Disease Prediction using Machine Learning

Juilee

3/29/2020

Introduction: The objective of the assignment is to predict whether or not a patient has a certain unspecified disease.It is a binary classification problem.Multiple machine learning algorithms, including naive Bayes classifier, K Nearest Neighbor, Support Vector Machine (with both linear and non-linear kernel functions), Random Forest and Gradient Boosting Classifier to build a disease diagnosis model will be used.

Loading packages

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────────────────────── tidyverse 1.2.1 ──
```

```
## ✔ ggplot2 3.2.1     ✔ purrr   0.3.2
## ✔ tibble  2.1.3     ✔ dplyr   0.8.3
## ✔ tidyr   1.0.0     ✔ stringr 1.4.0
## ✔ readr   1.3.1     ✔ forcats 0.4.0
```

```
## ── Conflicts ────────────────────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
library(ggplot2)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(caretEnsemble)
```

```
##
## Attaching package: 'caretEnsemble'
```

```
## The following object is masked from 'package:ggplot2':
##
##     autoplot
```

```
library(rpart)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(e1071)
library(klaR)
```

```
## Loading required package: MASS
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(naivebayes)
```

```
## naivebayes 0.9.7 loaded
```

```
library(doParallel)
```

```
## Loading required package: foreach
```

```
##
## Attaching package: 'foreach'
```

```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
library(Amelia)
```

```
## Loading required package: Rcpp
```

```
## ##
## ## Amelia II: Multiple Imputation
## ## (Version 1.7.6, built: 2019-11-24)
## ## Copyright (C) 2005-2020 James Honaker, Gary King and Matthew Blackwell
## ## Refer to http://gking.harvard.edu/amelia/ for more information
## ##
```

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:randomForest':
##
##     combine
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(grid)
library(resample)
```

```
## Registered S3 method overwritten by 'resample':
##   method         from
##   print.resample modelr
```

Loading training dataset.

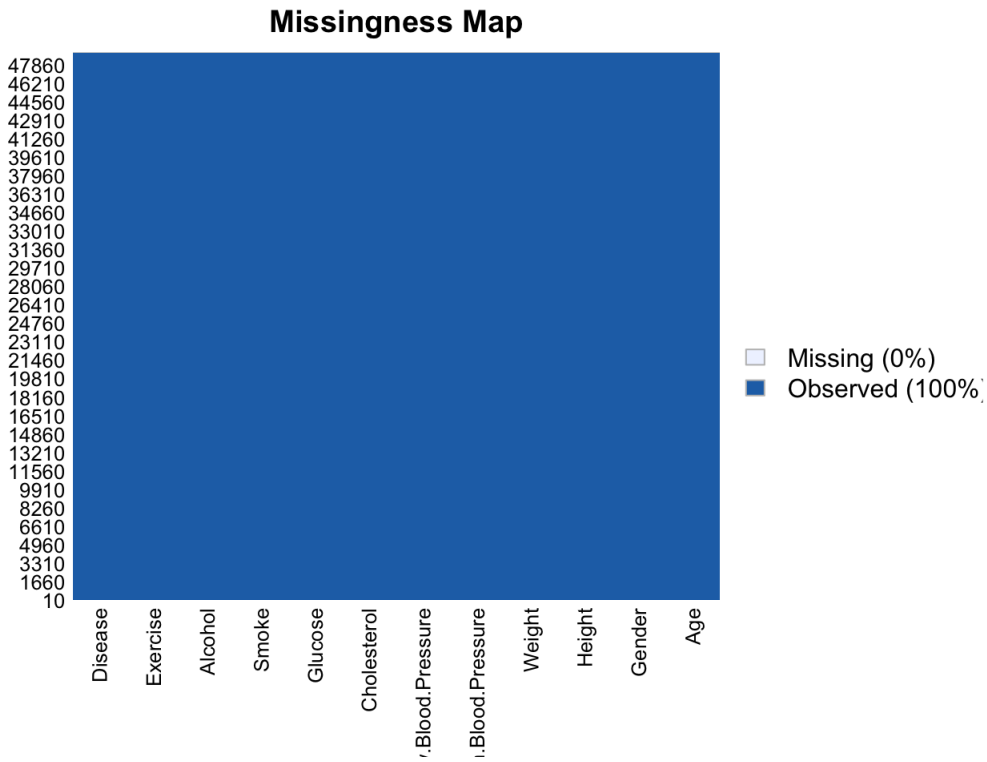
```
d_train<- read.csv("/Users/juilee81/Desktop/DA/DA_HW_03/Disease\ Prediction\ Training.csv",header=TRUE)
#View(d_train)
```

Checking for missing values column-wise and visualize the data:

```
colSums(is.na(d_train))
```

```
##                 Age              Gender              Height
##                   0                   0                   0
##              Weight High.Blood.Pressure  Low.Blood.Pressure
##                   0                   0                   0
##         Cholesterol             Glucose               Smoke
##                   0                   0                   0
##             Alcohol            Exercise             Disease
##                   0                   0                   0
```

```
#visualize the missing data
missmap(d_train)
```



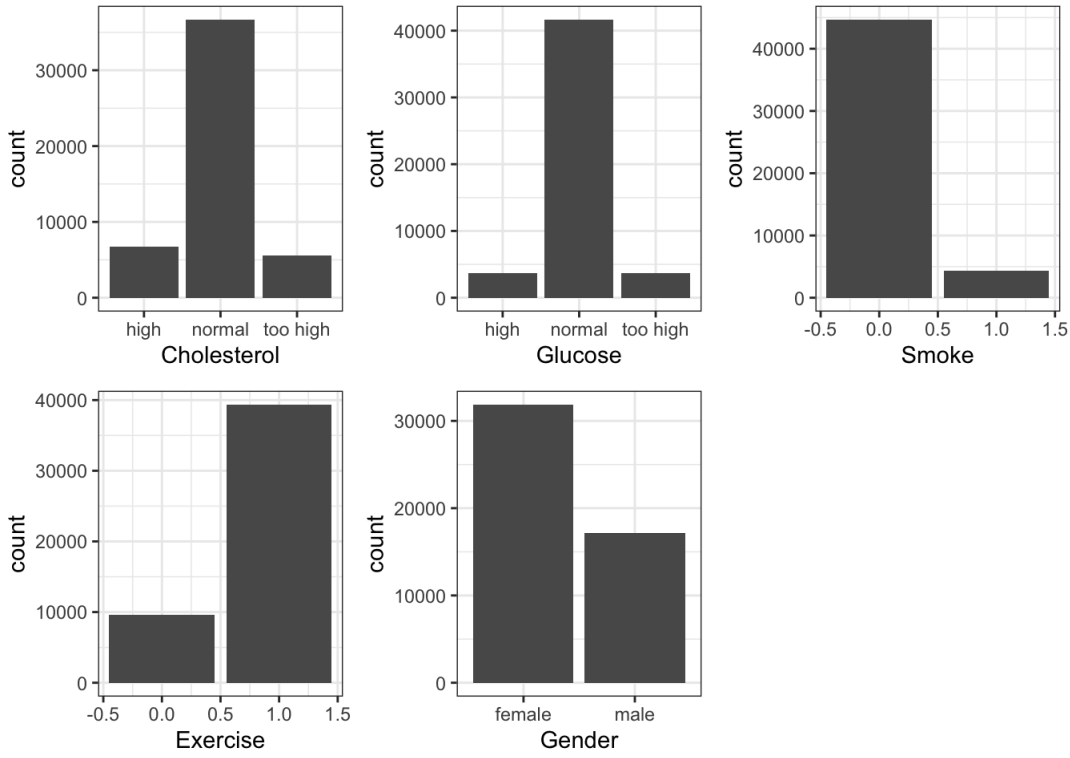
Normalising the numeric data

```
normalize <- function(x) {
return ((x - min(x)) / (max(x) - min(x))) }
```

```
d_train$Age <- normalize(d_train$Age)
d_train$Height <- normalize(d_train$Height)
d_train$Weight <- normalize(d_train$Weight)
d_train$Low.Blood.Pressure <- normalize(d_train$Low.Blood.Pressure)
d_train$High.Blood.Pressure  <- normalize(d_train$High.Blood.Pressure)
```

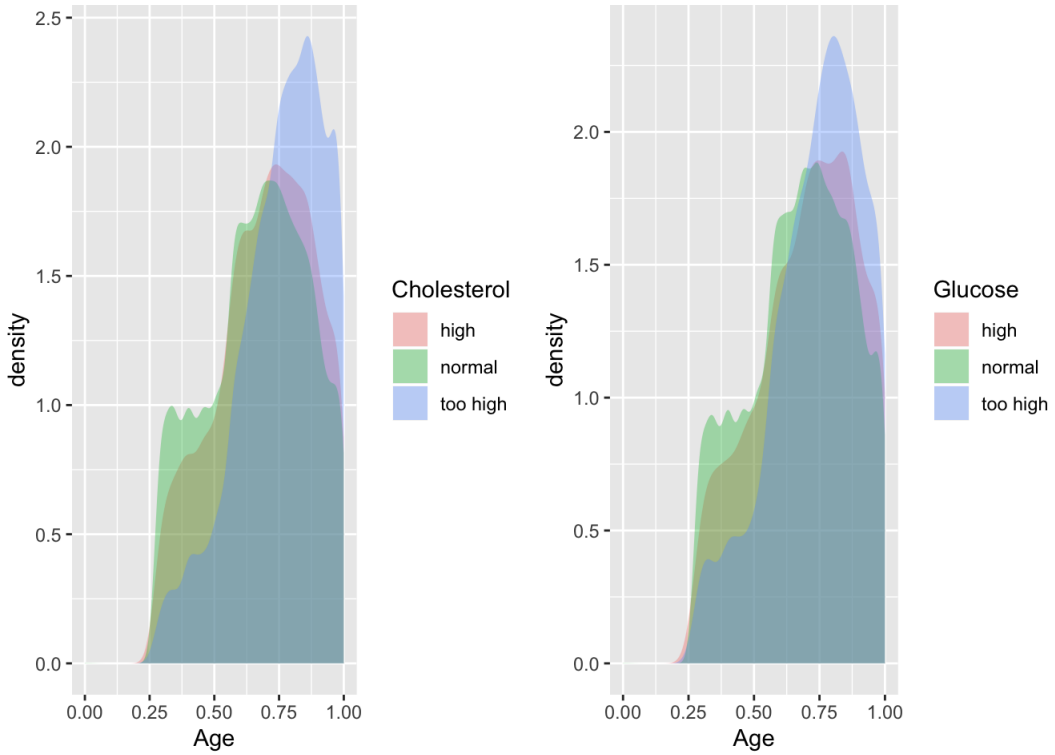
Exploratory Data Analysis:

```
p1<-ggplot(d_train, aes(x=Cholesterol,  fill=Disease)) +
  geom_bar(stat="count",position="dodge")+theme_bw()
p2<-ggplot(d_train, aes(x=Glucose,  fill=Disease)) +
  geom_bar(stat="count",position="dodge")+theme_bw()
p3<-ggplot(d_train, aes(x=Smoke,  fill=Disease)) +
  geom_bar(stat="count",position="dodge")+theme_bw()
p4<-ggplot(d_train, aes(x=Exercise,  fill=Disease)) +
  geom_bar(stat="count",position="dodge")+theme_bw()
p5<-ggplot(d_train, aes(x=Gender,  fill=Disease)) +
  geom_bar(stat="count",position="dodge")+theme_bw()
grid.arrange(p1,p2,p3,p4,p5,ncol=3, nrow= 2)
```




```
t1<-ggplot(d_train,aes(x=Age,fill=Cholesterol))+geom_density(col=NA,alpha=0.35)
t2<-ggplot(d_train,aes(x=Age,fill=Glucose))+geom_density(col=NA,alpha=0.35)

grid.arrange(t1,t2,ncol=2, nrow= 1)
```



