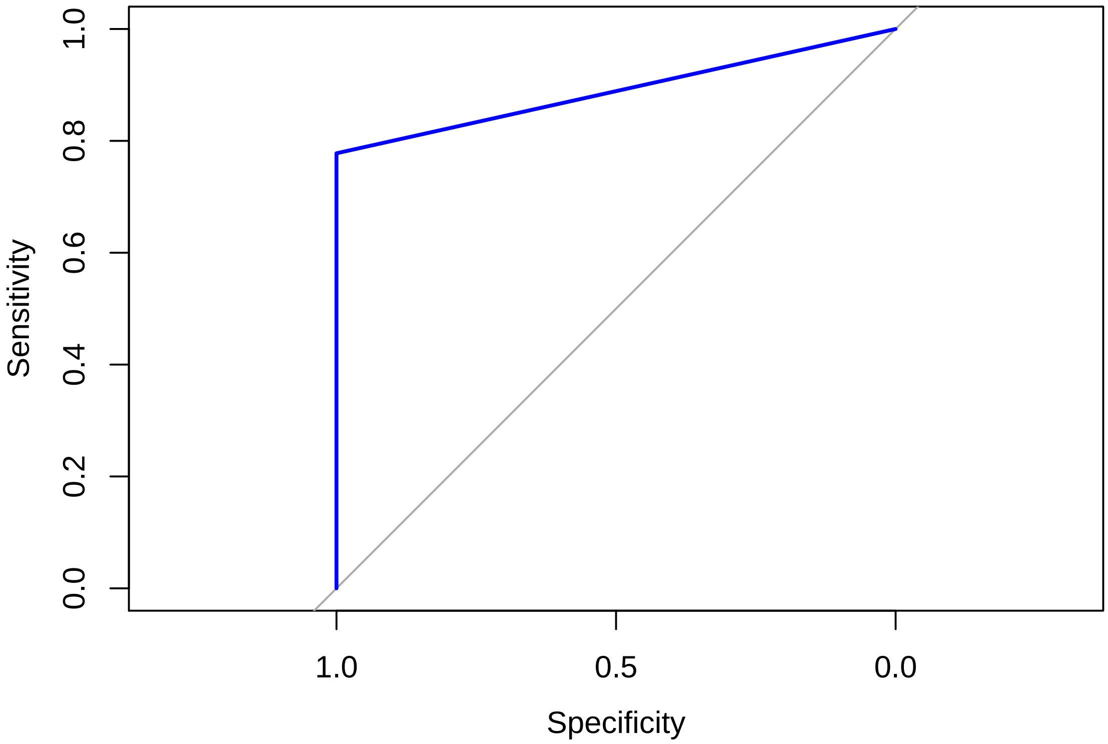
```
# Predict data
set.seed(1, sample.kind="Rounding")
knn_pred <- predict(knn_model, newdata = test_data)

# Evaluate confusion matrix
knn_confusionMatrix <- confusionMatrix(knn_pred, test_data$target)

knn_confusionMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no  21   0
##        yes  6  33
##
##                Accuracy : 0.9
##                  95% CI : (0.795, 0.962)
##     No Information Rate : 0.55
##     P-Value [Acc > NIR] : 4.56e-09
##
##                   Kappa : 0.794
##
##  Mcnemar's Test P-Value : 0.0412
##
##             Sensitivity : 0.778
##             Specificity : 1.000
##          Pos Pred Value : 1.000
##          Neg Pred Value : 0.846
##              Prevalence : 0.450
##          Detection Rate : 0.350
##    Detection Prevalence : 0.350
##       Balanced Accuracy : 0.889
##
##        'Positive' Class : no
##
```

