Classificacao

Labdaps

Pacotes

```
#install.packages("devtools")
#devtools::install_github("laderast/cvdRiskData")
library(caret)
library(tidyverse)
library(data.table)
library(cvdRiskData)
library(pROC)
library(ROCR)
library(dummies)
library(corrplot)
```

Importar bancos de dados

```
#full patient data
data(cvd_patient)
cvd_patient_c <- cvd_patient

#Selected Data set
cvd_patient_r <- cvd_patient_c %>%
                 filter(numAge>55) %>%
                 select(-c(patientID,age,treat))

#variables:
#pacientID (numero de identificacao do paciente)
#age (idade categorizada)
#htn (hipertensão, Sim ou não)
#treat (tratamento para hipertensão, sim ou não)
#smoking (sim ou não)
#race (4 categorias: AmInd, Asian/PI, Black/AfAm, White)
#t2d (diabetes tipo 2: sim ou não)
#gender (M ou F)
#num Age (idade contínua)
#bmi (indice de massa corporea)
#tchol (colesterol)
#sbp(pressao arterial sistólica)
#variable response: cvd (doenca cardiovascular, sim ou não)
```

```
#criar dummies para variável "race"
race_dummies <- dummy.data.frame(select(cvd_patient_r,race), names=names(select(cvd_patient_r
,race)), sep="_")
names(race_dummies)
names(race_dummies)[2]<-"race_Asian_PI"
names(race_dummies)[3]<-"race_Black_AfAm"
banco_final<-cbind(cvd_patient_r,race_dummies)
banco_final_r<-select(banco_final,-c(race,race_White))
banco_final_r$race_AmInd<-as.factor(banco_final_r$race_AmInd)
banco_final_r$race_Asian_PI<-as.factor(banco_final_r$race_Asian_PI)
banco_final_r$race_Black_AfAm<-as.factor(banco_final_r$race_Black_AfAm)
```

Divisão do banco de dados completo em treinamento e teste

```
#stratified random sampling
set.seed(1)
split1 <- createDataPartition(banco_final_r$cvd, p=.85)[[1]]
trainData <- banco_final_r[split1,]
testData <- banco_final_r[-split1,]
prop.table(table(trainData$cvd))
```

```
##
##         N         Y
## 0.7189374 0.2810626
```

```
prop.table(table(testData$cvd))
```

```
##
##         N         Y
## 0.7189362 0.2810638
```

Treinamento de modelos preditivos

Técnica de reamostragem (para evitar sobreajuste)

```
#to obtain different performance measures (accuracy, Kappa, area under ROC curve, sensitivit
y, specificity)
fiveStats <- function(...)c(twoClassSummary(...),defaultSummary(...))

#control function
set.seed(1)
ctrl <- trainControl(method = "LGOCV", #Leave-group out cross-validation
                     number = 1, #number of folds or number of resampling iterations
                     p = 0.85, #training percentage
                     savePredictions = TRUE, # save "all" hold-out predictions for each resam
ple
                     classProbs = TRUE,# compute class probabilities for classification model
s
                     summaryFunction = fiveStats,
                     verboseIter = TRUE) #A logical for printing a training log
```