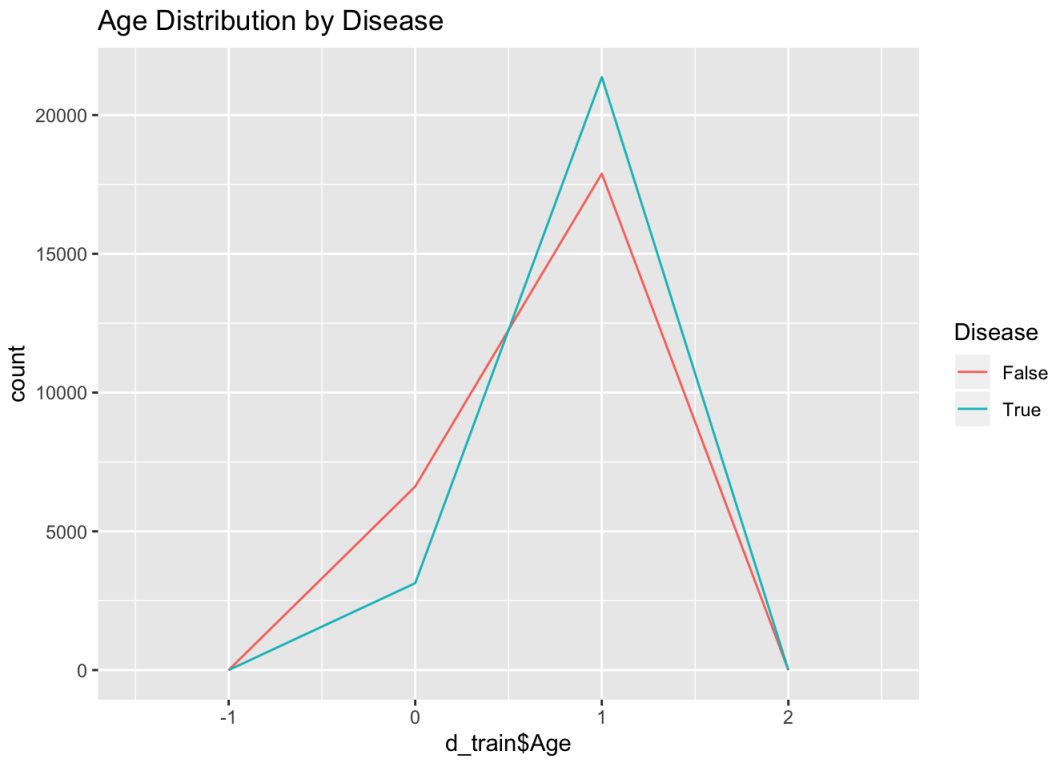
Creating dummy variables of all the categorical variables

```
library(fastDummies)
d_dummies <- fastDummies::dummy_cols(d_train,select_columns=c('Gender','Cholesterol','Glucose'))
d_dummies <- d_dummies[,c(-2,-7,-8)]
```

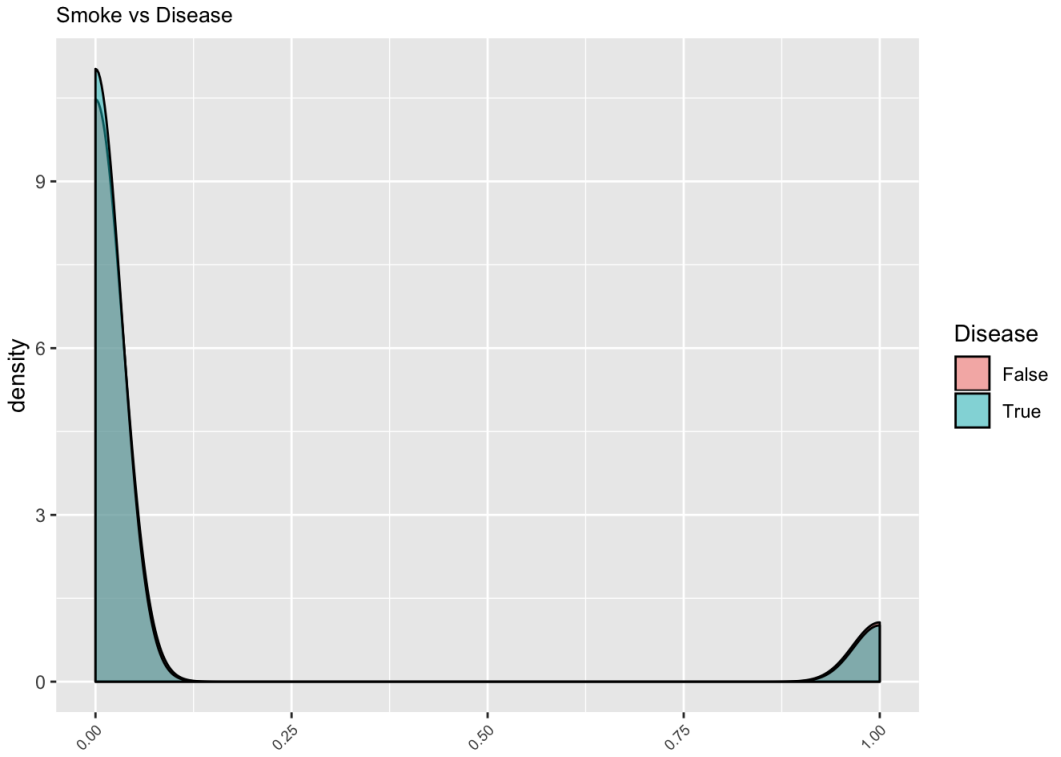
```
d_dummies$Disease <-factor(d_dummies$Disease,labels = c("False", "True"))
```

This is necessary because our output will be in the form of 2 classes, True or False. Where true will denote that a patient has a disease and false denotes that a person is disease free.

```
a<-ggplot(d_dummies, aes(d_train$Age, colour = Disease)) +
geom_freqpoly(binwidth = 1) + labs(title="Age Distribution by Disease")
a
```

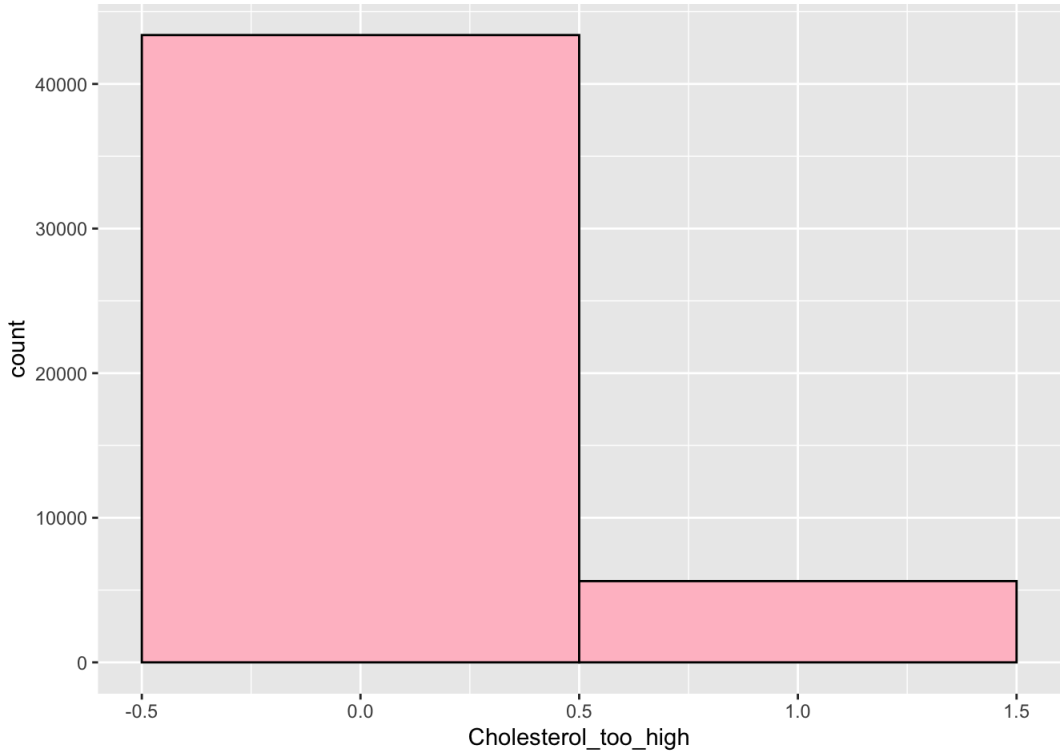


```
b<- d_dummies %>%
              ggplot(aes(x =Smoke, fill = Disease)) +
              geom_density(alpha = 0.5) +
              ggtitle("Smoke vs Disease") +
              theme(plot.title = element_text(size =10),axis.text.x = element_text(size =7,angle = 45, hjus
t = 1),axis.title.x=element_blank())
b
```




```
colnames(d_dummies)[14] <- "Cholesterol_too_high"
colnames(d_dummies)[17] <- "Glucose_too_high"
```

```
c <- ggplot(d_dummies) + geom_histogram(aes(Cholesterol_too_high,fill=Disease), binwidth = 1, fill = "pink",
col = "black")
c
```



```
#Building a model
#split data into training and test data sets
indxTrain <- createDataPartition(y = d_dummies$Disease,p = 0.75,list = FALSE)
training <- d_dummies[indxTrain,]
testing <- d_dummies[-indxTrain,]
prop.table(table(d_dummies$Disease)) * 100
```

```
##
##    False     True
## 50.00408 49.99592
```

```
testing$Disease <- factor(testing$Disease,labels = c("False", "True"))
```