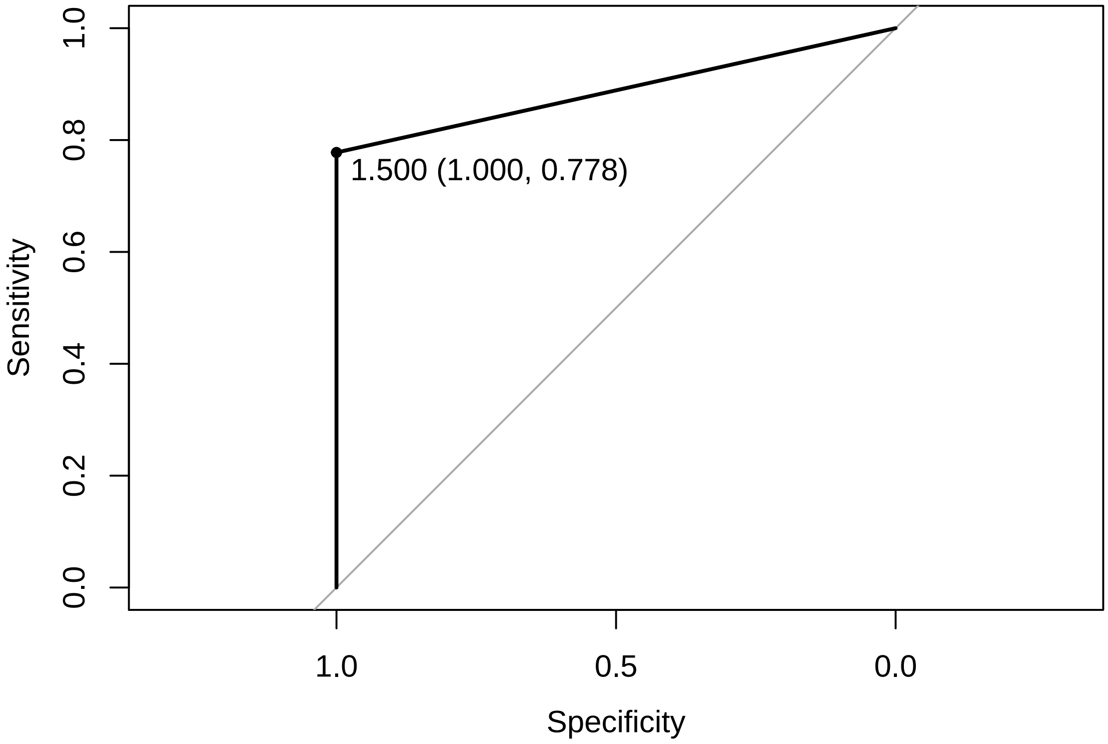
```
# Plot 10 most important variables
plot(varImp(knn_model), top=10, main="Top variables K Nearest Neighbor")
```



```
# Define ROC Curve
knn_rocCurve   <- roc(response = test_data$target,
                      predictor = as.numeric(knn_pred),
                      levels = rev(levels(test_data$target)),
                      plot = TRUE, col = "blue", auc = TRUE)
```



```
# Plot ROC curve
plot(knn_rocCurve, print.thres = "best")
```


1.500 (1.000, 0.778)
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
Specificity
1.0
0.5
0.0