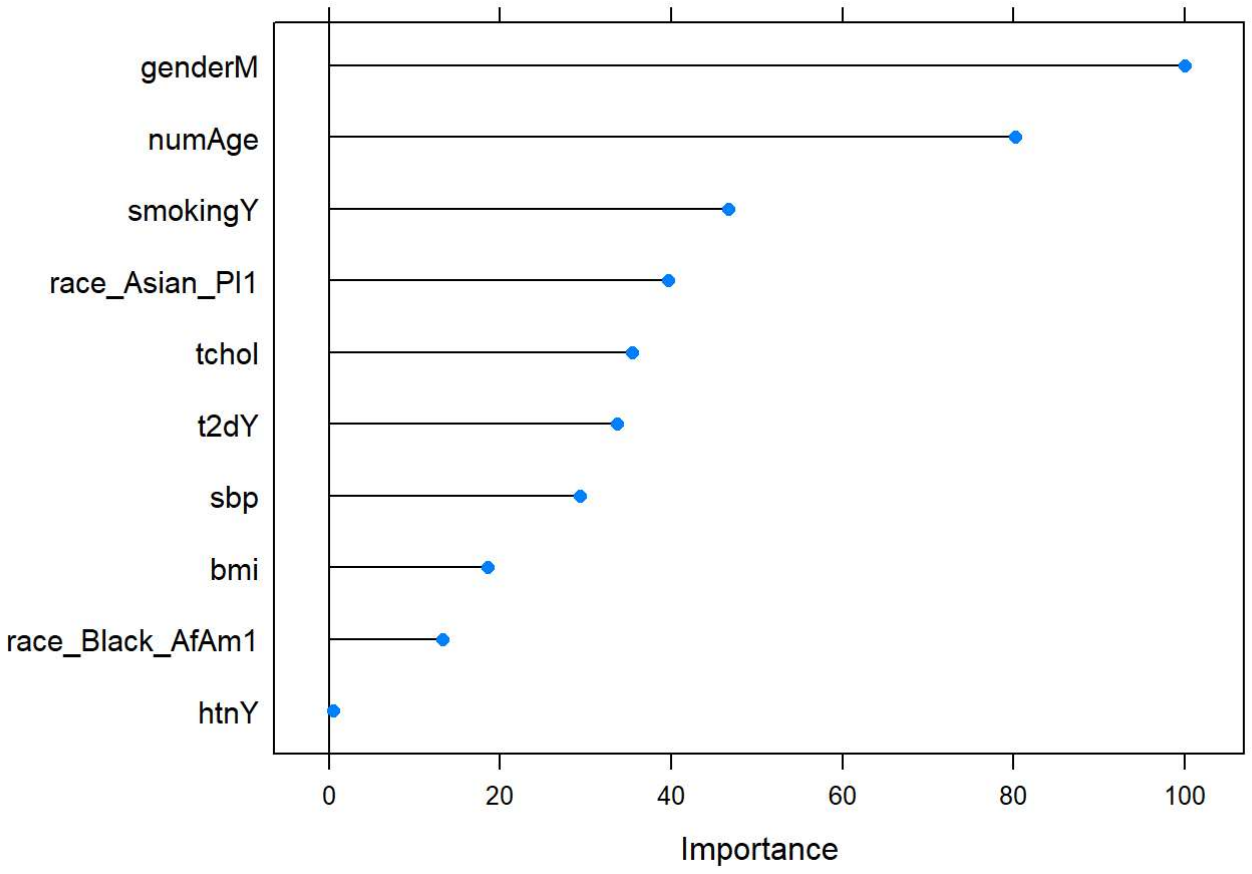
Treinamento de modelo inicial: regressão logística

```
set.seed(1)
reglogCVD <- train(cvd ~ ., data=trainData,
                   method="glm",
                   trControl=ctrl,
                   family="binomial",
                   metric="ROC")
```

```
reglogCVD
```

```
## Generalized Linear Model
##
## 106535 samples
##     11 predictor
##      2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (1 reps, 85%)
## Summary of sample sizes: 90556
## Resampling results:
##
##   ROC        Sens      Spec       Accuracy   Kappa
##   0.7723198  0.915477  0.3625028  0.7600601  0.3188927
```

```
varimp_RL<-varImp(reglogCVD)
plot(varimp_RL, top=10,scales=list(y=list(cex=.95)))
```



Predições do modelo inicial (RL)

```
#Aplique o modelo ajustado ao banco de teste

#Predict cvd on test data
classPredRL <- predict(reglogCVD, newdata=dplyr::select(testData,-cvd))

#calculate confusion Matrix and other measures of accuracy
confMatRL <- confusionMatrix(testData$cvd, classPredRL)
confMatRL
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     N     Y
##          N 12337  1179
##          Y  3334  1950
##
##                Accuracy : 0.7599
##                  95% CI : (0.7538, 0.766)
##     No Information Rate : 0.8336
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3218
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7873
##             Specificity : 0.6232
##          Pos Pred Value : 0.9128
##          Neg Pred Value : 0.3690
##              Prevalence : 0.8336
##          Detection Rate : 0.6562
##    Detection Prevalence : 0.7189
##       Balanced Accuracy : 0.7052
##
##        'Positive' Class : N
##
```