BUILDING MODELS

Building a model : Naive Bayes Classifier

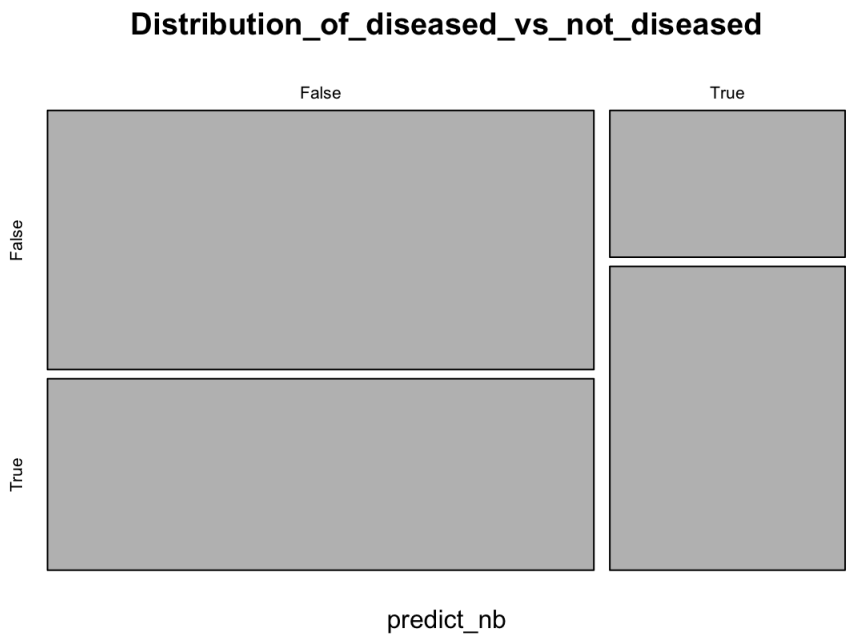
```
#newNBclassifier=naive_bayes(Disease ~.,usekernel=T,data=training)
#print(newNBclassifier)
set.seed(124)
Naive_Bayes_Model <- naiveBayes(Disease ~.,data=training)

predict_nb <- predict(Naive_Bayes_Model, newdata = testing)
Distribution_of_diseased_vs_not_diseased<-table(predict_nb,testing$Disease)
confusionMatrix(predict_nb,testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4924 3638
##      True   1201 2486
##
##                Accuracy : 0.6049
##                  95% CI : (0.5962, 0.6136)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2099
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8039
##             Specificity : 0.4059
##          Pos Pred Value : 0.5751
##          Neg Pred Value : 0.6743
##              Prevalence : 0.5000
##          Detection Rate : 0.4020
##    Detection Prevalence : 0.6990
##       Balanced Accuracy : 0.6049
##
##        'Positive' Class : False
##
```

The final output shows that the Naive Bayes classifier can predict whether a person has a disease or not, with an accuracy of approximately 60%.

```
plot(Distribution_of_diseased_vs_not_diseased)
```



Hyperparameter Tuning for

naive bayes:

```
set.seed(124)
Naive_Bayes_Model_tune <- naiveBayes(Disease ~.,data=training,laplace = 4,metric="Accuracy",type="prob")

predict_nb_tune <- predict(Naive_Bayes_Model_tune, newdata = testing)
confusionMatrix(predict_nb_tune,testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4924 3638
##      True   1201 2486
##
##                Accuracy : 0.6049
##                  95% CI : (0.5962, 0.6136)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2099
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8039
##             Specificity : 0.4059
##          Pos Pred Value : 0.5751
##          Neg Pred Value : 0.6743
##              Prevalence : 0.5000
##          Detection Rate : 0.4020
##    Detection Prevalence : 0.6990
##       Balanced Accuracy : 0.6049
##
##        'Positive' Class : False
##
```

Laplace tuning: The goal is to increase the zero probability values to a small positive number.So that the posterior probabilities don't suddenly drop to zero.

Building a model : KNN

```
set.seed(124)
model_knn <- train(Disease ~ ., data = training, method = "knn")
```

```
predict_knn <- predict(model_knn, newdata = testing)
confusionMatrix(predict_knn, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  3920 2386
##      True   2205 3738
##
##                Accuracy : 0.6252
##                  95% CI : (0.6166, 0.6338)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2504
##
##  Mcnemar's Test P-Value : 0.007894
##
##             Sensitivity : 0.6400
##             Specificity : 0.6104
##          Pos Pred Value : 0.6216
##          Neg Pred Value : 0.6290
##              Prevalence : 0.5000
##          Detection Rate : 0.3200
##    Detection Prevalence : 0.5148
##       Balanced Accuracy : 0.6252
##
##        'Positive' Class : False
##
```

Hyperparameter Tuning for KNN:

```
pre_process <- preProcess(training, method = c("scale", "center"))
pre_process
```

```
## Created from 36751 samples and 17 variables
##
## Pre-processing:
##   - centered (16)
##   - ignored (1)
##   - scaled (16)
```

```
set.seed(124)
model_knn_tune <- train(Disease ~ ., data =training, method = "knn",
                    tuneGrid = data.frame(k = seq(1, 25)),
                    trControl = trainControl(method = "repeatedcv",
                                             number = 3, repeats = 3))

model_knn_tune
```

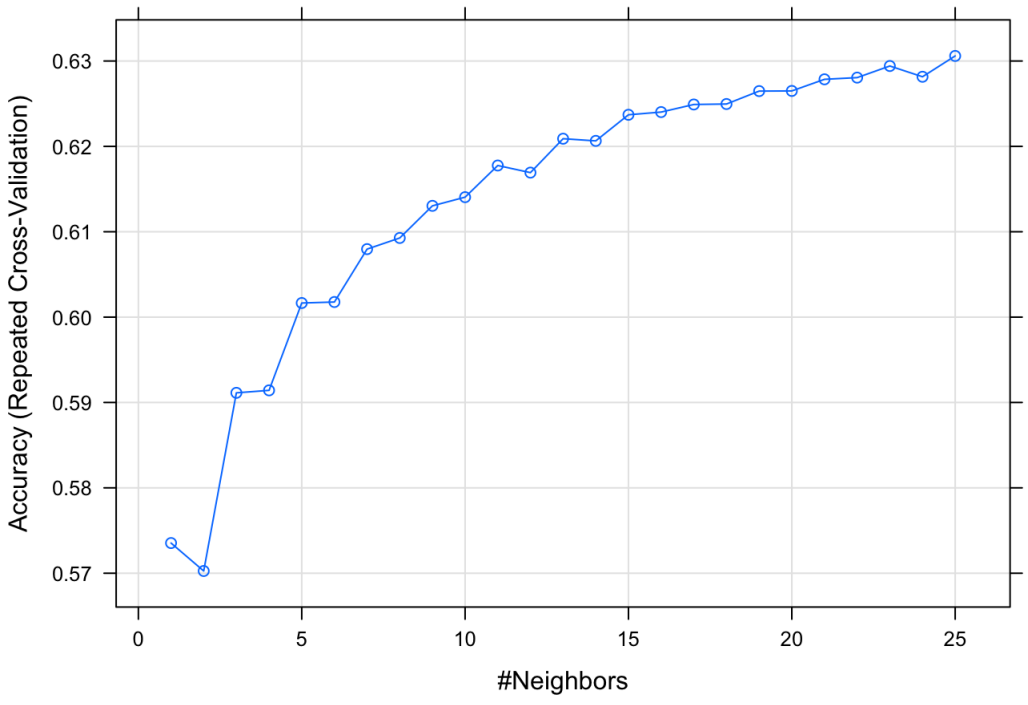
```
## k-Nearest Neighbors
##
## 36751 samples
##    16 predictor
##     2 classes: 'False', 'True'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 3 times)
## Summary of sample sizes: 24501, 24500, 24501, 24502, 24500, 24500, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.5735354  0.1470698
##    2  0.5702610  0.1405208
##    3  0.5911312  0.1822614
##    4  0.5914215  0.1828419
##    5  0.6016525  0.2033046
##    6  0.6017703  0.2035402
##    7  0.6079653  0.2159300
##    8  0.6092714  0.2185422
##    9  0.6130354  0.2260702
##   10  0.6140512  0.2281016
##   11  0.6177609  0.2355208
##   12  0.6169265  0.2338518
##   13  0.6208991  0.2417972
##   14  0.6206453  0.2412894
##   15  0.6237018  0.2474026
##   16  0.6240193  0.2480374
##   17  0.6249082  0.2498152
##   18  0.6249626  0.2499240
##   19  0.6264773  0.2529534
##   20  0.6264954  0.2529896
##   21  0.6278559  0.2557107
##   22  0.6280555  0.2561097
##   23  0.6294159  0.2588307
##   24  0.6281552  0.2563092
##   25  0.6305951  0.2611888
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 25.
```