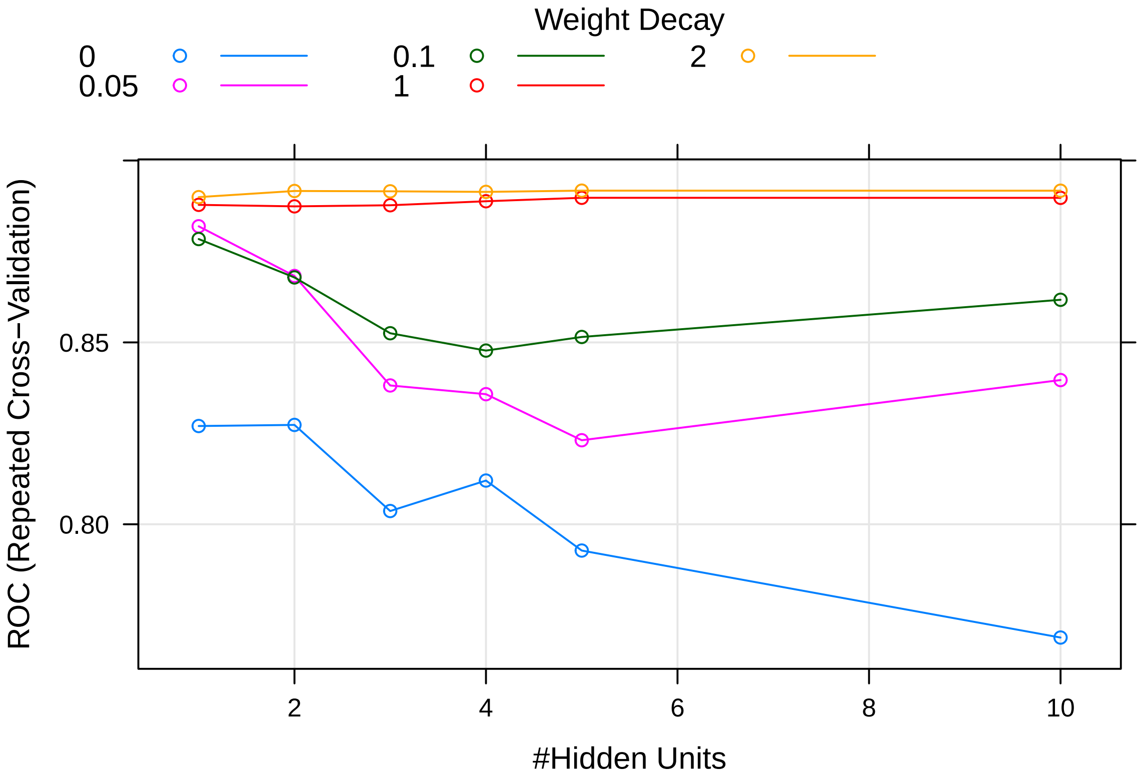
3.6 Neural Network

```
# Set up tuning grid
nn_grid <- expand.grid(size = c(1:5, 10),
                       decay = c(0, 0.05, 0.1, 1, 2))
```

Tuning parameters:

1. size (#Hidden Units)
2. decay (Weight Decay)

```
# Train model
set.seed(1, sample.kind="Rounding")
nn_model <- train(target~., data=train_data,
                  method="nnet",
                  metric="ROC",
                  # center, scale - centering and scaling data
                  preProcess = c("center", "scale"),
                  tuneGrid = nn_grid,
                  trace=FALSE,
                  trControl=fitControl)

plot(nn_model)
```

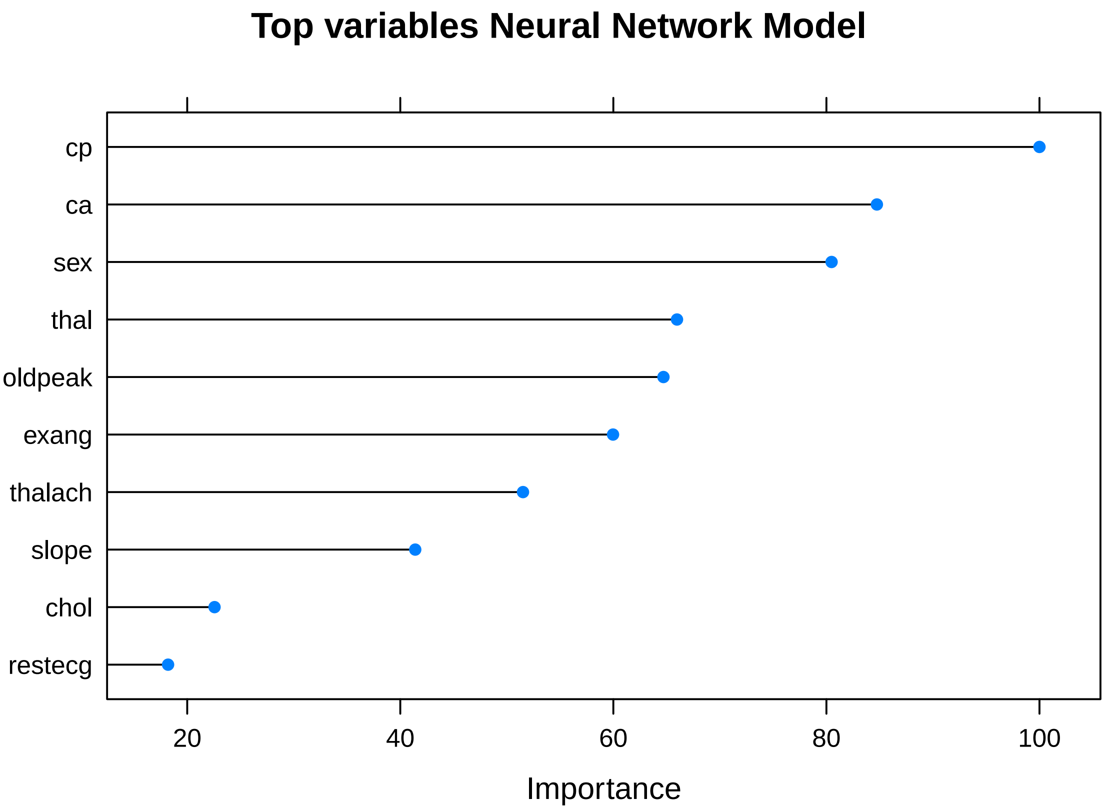


```
# Predict data
set.seed(1, sample.kind="Rounding")
nn_pred <- predict(nn_model, newdata = test_data)

# Evaluate confusion matrix
```