-Area under the ROC curve

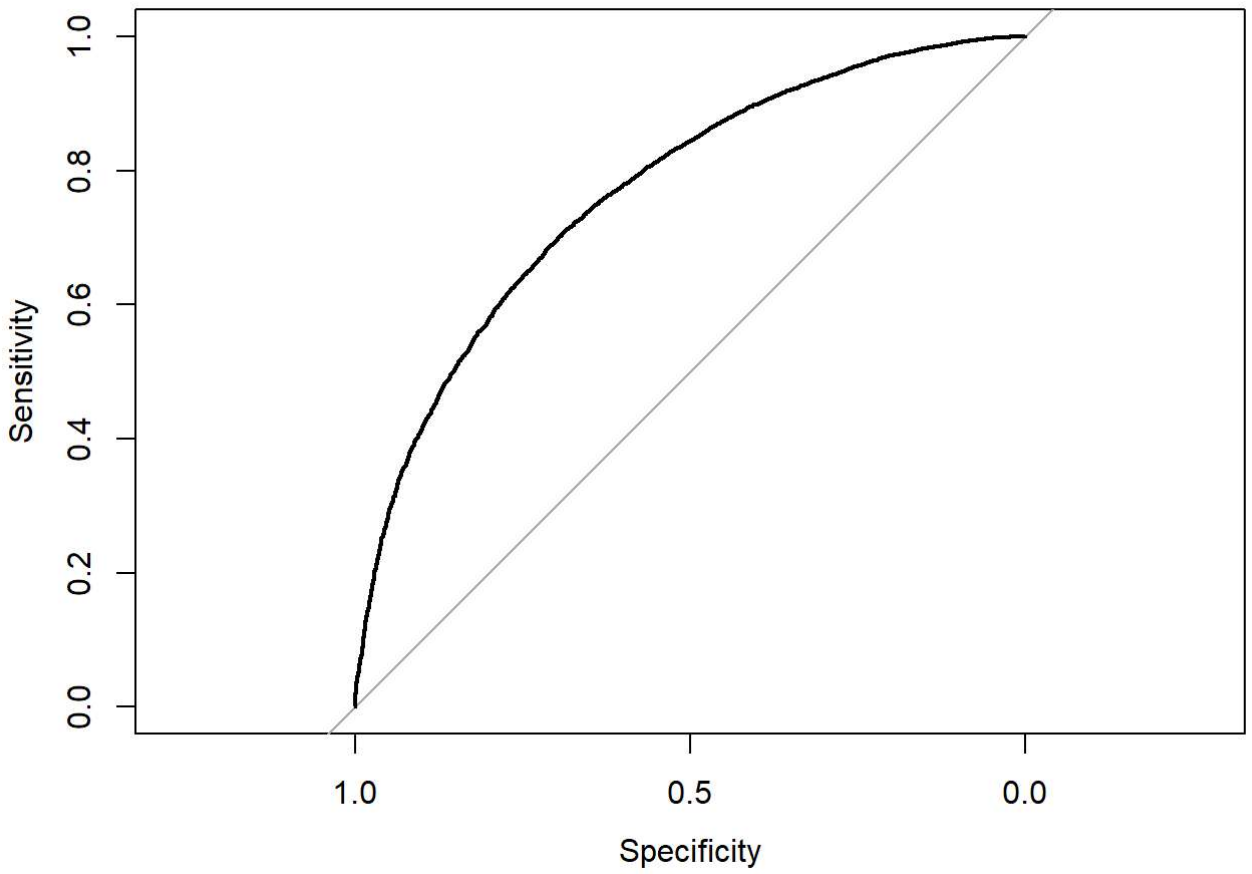
```
predicao_probRL <- predict(reglogCVD,dplyr::select(testData,-cvd),type="prob")
p_YesRL <- predicao_probRL[,"Y"]
rocCurveRL <- roc(response = testData$cvd,
                  predictor = p_YesRL,
                  levels = rev(levels(testData$cvd)))

AUC_rl<-pROC::auc(rocCurveRL)
AUC_rl
```

```
## Area under the curve: 0.7686
```

```
IC_AUC_rl<-pROC::ci.auc(rocCurveRL)
IC_AUC_rl
```

```
## 95% CI: 0.7613-0.776 (DeLong)
```

```
plot(rocCurveRL)
```