1.**k:** The performance of KNNs is very sensitive to the choice of k.For high signal data with very few noisy (irrelevant) features, smaller values of k tend to work best. As more irrelevant features are involved, larger values of k are required to smooth out the noise.

```
predict_knn_tune <- predict(model_knn_tune, newdata = testing)
confusionMatrix(predict_knn_tune,testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  3982 2338
##      True   2143 3786
##
##                Accuracy : 0.6342
##                  95% CI : (0.6256, 0.6427)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2683
##
##  Mcnemar's Test P-Value : 0.003754
##
##             Sensitivity : 0.6501
##             Specificity : 0.6182
##          Pos Pred Value : 0.6301
##          Neg Pred Value : 0.6386
##              Prevalence : 0.5000
##          Detection Rate : 0.3251
##    Detection Prevalence : 0.5160
##       Balanced Accuracy : 0.6342
##
##        'Positive' Class : False
##
```

```
plot(model_knn_tune)
```



Building a model : SVM-Linear

```
set.seed(124)
model_svm_linear <- train(Disease ~ ., data = training,method = "svmLinear")
model_svm_linear
```

```
## Support Vector Machines with Linear Kernel
##
## 36751 samples
##    16 predictor
##     2 classes: 'False', 'True'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 36751, 36751, 36751, 36751, 36751, 36751, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7235286  0.4470159
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
predict_svm_linear <- predict(model_svm_linear, newdata = testing)
confusionMatrix(predict_svm_linear, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  5023 2250
##      True   1102 3874
##
##                Accuracy : 0.7263
##                  95% CI : (0.7184, 0.7342)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4527
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8201
##             Specificity : 0.6326
##          Pos Pred Value : 0.6906
##          Neg Pred Value : 0.7785
##              Prevalence : 0.5000
##          Detection Rate : 0.4101
##    Detection Prevalence : 0.5938
##       Balanced Accuracy : 0.7263
##
##        'Positive' Class : False
##
```

Hyperparameter Tuning for SVM-Linear:

```
trctrl <- trainControl(method = "cv",number=3,repeats = 2)
```

```
## Warning: `repeats` has no meaning for this resampling method.
```

```
set.seed(3233)
unwantedoutput1 <- capture.output(model_svm_linear_tune <- train(Disease ~ ., data =training,
                          method = "svmLinear",
                          metric="Accuracy",
                          trControl=trctrl,
                          tuneLength = 10,
                          tuneGrid = expand.grid(C = seq(0.5, 1, 2))))
print(model_svm_linear_tune)
```

```
## Support Vector Machines with Linear Kernel
##
## 36751 samples
##    16 predictor
##     2 classes: 'False', 'True'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 24501, 24501, 24500
## Resampling results:
##
##   Accuracy   Kappa
##   0.7257216  0.4514352
##
## Tuning parameter 'C' was held constant at a value of 0.5
```

```
predict_svm_linear_tune <- predict(model_svm_linear_tune, newdata = testing)
confusionMatrix(predict_svm_linear_tune, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  5018 2244
##      True   1107 3880
##
##                Accuracy : 0.7264
##                  95% CI : (0.7184, 0.7343)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4528
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8193
##             Specificity : 0.6336
##          Pos Pred Value : 0.6910
##          Neg Pred Value : 0.7780
##              Prevalence : 0.5000
##          Detection Rate : 0.4097
##    Detection Prevalence : 0.5929
##       Balanced Accuracy : 0.7264
##
##        'Positive' Class : False
##
```

1.For a linear kernel, the choice of C does not seem to affect performance very much.

Building a model : Non linear SVM

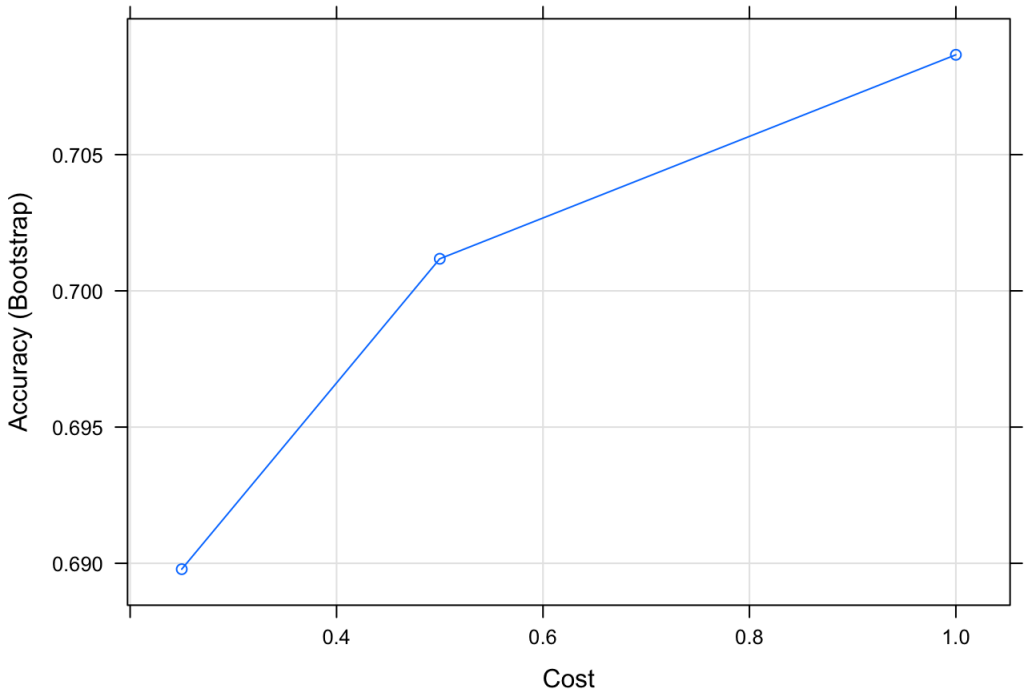
```
set.seed(124)
stratified_data <- training %>% sample_frac(0.5)
model_svm_rbf <- train(Disease ~ ., data = stratified_data,method="svmRadial")
model_svm_rbf
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 18376 samples
##    16 predictor
##     2 classes: 'False', 'True'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 18376, 18376, 18376, 18376, 18376, 18376, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.6897847  0.3796214
##   0.50  0.7011774  0.4024099
##   1.00  0.7086642  0.4173922
##
## Tuning parameter 'sigma' was held constant at a value of 0.09694431
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.09694431 and C = 1.
```

```
predict_svm_rbf <- predict(model_svm_rbf, newdata = testing)
confusionMatrix(predict_svm_rbf, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4520 1876
##      True   1605 4248
##
##                Accuracy : 0.7158
##                  95% CI : (0.7077, 0.7238)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4316
##
##  Mcnemar's Test P-Value : 4.733e-06
##
##             Sensitivity : 0.7380
##             Specificity : 0.6937
##          Pos Pred Value : 0.7067
##          Neg Pred Value : 0.7258
##              Prevalence : 0.5000
##          Detection Rate : 0.3690
##    Detection Prevalence : 0.5222
##       Balanced Accuracy : 0.7158
##
##        'Positive' Class : False
##
```

```
plot(model_svm_rbf)
```



Hyperparameter Tuning for

SVM-Non-Linear:

```
set.seed(124)
g <- expand.grid(sigma= seq(0.1,0.3,0.1), C= seq(0.5,1.5,0.5))
tr <- trainControl(method="cv",number = 3,repeats = 1)
```

```
## Warning: `repeats` has no meaning for this resampling method.
```

```
model_svm_rbf_tune <- train(Disease ~ ., data = stratified_data,
                            metric='Accuracy',
                            tuneGrid = g ,
                            method ="svmRadial",
                            trControl=tr)
model_svm_rbf_tune
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 18376 samples
##    16 predictor
##     2 classes: 'False', 'True'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 12251, 12250, 12251
## Resampling results across tuning parameters:
##
##   sigma  C    Accuracy   Kappa
##   0.1    0.5  0.6942755  0.3885823
##   0.1    1.0  0.7069006  0.4138343
##   0.1    1.5  0.7110364  0.4221184
##   0.2    0.5  0.6905206  0.3810278
##   0.2    1.0  0.7005336  0.4010760
##   0.2    1.5  0.7039076  0.4078387
##   0.3    0.5  0.6866568  0.3732720
##   0.3    1.0  0.6956358  0.3912682
##   0.3    1.5  0.6999349  0.3998881
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.1 and C = 1.5.
```

1. C: The C parameter controls how much you want to punish your model for each misclassified point for a given curve.Lower values of the C parameter allow the classifier to learn better under noisy data.
2. Sigma:Smaller sigma tends to be less bias and more variance while larger sigma tends to be less variance and more bias.Parameter

defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’.

```
predict_svm_rbf_tune <- predict(model_svm_rbf_tune, newdata = testing)
confusionMatrix(predict_svm_rbf_tune, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4542 1878
##      True   1583 4246
##
##                Accuracy : 0.7174
##                  95% CI : (0.7094, 0.7254)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4349
##
##  Mcnemar's Test P-Value : 5.81e-07
##
##             Sensitivity : 0.7416
##             Specificity : 0.6933
##          Pos Pred Value : 0.7075
##          Neg Pred Value : 0.7284
##              Prevalence : 0.5000
##          Detection Rate : 0.3708
##    Detection Prevalence : 0.5241
##       Balanced Accuracy : 0.7174
##
##        'Positive' Class : False
##
```