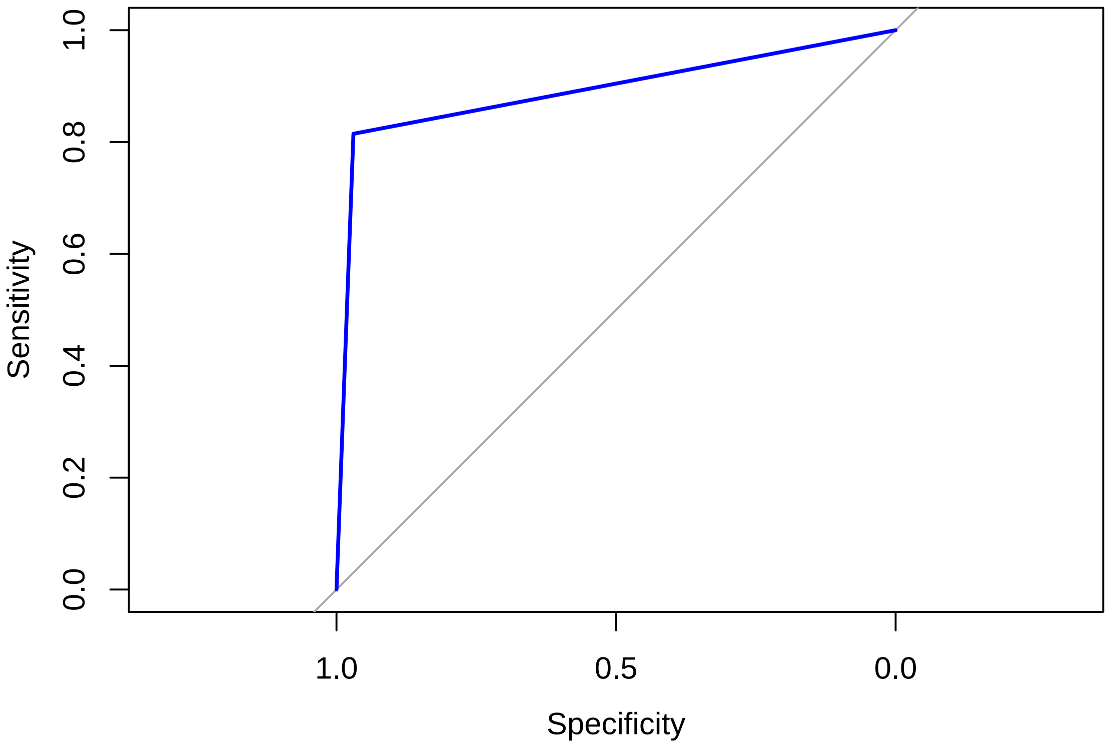
```
nn_confusionMatrix <- confusionMatrix(nn_pred, test_data$target)

nn_confusionMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no  22   1
##        yes  5  32
##
##                Accuracy : 0.9
##                  95% CI : (0.795, 0.962)
##     No Information Rate : 0.55
##     P-Value [Acc > NIR] : 4.56e-09
##
##                   Kappa : 0.795
##
##  Mcnemar's Test P-Value : 0.221
##
##             Sensitivity : 0.815
##             Specificity : 0.970
##          Pos Pred Value : 0.957
##          Neg Pred Value : 0.865
##              Prevalence : 0.450
##          Detection Rate : 0.367
##    Detection Prevalence : 0.383
##       Balanced Accuracy : 0.892
##
##        'Positive' Class : no
##
```