Modelos com hiperparâmetros

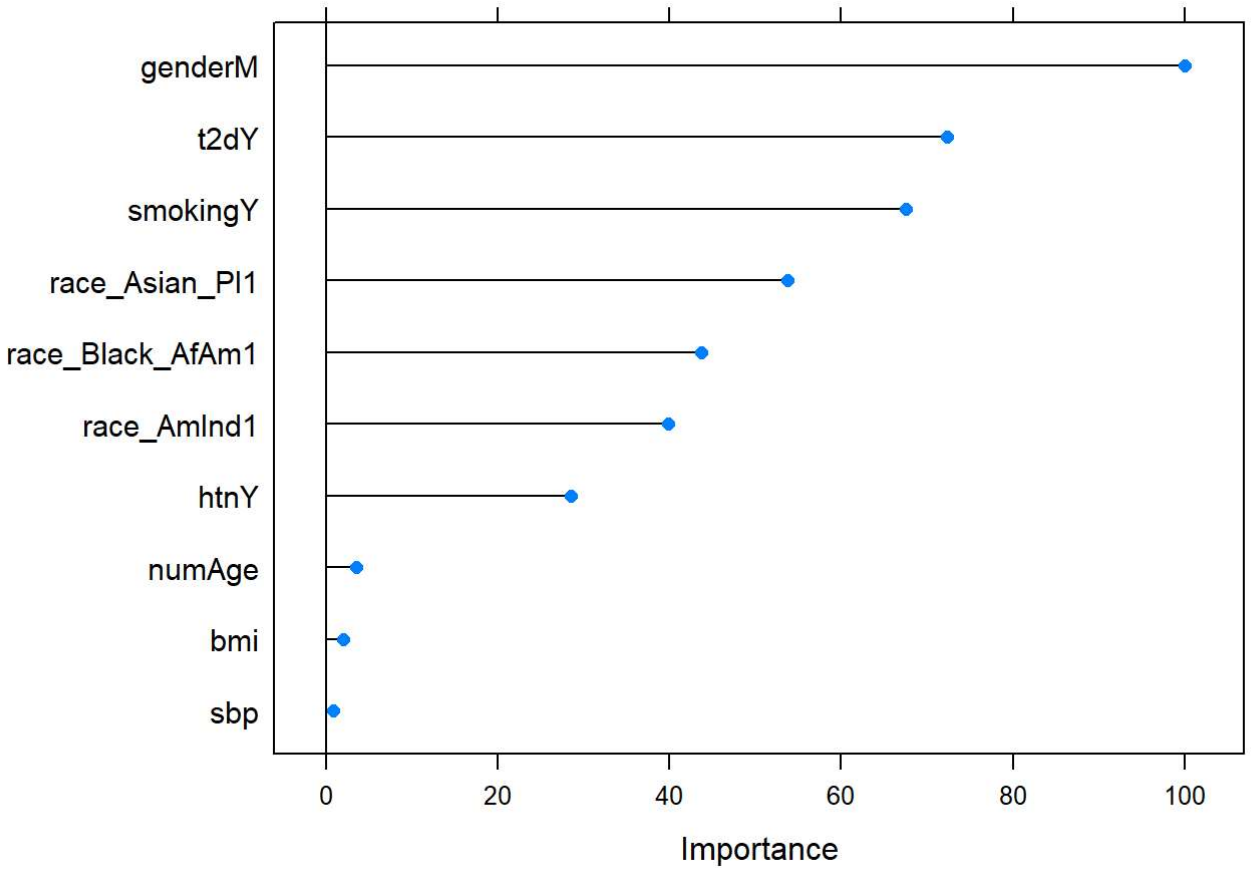
Regressão logística penalizada

```
set.seed(1)
glmnetCVD <- train(cvd ~ ., data=trainData,
                   method="glmnet",
                   trControl=ctrl,
                   metric="ROC")
```

```
glmnetCVD
```

```
## glmnet
##
## 106535 samples
##     11 predictor
##      2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (1 reps, 85%)
## Summary of sample sizes: 90556
## Resampling results across tuning parameters:
##
##   alpha  lambda       ROC        Sens       Spec       Accuracy
##   0.10   0.000256807  0.7723229  0.9166957  0.3598308  0.7601852
##   0.10   0.002568070  0.7723122  0.9173050  0.3589401  0.7603730
##   0.10   0.025680701  0.7716775  0.9345404  0.3126253  0.7597472
##   0.55   0.000256807  0.7722838  0.9164345  0.3598308  0.7599975
##   0.55   0.002568070  0.7722206  0.9185237  0.3544868  0.7599975
##   0.55   0.025680701  0.7689392  0.9504701  0.2683144  0.7587459
##   1.00   0.000256807  0.7722547  0.9164345  0.3593854  0.7598723
##   1.00   0.002568070  0.7720685  0.9201776  0.3489201  0.7596220
##   1.00   0.025680701  0.7627734  0.9661386  0.2106435  0.7538019
##   Kappa
##   0.3178683
##   0.3178174
##   0.2936633
##   0.3174939
##   0.3149313
##   0.2681971
##   0.3170315
##   0.3114898
##   0.2249161
##
## ROC was used to select the optimal model using  the largest value.
## The final values used for the model were alpha = 0.1 and lambda
##  = 0.000256807.
```

```
varimp_glmnet<-varImp(glmnetCVD)
plot(varimp_glmnet, top=10,scales=list(y=list(cex=.95)))
```




```
#alpha: the elastic-net mixing parameter
#alpha=1: lasso
#alpha=0: ridge
```