Building a model : Random forest

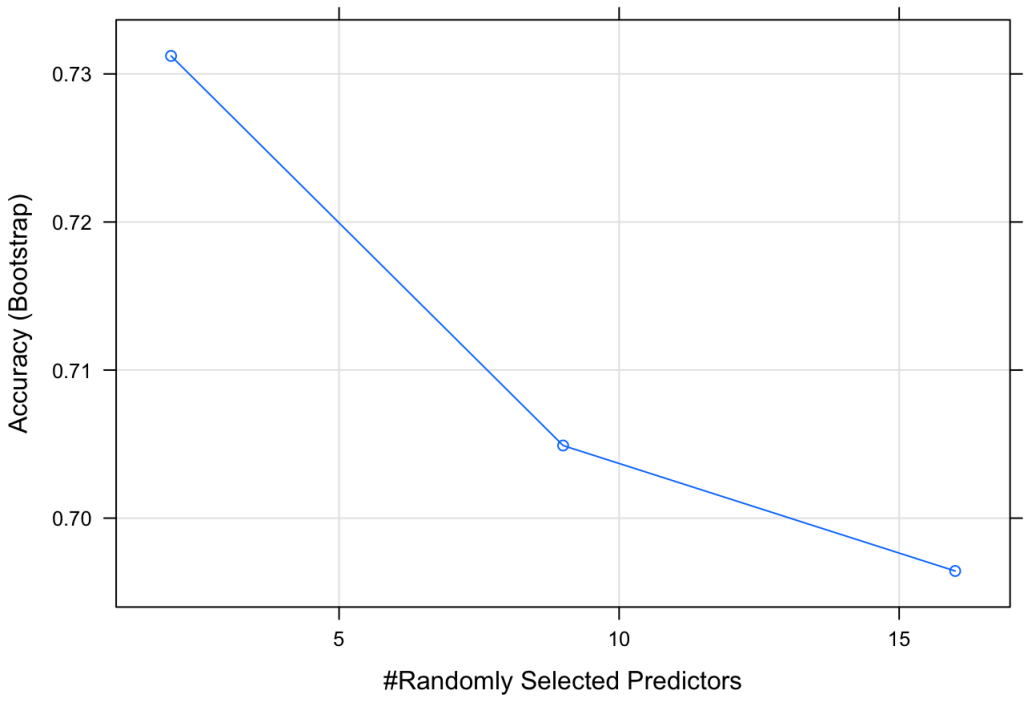
```
set.seed(124)
model_rf <- train(Disease ~ ., data = training, method = "rf")
model_rf
```

```
## Random Forest
##
## 36751 samples
##    16 predictor
##     2 classes: 'False', 'True'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 36751, 36751, 36751, 36751, 36751, 36751, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.7312160  0.4624142
##    9    0.7049058  0.4098188
##   16    0.6964306  0.3928758
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
predict_rf <- predict(model_rf, newdata = testing)
confusionMatrix(predict_rf, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4914 2002
##      True   1211 4122
##
##                Accuracy : 0.7377
##                  95% CI : (0.7298, 0.7455)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4754
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8023
##             Specificity : 0.6731
##          Pos Pred Value : 0.7105
##          Neg Pred Value : 0.7729
##              Prevalence : 0.5000
##          Detection Rate : 0.4012
##    Detection Prevalence : 0.5646
##       Balanced Accuracy : 0.7377
##
##        'Positive' Class : False
##
```

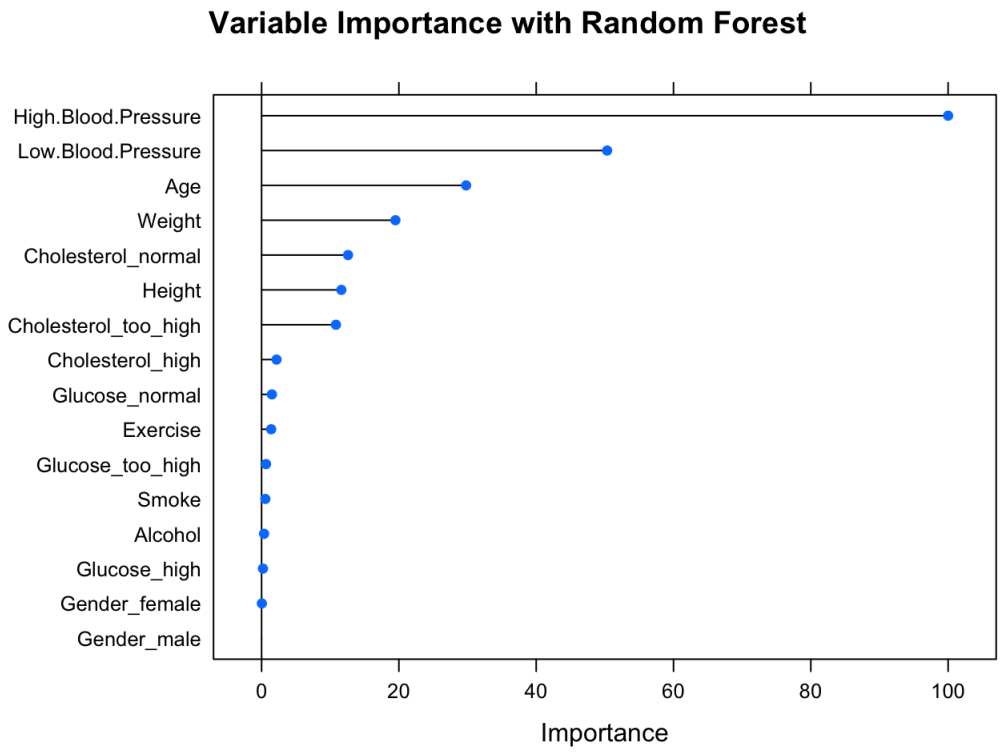
```
plot(model_rf)
```




```
varimp_rf <- varImp(model_rf)
varimp_rf
```

```
## rf variable importance
##
##                       Overall
## High.Blood.Pressure  100.0000
## Low.Blood.Pressure    50.3321
## Age                   29.7921
## Weight                19.4968
## Cholesterol_normal    12.5957
## Height                11.6186
## Cholesterol_too_high  10.8305
## Cholesterol_high       2.1781
## Glucose_normal         1.5036
## Exercise               1.3956
## Glucose_too_high       0.6546
## Smoke                  0.5482
## Alcohol                0.3626
## Glucose_high           0.1995
## Gender_female          0.0395
## Gender_male            0.0000
```

```
plot(varimp_rf, main = "Variable Importance with Random Forest")
```



Hyperparameter Tuning for

Random forest:

```
control <- trainControl(method="cv", number=3, repeats=1)
```

```
## Warning: `repeats` has no meaning for this resampling method.
```

```
mtry <- c(1,2,5,10)
tunegrid <- expand.grid(.mtry=mtry)
```

Each axis of the grid is an algorithm parameter, and points in the grid are specific combinations of parameters. Because we are only tuning one parameter, the grid search is a linear search through a vector of candidate values.mtry parameter is available in caret for tuning. 1. **mtry:** Number of variables randomly sampled as candidates at each split.

```
set.seed(124)
model_rf_tune <- train(Disease~., data=training, method="rf",metric='Accuracy',tuneGrid=tunegrid, trControl=
control)
model_rf_tune
```

```
## Random Forest
##
## 36751 samples
##    16 predictor
##     2 classes: 'False', 'True'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 24501, 24500, 24501
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    1    0.7095045  0.4189989
##    2    0.7309188  0.4618322
##    5    0.7269733  0.4539433
##   10    0.7090418  0.4180832
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
predict_rf_tune <- predict(model_rf_tune, newdata = testing)
confusionMatrix(predict_rf_tune, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4923 2019
##      True   1202 4105
##
##                Accuracy : 0.737
##                  95% CI : (0.7291, 0.7448)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4741
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8038
##             Specificity : 0.6703
##          Pos Pred Value : 0.7092
##          Neg Pred Value : 0.7735
##              Prevalence : 0.5000
##          Detection Rate : 0.4019
##    Detection Prevalence : 0.5667
##       Balanced Accuracy : 0.7370
##
##        'Positive' Class : False
##
```

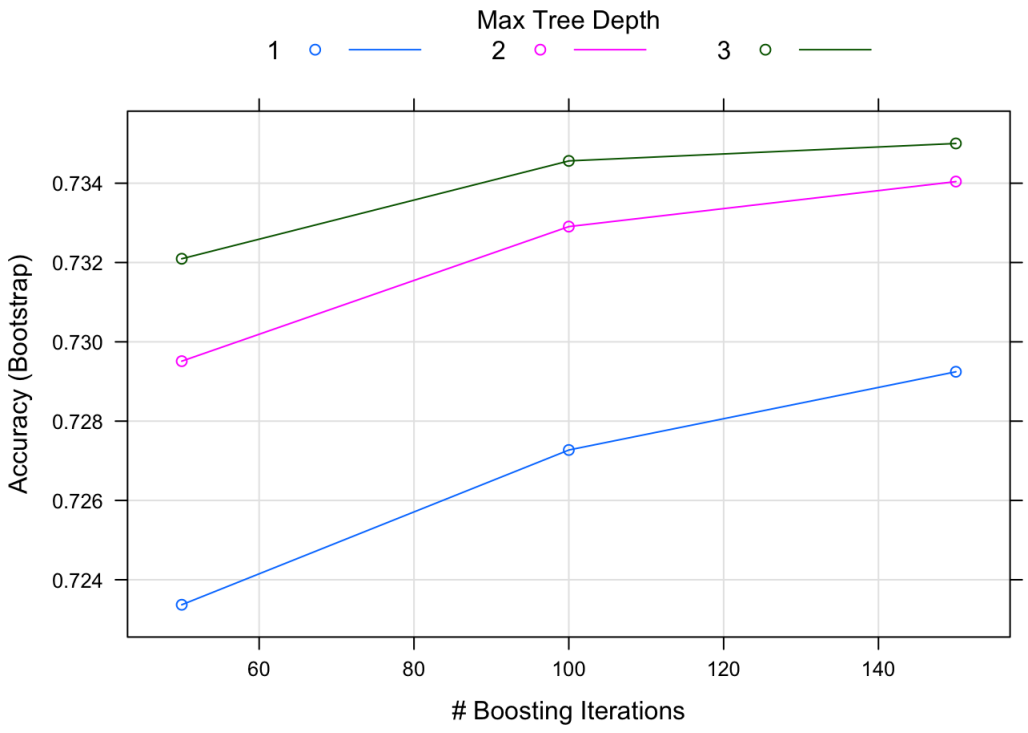
Building a model : Gradient Boosting

```
set.seed(124)
unwantedoutput <- capture.output(model_gbm <- train(Disease ~ ., data = training, method = "gbm"))
```

```
predict_gbm <- predict(model_gbm, newdata = testing)
confusionMatrix(predict_gbm, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4759 1816
##      True   1366 4308
##
##                Accuracy : 0.7402
##                  95% CI : (0.7324, 0.748)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4804
##
##  Mcnemar's Test P-Value : 1.725e-15
##
##             Sensitivity : 0.7770
##             Specificity : 0.7035
##          Pos Pred Value : 0.7238
##          Neg Pred Value : 0.7593
##              Prevalence : 0.5000
##          Detection Rate : 0.3885
##    Detection Prevalence : 0.5368
##       Balanced Accuracy : 0.7402
##
##        'Positive' Class : False
##
```

```
plot(model_gbm)
```