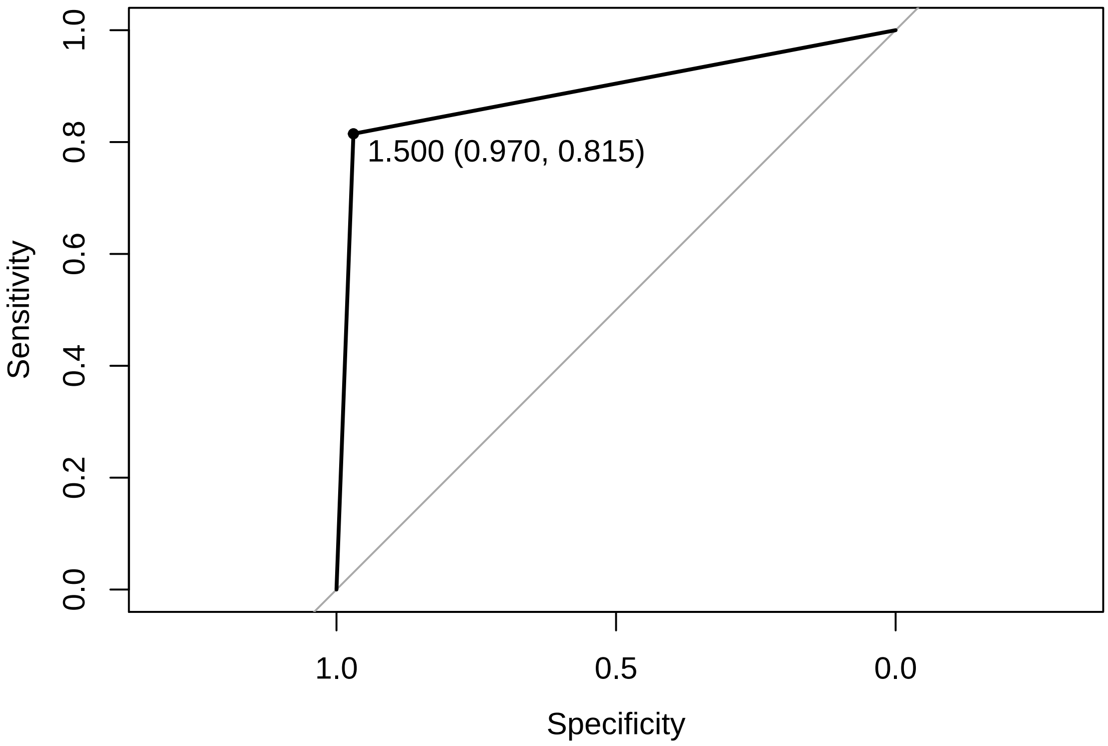
```
# Plot 10 most important variables
plot(varImp(nn_model), top=10, main="Top variables Neural Network Model")
```



```
# Define ROC Curve
nn_rocCurve   <- roc(response = test_data$target,
                     predictor = as.numeric(nn_pred),
                     levels = rev(levels(test_data$target)),
                     plot = TRUE, col = "blue", auc = TRUE)
```



```
# Plot ROC curve
plot(nn_rocCurve, print.thres = "best")
```