Predições GLMNET

```
#Aplique o modelo ajustado ao banco de teste

#Predict cvd on test data
classPredGLMNET <- predict(glmnetCVD, newdata=dplyr::select(testData,-cvd))

#calculate confusion Matrix and other measures of accuracy
confMatGLMNET <- confusionMatrix(testData$cvd, classPredGLMNET)
confMatGLMNET
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     N     Y
##          N 12363  1153
##          Y  3352  1932
##
##                Accuracy : 0.7604
##                  95% CI : (0.7542, 0.7665)
##     No Information Rate : 0.8359
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.321
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7867
##             Specificity : 0.6263
##          Pos Pred Value : 0.9147
##          Neg Pred Value : 0.3656
##              Prevalence : 0.8359
##          Detection Rate : 0.6576
##    Detection Prevalence : 0.7189
##       Balanced Accuracy : 0.7065
##
##        'Positive' Class : N
##
```

-Area under the ROC curve

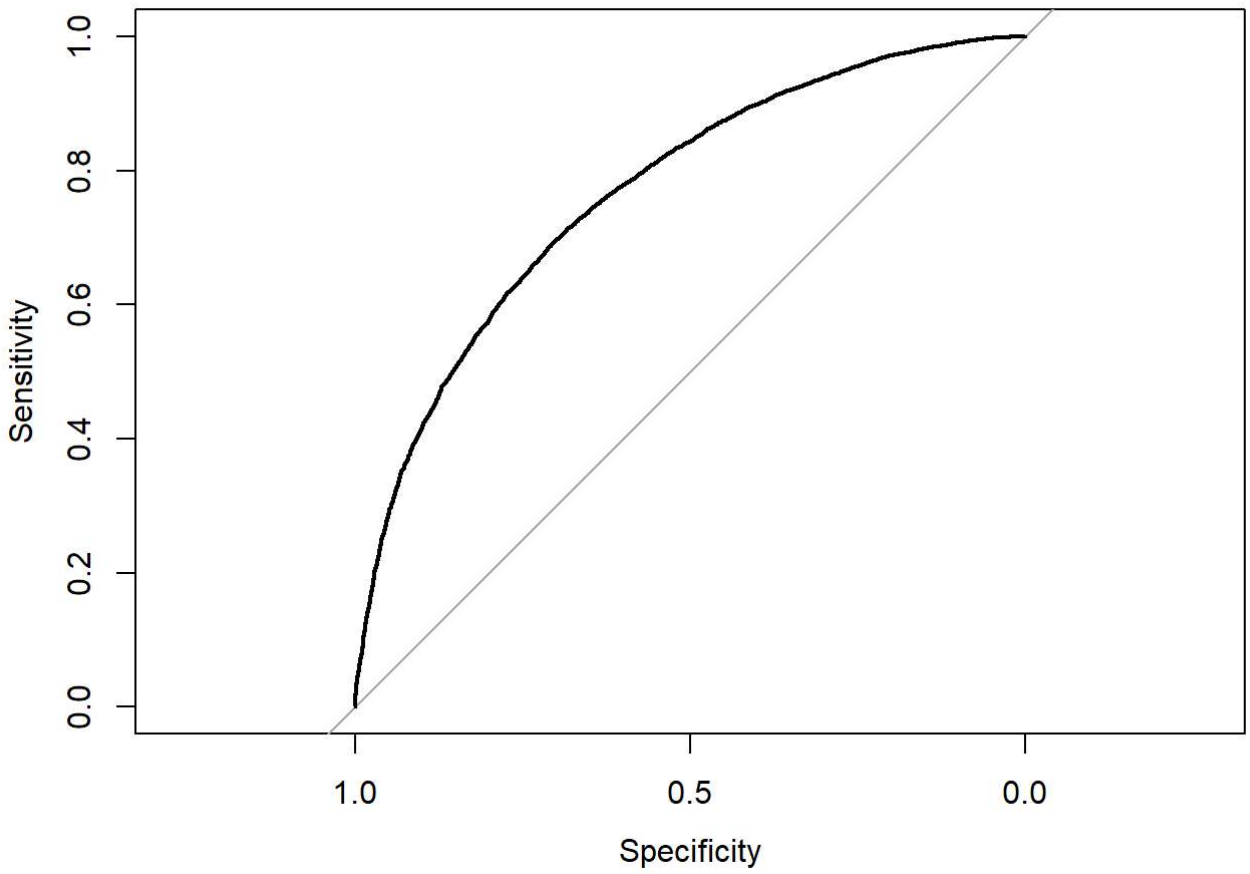
```
predicao_probGLMNET <- predict(glmnetCVD,dplyr::select(testData,-cvd),type="prob")
p_YesGLMNET <- predicao_probGLMNET[,"Y"]
rocCurveGLMNET <- roc(response = testData$cvd,
                      predictor = p_YesGLMNET,
                      levels = rev(levels(testData$cvd)))

AUC_glmnet<-pROC::auc(rocCurveGLMNET)
AUC_glmnet
```

```
## Area under the curve: 0.7687
```

```
IC_AUC_glmnet<-pROC::ci.auc(rocCurveGLMNET)
IC_AUC_glmnet
```

```
## 95% CI: 0.7613-0.776 (DeLong)
```

```
plot(rocCurveGLMNET)
```