Hyperparameter Tuning for

Gradient Boosting:

```
#install.packages("doParallel")
control <- trainControl(method="repeatedcv", number=3, repeats=3)
tgrid<- expand.grid(n.trees =c(1080:1100),
                    interaction.depth=c(1:3),
                    shrinkage=c(0.001,0.2,0.3),
                    n.minobsinnode=15)
```

1.**n.trees:** The total number of trees in the sequence or ensemble.Since they can easily overfit if there are many number of trees,we must find the optimal number of trees that minimize the loss function of interest with cross validation. 2.**shrinkage:** Determines the contribution of each tree on the final outcome and controls how quickly the algorithm proceeds down the gradient descent.Generally, the smaller this value, the more accurate the model can be but also will require more trees in the sequence. 3.**interaction.depth:** Controls the depth of the

individual trees.Higher depth trees allow the algorithm to capture unique interactions but also increase the risk of over-fitting.
4.**n.minobsinnode:** Controls the complexity of each tree.Higher values help prevent a model from learning relationships which might be highly specific to the particular sample selected for a tree (overfitting) but smaller values can help with imbalanced target classes in classification problems.

```
set.seed(124) #for reproducability
unwantedoutput <- capture.output(model_gbm_tune <- train(Disease ~ ., data = training, method = "gbm",metric
="Accuracy",tuneGrid=tgrid,trControl=control))
print(model_gbm_tune)
```

```
## Stochastic Gradient Boosting
##
## 36751 samples
##    16 predictor
##     2 classes: 'False', 'True'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 3 times)
## Summary of sample sizes: 24501, 24500, 24501, 24502, 24500, 24500, ...
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.trees  Accuracy   Kappa
##   0.001      1                  1080     0.7159261  0.4318439
##   0.001      1                  1081     0.7159261  0.4318439
##   0.001      1                  1082     0.7159261  0.4318439
##   0.001      1                  1083     0.7159261  0.4318439
##   0.001      1                  1084     0.7159261  0.4318439
##   0.001      1                  1085     0.7159261  0.4318439
##   0.001      1                  1086     0.7159261  0.4318439
##   0.001      1                  1087     0.7159261  0.4318439
##   0.001      1                  1088     0.7159261  0.4318439
##   0.001      1                  1089     0.7159261  0.4318439
##   0.001      1                  1090     0.7159261  0.4318439
##   0.001      1                  1091     0.7159261  0.4318439
##   0.001      1                  1092     0.7159261  0.4318439
##   0.001      1                  1093     0.7159261  0.4318439
##   0.001      1                  1094     0.7159261  0.4318439
##   0.001      1                  1095     0.7159261  0.4318439
##   0.001      1                  1096     0.7159261  0.4318439
##   0.001      1                  1097     0.7159261  0.4318439
##   0.001      1                  1098     0.7159261  0.4318439
##   0.001      1                  1099     0.7159261  0.4318439
##   0.001      1                  1100     0.7159261  0.4318439
##   0.001      2                  1080     0.7253952  0.4507847
##   0.001      2                  1081     0.7253952  0.4507847
##   0.001      2                  1082     0.7253952  0.4507847
##   0.001      2                  1083     0.7253952  0.4507847
##   0.001      2                  1084     0.7253952  0.4507847
##   0.001      2                  1085     0.7253952  0.4507847
##   0.001      2                  1086     0.7253952  0.4507847
##   0.001      2                  1087     0.7253952  0.4507847
##   0.001      2                  1088     0.7253952  0.4507847
##   0.001      2                  1089     0.7253952  0.4507847
##   0.001      2                  1090     0.7253952  0.4507847
##   0.001      2                  1091     0.7253952  0.4507847
##   0.001      2                  1092     0.7253952  0.4507847
##   0.001      2                  1093     0.7253952  0.4507847
##   0.001      2                  1094     0.7253952  0.4507847
##   0.001      2                  1095     0.7253952  0.4507847
##   0.001      2                  1096     0.7253952  0.4507847
##   0.001      2                  1097     0.7253952  0.4507847
##   0.001      2                  1098     0.7253952  0.4507847
##   0.001      2                  1099     0.7253952  0.4507847
##   0.001      2                  1100     0.7253952  0.4507847
##   0.001      3                  1080     0.7257761  0.4515469
##   0.001      3                  1081     0.7257761  0.4515469
##   0.001      3                  1082     0.7257761  0.4515469
##   0.001      3                  1083     0.7257761  0.4515469
##   0.001      3                  1084     0.7257761  0.4515469
##   0.001      3                  1085     0.7257761  0.4515469
##   0.001      3                  1086     0.7257761  0.4515469
##   0.001      3                  1087     0.7257761  0.4515469
```

```
##   0.001      3                  1088     0.7257761  0.4515469
##   0.001      3                  1089     0.7257761  0.4515469
##   0.001      3                  1090     0.7257761  0.4515469
##   0.001      3                  1091     0.7257852  0.4515650
##   0.001      3                  1092     0.7257852  0.4515650
##   0.001      3                  1093     0.7257852  0.4515650
##   0.001      3                  1094     0.7257580  0.4515106
##   0.001      3                  1095     0.7257489  0.4514925
##   0.001      3                  1096     0.7257580  0.4515106
##   0.001      3                  1097     0.7257580  0.4515106
##   0.001      3                  1098     0.7257580  0.4515106
##   0.001      3                  1099     0.7257580  0.4515106
##   0.001      3                  1100     0.7257762  0.4515469
##   0.200      1                  1080     0.7331228  0.4662398
##   0.200      1                  1081     0.7330775  0.4661491
##   0.200      1                  1082     0.7330140  0.4660221
##   0.200      1                  1083     0.7329324  0.4658588
##   0.200      1                  1084     0.7331047  0.4662035
##   0.200      1                  1085     0.7331682  0.4663305
##   0.200      1                  1086     0.7330594  0.4661128
##   0.200      1                  1087     0.7330503  0.4660947
##   0.200      1                  1088     0.7330775  0.4661491
##   0.200      1                  1089     0.7331591  0.4663124
##   0.200      1                  1090     0.7331229  0.4662398
##   0.200      1                  1091     0.7331138  0.4662216
##   0.200      1                  1092     0.7332136  0.4664212
##   0.200      1                  1093     0.7331591  0.4663123
##   0.200      1                  1094     0.7331682  0.4663305
##   0.200      1                  1095     0.7330503  0.4660947
##   0.200      1                  1096     0.7332045  0.4664031
##   0.200      1                  1097     0.7332408  0.4664757
##   0.200      1                  1098     0.7331863  0.4663668
##   0.200      1                  1099     0.7330956  0.4661854
##   0.200      1                  1100     0.7332498  0.4664938
##   0.200      2                  1080     0.7339754  0.4679480
##   0.200      2                  1081     0.7337033  0.4674038
##   0.200      2                  1082     0.7336579  0.4673131
##   0.200      2                  1083     0.7336307  0.4672587
##   0.200      2                  1084     0.7335945  0.4671861
##   0.200      2                  1085     0.7336761  0.4673494
##   0.200      2                  1086     0.7339210  0.4678392
##   0.200      2                  1087     0.7338847  0.4677665
##   0.200      2                  1088     0.7339573  0.4679116
##   0.200      2                  1089     0.7339119  0.4678209
##   0.200      2                  1090     0.7339300  0.4678571
##   0.200      2                  1091     0.7340026  0.4680023
##   0.200      2                  1092     0.7338031  0.4676032
##   0.200      2                  1093     0.7338031  0.4676032
##   0.200      2                  1094     0.7339482  0.4678935
##   0.200      2                  1095     0.7340661  0.4681293
##   0.200      2                  1096     0.7339119  0.4678209
##   0.200      2                  1097     0.7340298  0.4680568
##   0.200      2                  1098     0.7339391  0.4678754
##   0.200      2                  1099     0.7339844  0.4679660
##   0.200      2                  1100     0.7340207  0.4680386
##   0.200      3                  1080     0.7309007  0.4617984
##   0.200      3                  1081     0.7309460  0.4618891
##   0.200      3                  1082     0.7309007  0.4617985
##   0.200      3                  1083     0.7308644  0.4617259
##   0.200      3                  1084     0.7307918  0.4615808
##   0.200      3                  1085     0.7307374  0.4614720
##   0.200      3                  1086     0.7308190  0.4616353
##   0.200      3                  1087     0.7307918  0.4615808
##   0.200      3                  1088     0.7307102  0.4614175
##   0.200      3                  1089     0.7309551  0.4619074
##   0.200      3                  1090     0.7308644  0.4617260
##   0.200      3                  1091     0.7309007  0.4617985
##   0.200      3                  1092     0.7307102  0.4614176
##   0.200      3                  1093     0.7304562  0.4609096
##   0.200      3                  1094     0.7304199  0.4608371
##   0.200      3                  1095     0.7307465  0.4614901
##   0.200      3                  1096     0.7308190  0.4616352
```

```
##   0.200      3                  1097     0.7307102  0.4614175
##   0.200      3                  1098     0.7307827  0.4615627
##   0.200      3                  1099     0.7309914  0.4619798
##   0.200      3                  1100     0.7308734  0.4617440
##   0.300      1                  1080     0.7329687  0.4659313
##   0.300      1                  1081     0.7328326  0.4656593
##   0.300      1                  1082     0.7326875  0.4653690
##   0.300      1                  1083     0.7327056  0.4654053
##   0.300      1                  1084     0.7323610  0.4647158
##   0.300      1                  1085     0.7324245  0.4648428
##   0.300      1                  1086     0.7326059  0.4652056
##   0.300      1                  1087     0.7326240  0.4652419
##   0.300      1                  1088     0.7326331  0.4652601
##   0.300      1                  1089     0.7326603  0.4653146
##   0.300      1                  1090     0.7327963  0.4655867
##   0.300      1                  1091     0.7325514  0.4650969
##   0.300      1                  1092     0.7324154  0.4648248
##   0.300      1                  1093     0.7323610  0.4647159
##   0.300      1                  1094     0.7322612  0.4645163
##   0.300      1                  1095     0.7322068  0.4644076
##   0.300      1                  1096     0.7319437  0.4638815
##   0.300      1                  1097     0.7319709  0.4639359
##   0.300      1                  1098     0.7321977  0.4643894
##   0.300      1                  1099     0.7320435  0.4640810
##   0.300      1                  1100     0.7321705  0.4643350
##   0.300      2                  1080     0.7321342  0.4642653
##   0.300      2                  1081     0.7320435  0.4640840
##   0.300      2                  1082     0.7321161  0.4642292
##   0.300      2                  1083     0.7323065  0.4646101
##   0.300      2                  1084     0.7322430  0.4644832
##   0.300      2                  1085     0.7322702  0.4645376
##   0.300      2                  1086     0.7321342  0.4642655
##   0.300      2                  1087     0.7322249  0.4644469
##   0.300      2                  1088     0.7321614  0.4643199
##   0.300      2                  1089     0.7322067  0.4644106
##   0.300      2                  1090     0.7320707  0.4641385
##   0.300      2                  1091     0.7318349  0.4636668
##   0.300      2                  1092     0.7317623  0.4635216
##   0.300      2                  1093     0.7315991  0.4631951
##   0.300      2                  1094     0.7318621  0.4637211
##   0.300      2                  1095     0.7319437  0.4638844
##   0.300      2                  1096     0.7318893  0.4637755
##   0.300      2                  1097     0.7317442  0.4634854
##   0.300      2                  1098     0.7318621  0.4637211
##   0.300      2                  1099     0.7319709  0.4639388
##   0.300      2                  1100     0.7319528  0.4639025
##   0.300      3                  1080     0.7265380  0.4530730
##   0.300      3                  1081     0.7267647  0.4535266
##   0.300      3                  1082     0.7267556  0.4535084
##   0.300      3                  1083     0.7265017  0.4530004
##   0.300      3                  1084     0.7264563  0.4529097
##   0.300      3                  1085     0.7264926  0.4529822
##   0.300      3                  1086     0.7265289  0.4530548
##   0.300      3                  1087     0.7266377  0.4532725
##   0.300      3                  1088     0.7267466  0.4534902
##   0.300      3                  1089     0.7267194  0.4534357
##   0.300      3                  1090     0.7267284  0.4534540
##   0.300      3                  1091     0.7267103  0.4534177
##   0.300      3                  1092     0.7267466  0.4534903
##   0.300      3                  1093     0.7267194  0.4534359
##   0.300      3                  1094     0.7267194  0.4534359
##   0.300      3                  1095     0.7267194  0.4534359
##   0.300      3                  1096     0.7267738  0.4535448
##   0.300      3                  1097     0.7268101  0.4536172
##   0.300      3                  1098     0.7268464  0.4536898
##   0.300      3                  1099     0.7268101  0.4536172
##   0.300      3                  1100     0.7269552  0.4539074
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 15
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 1095,
##  interaction.depth = 2, shrinkage = 0.2 and n.minobsinnode = 15.
```

```
predict_gbm_tune <- predict(model_gbm_tune, newdata = testing)
confusionMatrix(predict_gbm_tune, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4766 1817
##      True   1359 4307
##
##                Accuracy : 0.7407
##                  95% CI : (0.7329, 0.7485)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4814
##
##  Mcnemar's Test P-Value : 5.097e-16
##
##             Sensitivity : 0.7781
##             Specificity : 0.7033
##          Pos Pred Value : 0.7240
##          Neg Pred Value : 0.7601
##              Prevalence : 0.5000
##          Detection Rate : 0.3891
##    Detection Prevalence : 0.5374
##       Balanced Accuracy : 0.7407
##
##        'Positive' Class : False
##
```