3.7 Compare algorithms

In this last phase, are compared the results produced by the various algorithms

```
#List of all algorithms
models_list <- list(Adapt_Boost = am1_model,
                    Gradient_Bost=gbm_model,
                    Class_Tree = ct_model,
                    Random_Forest=rf_model,
                    KNN=knn_model,
                    Neural_Network=nn_model)

models_results <- resamples(models_list)

# Summary of algorithms
summary(models_results)
```

```
##
## Call:
## summary.resamples(object = models_results)
##
## Models: Adapt_Boost, Gradient_Bost, Class_Tree, Random_Forest, KNN, Neural_Network
## Number of resamples: 100
##
## ROC
##                 Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
## Adapt_Boost    0.682   0.822  0.886 0.881   0.940 1.000    0
## Gradient_Bost  0.745   0.846  0.903 0.894   0.944 1.000    0
## Class_Tree     0.549   0.776  0.817 0.821   0.874 0.997    0
## Random_Forest  0.755   0.863  0.925 0.902   0.951 1.000    0
## KNN            0.699   0.860  0.916 0.903   0.944 1.000    0
## Neural_Network 0.706   0.850  0.903 0.892   0.942 1.000    0
##
## Sens
##                 Min. 1st Qu. Median  Mean 3rd Qu. Max. NA's
## Adapt_Boost    0.455   0.727  0.818 0.776   0.852    1    0
## Gradient_Bost  0.455   0.727  0.750 0.765   0.818    1    0
## Class_Tree     0.364   0.727  0.727 0.754   0.818    1    0
## Random_Forest  0.273   0.636  0.750 0.758   0.818    1    0
## KNN            0.182   0.583  0.727 0.681   0.767    1    0
## Neural_Network 0.364   0.636  0.727 0.727   0.818    1    0
##
## Spec
##                 Min. 1st Qu. Median  Mean 3rd Qu. Max. NA's
## Adapt_Boost    0.538   0.769  0.846 0.848   0.923    1    0
## Gradient_Bost  0.538   0.786  0.857 0.870   0.924    1    0
## Class_Tree     0.462   0.692  0.769 0.773   0.846    1    0
## Random_Forest  0.615   0.786  0.846 0.860   0.923    1    0
## KNN            0.615   0.846  0.923 0.913   1.000    1    0
## Neural_Network 0.571   0.846  0.923 0.886   0.929    1    0
```

```
# Confusion matrix of the algorithms
confusion_matrix_list <- list(
  Adapt_Boost=am1_confusionMatrix,
  Gradient_Bost=gbm_confusionMatrix,
  Class_Tree = ct_confusionMatrix,
  Random_Forest=rf_confusionMatrix,
```

```r
  KNN=knn_confusionMatrix,
  Neural_Network=nn_confusionMatrix)

confusion_matrix_list_results <- sapply(confusion_matrix_list, function(x) x$byClass)
confusion_matrix_list_results %>% kable()
```

	Adapt_Boost	Gradient_Bost	Class_Tree	Random_Forest	KNN	Neural_Network
Sensitivity	0.815	0.815	0.889	0.852	0.778	0.815
Specificity	0.879	0.939	0.848	1.000	1.000	0.970
Pos Pred Value	0.846	0.917	0.828	1.000	1.000	0.957
Neg Pred Value	0.853	0.861	0.903	0.892	0.846	0.865
Precision	0.846	0.917	0.828	1.000	1.000	0.957
Recall	0.815	0.815	0.889	0.852	0.778	0.815
F1	0.830	0.863	0.857	0.920	0.875	0.880
Prevalence	0.450	0.450	0.450	0.450	0.450	0.450
Detection Rate	0.367	0.367	0.400	0.383	0.350	0.367
Detection Prevalence	0.433	0.400	0.483	0.383	0.350	0.383
Balanced Accuracy	0.847	0.877	0.869	0.926	0.889	0.892

Classification Trees algorithm has been identified as the best one for ***sensitivity***, instead ***Random Forest*** and ***K Nearest Neighbor*** are the best for ***specificity***

4 Conclusion

The objective of this project was to analyze the Heart Disease dataset to build a classifiers to predict whether people have heart disease or not. In particular was compared the ***classification*** capacity of the algorithms, with particular regard to the topic of ***specificity*** and ***sensitivity***.

We started by analyzing the dataset to understand the structure of the data and in particular was analyzed the correlation between predictors and the possibility of reducing their number

We then selected ROC as the metric to compare algorithms

We identified the algorithms to be implemented to create the system and then evaluated their quality in terms of ***sensitivity*** and ***specificity***.

For each algorithm we evaluate to most important predictors.

Classification Trees algorithm has been identified as the best one for ***sensitivity***, instead ***Random Forest*** and ***K Nearest Neighbor*** are the best performance for ***specificity***.

The present project does not represent an exhaustive analysis, to obtain this result should be considered the possible variants of the implemented algorithms, as well as the opportunity to implement other algorithms among those available for classification problems.