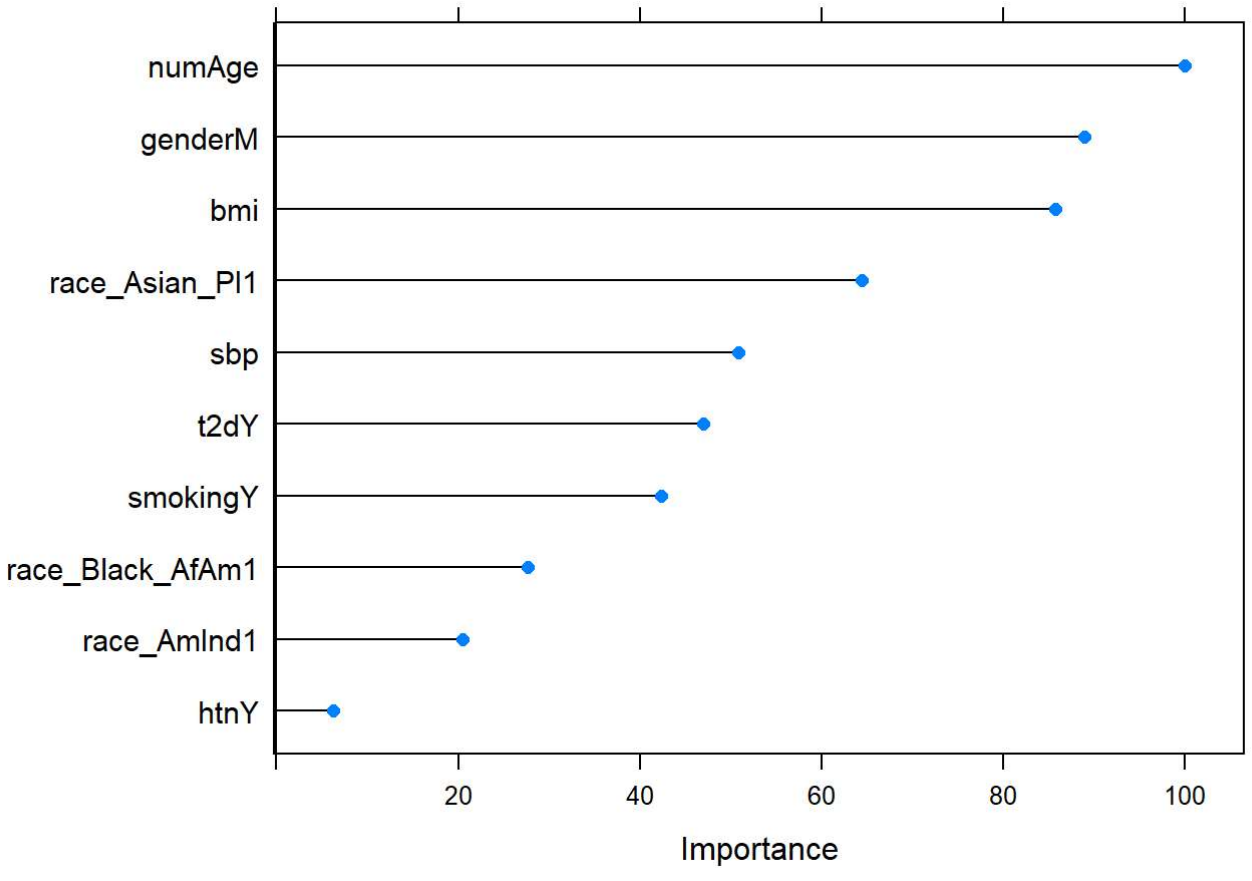
Redes neurais

```
set.seed(1)
nnetCVD <- train(cvd ~ ., data=trainData,
                 method="nnet",
                 trControl=ctrl,
                 metric="ROC")
```

```
nnetCVD
```

```
## Neural Network
##
## 106535 samples
##     11 predictor
##      2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (1 reps, 85%)
## Summary of sample sizes: 90556
## Resampling results across tuning parameters:
##
##   size  decay  ROC        Sens       Spec       Accuracy   Kappa
##   1     0e+00  0.5000000  1.0000000  0.0000000  0.7189436  0.0000000
##   1     1e-04  0.5000000  1.0000000  0.0000000  0.7189436  0.0000000
##   1     1e-01  0.5000000  1.0000000  0.0000000  0.7189436  0.0000000
##   3     0e+00  0.7437116  0.8474930  0.4386551  0.7325865  0.3018905
##   3     1e-04  0.7563809  1.0000000  0.0000000  0.7189436  0.0000000
##   3     1e-01  0.7731573  0.9304492  0.3248720  0.7602478  0.3008804
##   5     0e+00  0.5000000  1.0000000  0.0000000  0.7189436  0.0000000
##   5     1e-04  0.5000000  1.0000000  0.0000000  0.7189436  0.0000000
##   5     1e-01  0.7721137  0.9412430  0.2968159  0.7601227  0.2862566
##
## ROC was used to select the optimal model using  the largest value.
## The final values used for the model were size = 3 and decay = 0.1.
```

```
varimp_nnet<-varImp(nnetCVD)
plot(varimp_nnet, top=10,scales=list(y=list(cex=.95)))
```



Predições Redes neurais

```
#Aplique o modelo ajustado ao banco de teste

#Predict cvd on test data
classPredNNET <- predict(nnetCVD, newdata=dplyr::select(testData,-cvd))

#calculate confusion Matrix and other measures of accuracy
confMatNNET <- confusionMatrix(testData$cvd, classPredNNET)
confMatNNET
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     N     Y
##          N 12184  1332
##          Y  3199  2085
##
##                Accuracy : 0.759
##                  95% CI : (0.7528, 0.7651)
##     No Information Rate : 0.8182
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3317
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7920
##             Specificity : 0.6102
##          Pos Pred Value : 0.9015
##          Neg Pred Value : 0.3946
##              Prevalence : 0.8182
##          Detection Rate : 0.6481
##    Detection Prevalence : 0.7189
##       Balanced Accuracy : 0.7011
##
##        'Positive' Class : N
##
```