ROC AND AUC An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. #Naive Bayes

```
naive_bayes.probs <- predict(Naive_Bayes_Model_tune,testing,type="raw")
head(naive_bayes.probs)
```

```
##            False       True
## [1,] 0.953809075 0.04619093
## [2,] 0.957517127 0.04248287
## [3,] 0.944799779 0.05520022
## [4,] 0.003508703 0.99649130
## [5,] 0.938687528 0.06131247
## [6,] 0.813027263 0.18697274
```

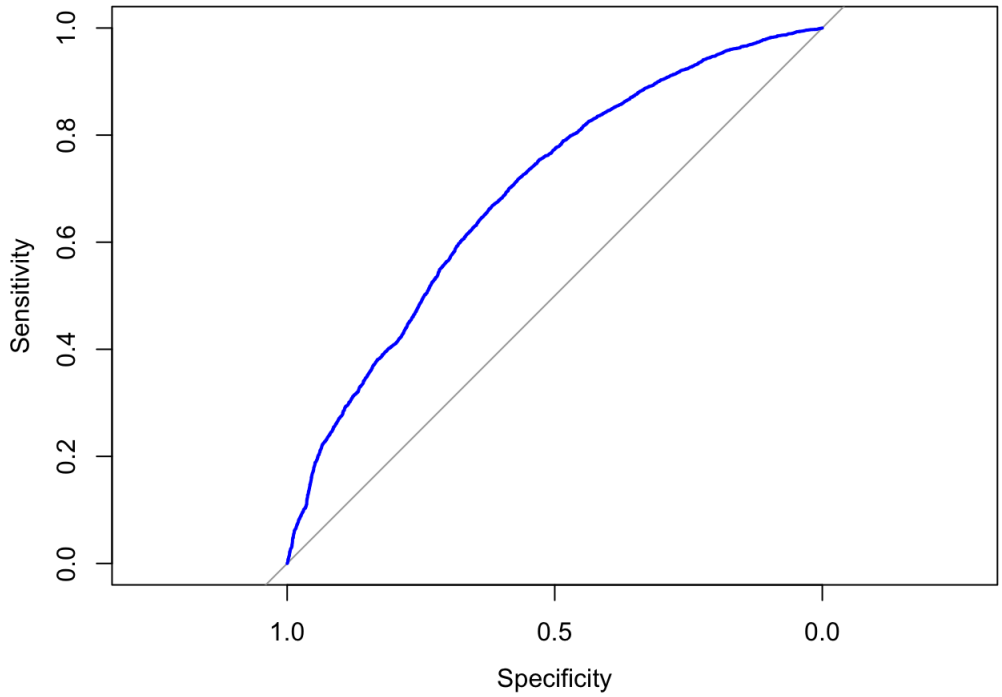
```
colnames(naive_bayes.probs)[1]="False"
colnames(naive_bayes.probs)[2]="True"

naive_bayes_roc_curve <- roc(testing$Disease,naive_bayes.probs[, "True"])
```

```
## Setting levels: control = False, case = True
```

```
## Setting direction: controls < cases
```

```
r1<-plot(naive_bayes_roc_curve,col="blue")
```



#knn

```
knn.probs <- predict(model_knn_tune,testing,type="prob")
head(knn.probs)
```

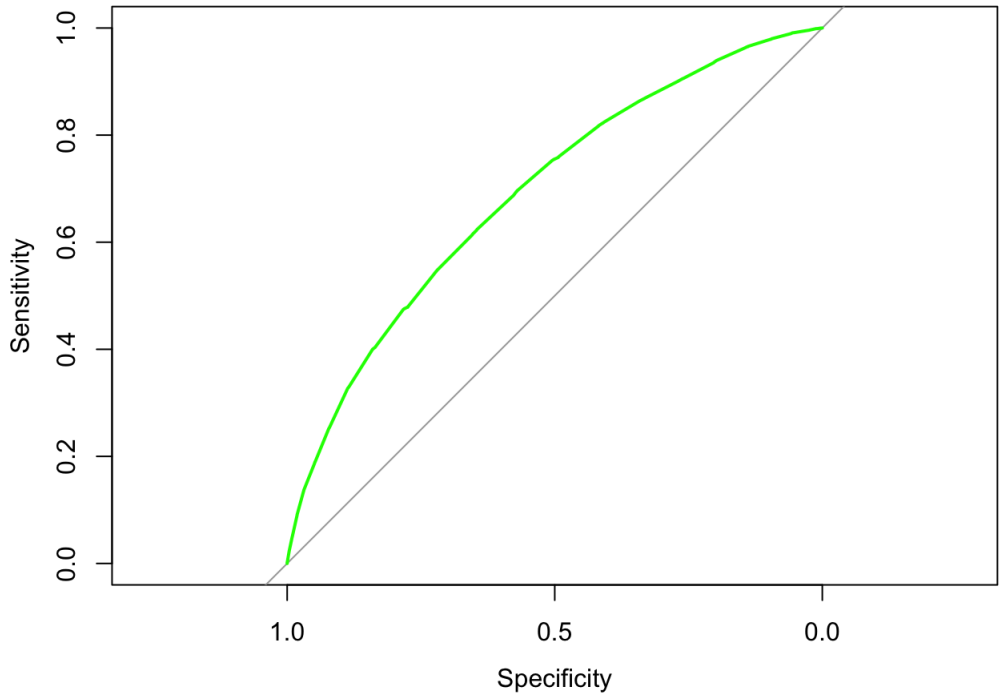
```
##   False True
## 1  0.72 0.28
## 2  0.76 0.24
## 3  0.50 0.50
## 4  0.12 0.88
## 5  0.52 0.48
## 6  0.52 0.48
```

```
colnames(knn.probs)[1]="False"
colnames(knn.probs)[2]="True"
knn_roc_curve <- roc(testing$Disease,knn.probs$True)
```

```
## Setting levels: control = False, case = True
```

```
## Setting direction: controls < cases
```

```
r2<-plot(knn_roc_curve,col="green")
```



#Random forest

```
rf.probs <- predict(model_rf_tune,testing,type="prob")
head(rf.probs)
```

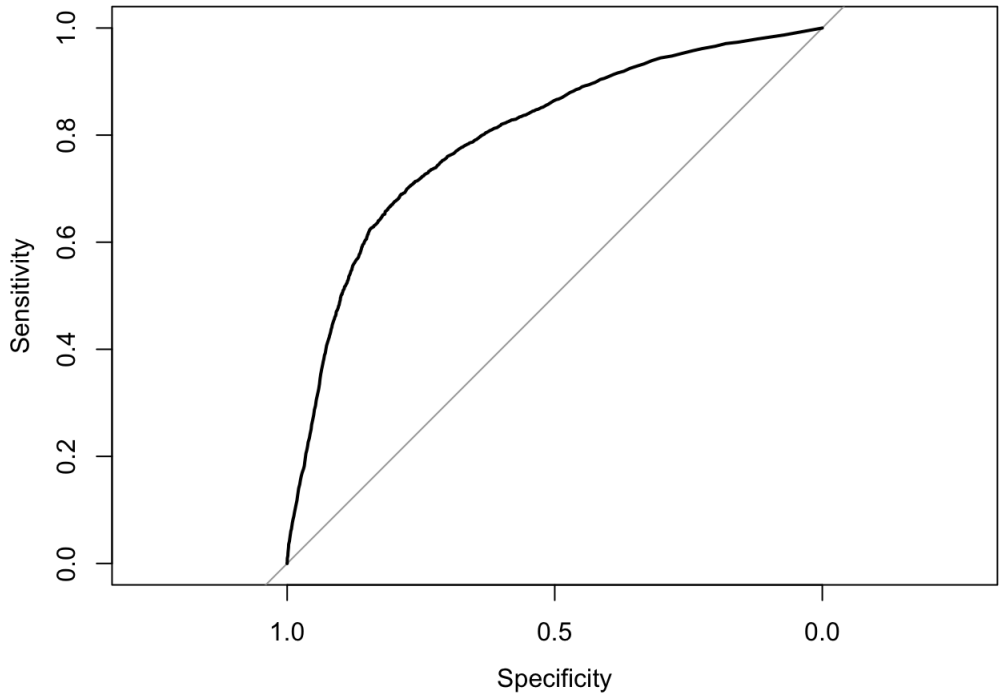
```
##    False  True
## 6  0.990 0.010
## 8  0.994 0.006
## 19 0.322 0.678
## 20 0.008 0.992
## 22 0.864 0.136
## 24 0.618 0.382
```

```
colnames(rf.probs)[1]="False"
colnames(rf.probs)[2]="True"
rf_roc_curve <- roc(testing$Disease,rf.probs$True)
```

```
## Setting levels: control = False, case = True
```

```
## Setting direction: controls < cases
```

```
r3<-plot(rf_roc_curve)
```



#Gradient Boosting

```
gbm.probs <- predict(model_gbm_tune,testing,type="prob")
head(gbm.probs)
```

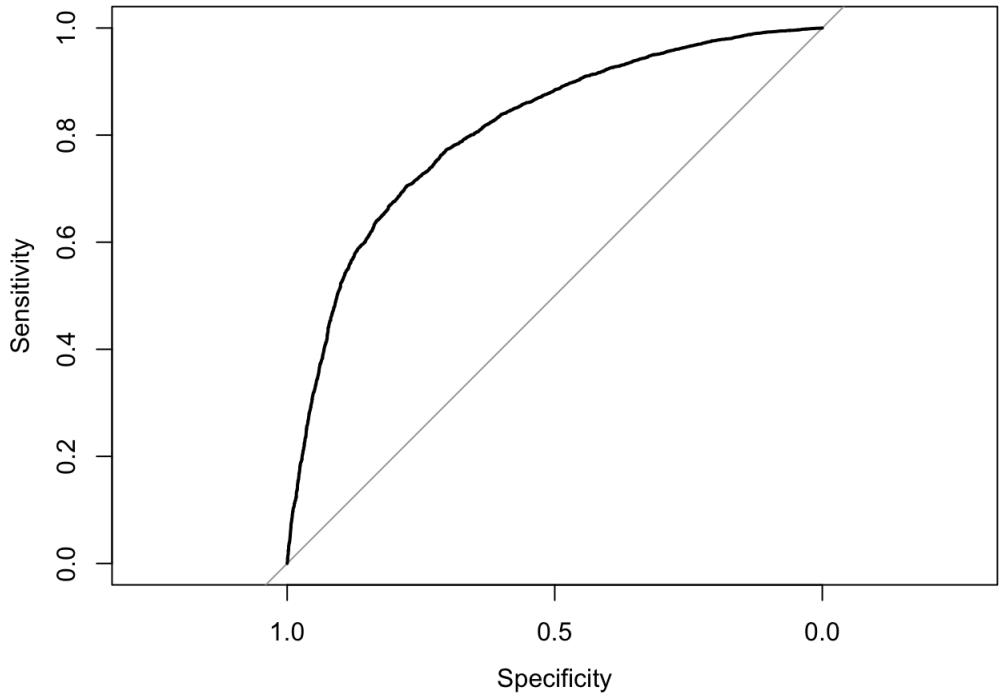
```
##       False      True
## 1 0.7371233 0.2628767
## 2 0.8483674 0.1516326
## 3 0.3681937 0.6318063
## 4 0.1323264 0.8676736
## 5 0.6457974 0.3542026
## 6 0.6089439 0.3910561
```

```
colnames(gbm.probs)[1]="False"
colnames(gbm.probs)[2]="True"
gbm_roc_curve <- roc(testing$Disease,gbm.probs$True)
```

```
## Setting levels: control = False, case = True
```

```
## Setting direction: controls < cases
```

```
r4<-plot(gbm_roc_curve)
```




```
#grid.arrange(r1,r2,r3,r4,nrow = 2, ncol = 2)
```

Area under the curve AUC provides an aggregate measure of performance across all possible classification thresholds. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

```
auc_naive_bayes<-auc(testing$Disease, naive_bayes.probs[, "True"])
```

```
## Setting levels: control = False, case = True
```

```
## Setting direction: controls < cases
```

```
auc_knn<-auc(testing$Disease, knn.probs$True)
```

```
## Setting levels: control = False, case = True
## Setting direction: controls < cases
```

```
auc_rf<-auc(testing$Disease, rf.probs$True)
```

```
## Setting levels: control = False, case = True
## Setting direction: controls < cases
```

```
auc_gbm<-auc(testing$Disease, gbm.probs$True)
```

```
## Setting levels: control = False, case = True
## Setting direction: controls < cases
```

```
Model <- c("NB","KNN","Random_Forest","Gradient_boosting")
AUC <- c(auc_naive_bayes,auc_knn,auc_rf,auc_gbm)

auc<-data.frame(Model,AUC)
auc
```

```
##               Model       AUC
## 1                NB 0.6925147
## 2               KNN 0.6908712
## 3     Random_Forest 0.7964372
## 4 Gradient_boosting 0.8089440
```

SECTION 3: TESTING DATA

1. Prediction & Interpretation on Test Data Reading the Testing data csv and storing it into a dataframe

```
#Loading Weather Forecasting training dataset.

d_test <- read.csv("/Users/juilee81/Desktop/DA/DA_HW_03/Disease\ Prediction\ Testing.csv",header = TRUE)
#View(d_test)
```

Checking for missing values column-wise and visualize the data:

```
colSums(is.na(d_test))
```

```
##                  ID                 Age              Gender
##                   0                   0                   0
##              Height              Weight High.Blood.Pressure
##                   0                   0                   0
##  Low.Blood.Pressure         Cholesterol             Glucose
##                   0                   0                   0
##               Smoke             Alcohol            Exercise
##                   0                   0                   0
```

Normalising the numeric data

```
normalize <- function(x) {
return ((x - min(x)) / (max(x) - min(x))) }
```

```
d_test$Age <- normalize(d_test$Age)
d_test$Height <- normalize(d_test$Height)
d_test$Weight <- normalize(d_test$Weight)
d_test$Low.Blood.Pressure <- normalize(d_test$Low.Blood.Pressure)
d_test$High.Blood.Pressure  <- normalize(d_test$High.Blood.Pressure)
```

Creating dummy variables of all the categorical variables

```
library(fastDummies)
ID <- d_test$ID
d_dummies_test <- fastDummies::dummy_cols(d_test,select_columns=c('Gender','Cholesterol','Glucose'))
d_dummies_test <- d_dummies_test[,c(-1,-3,-8,-9)]
colnames(d_dummies_test)[13] <- "Cholesterol_too_high"
colnames(d_dummies_test)[16] <- "Glucose_too_high"
```

Naive Bayes final model

```
predict_nb_final <- predict(Naive_Bayes_Model_tune, newdata = d_dummies_test)
```

KNN final model

```
predict_knn_final <- predict(model_knn_tune, newdata = d_dummies_test)
```

Random forest final model

```
predict_rf_final <- predict(model_rf_tune, newdata = d_dummies_test)
```

Gradient Boosting Model

```
predict_gbm_final <- predict(model_gbm_tune, newdata = d_dummies_test)
```

SVM-Linear final model

```
predict_svm_linear_final<-predict(model_svm_linear_tune, newdata = d_dummies_test)
```

Non linear final model

```
predict_svm_non_linear_final <- predict(model_svm_rbf_tune, newdata = d_dummies_test)
```

INTO CSV

```
Disease_predictions_csv <- data.frame(ID,predict_nb_final,predict_knn_final,predict_svm_linear_final,predict
_svm_non_linear_final,predict_rf_final,predict_gbm_final)
colnames(Disease_predictions_csv) <- c('ID','NB','KNN','SVM-Li','SVM-NLi','RF','GBM')
rownames(Disease_predictions_csv)<-NULL
```

Writing to CSV

```
write.csv(Disease_predictions_csv,"HW_03_Juilee_Salunkhe_Predictions.csv")
```

Conclusion In this study we have explored the data of unspecified disease dataset and gain insights about the key factors that decide the whether or not the person has the disease or not using multiple machine learning algorithms and data analysis.