-Area under the ROC curve

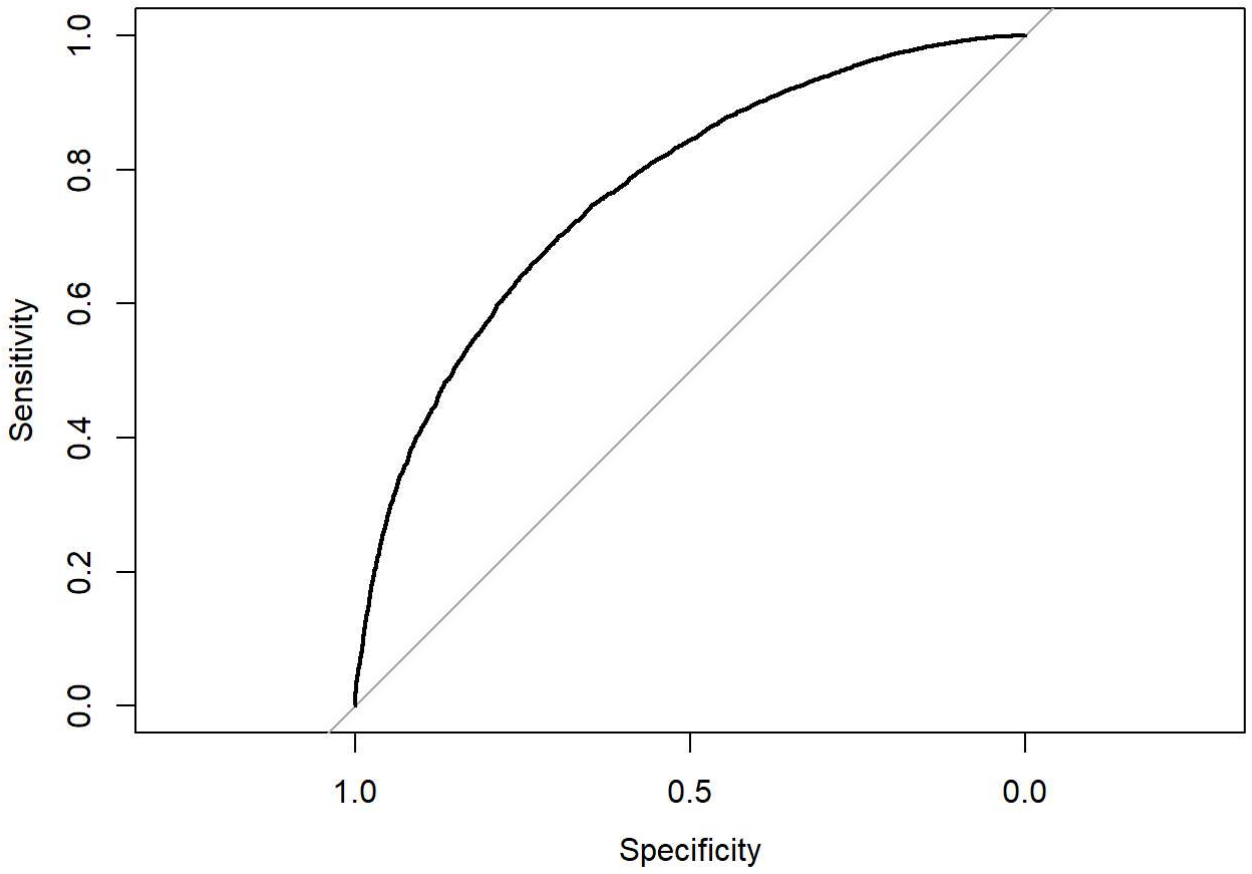
```
predicao_probNNET <- predict(nnetCVD,dplyr::select(testData,-cvd),type="prob")
p_YesNNET <- predicao_probNNET[,"Y"]
rocCurveNNET <- roc(response = testData$cvd,
                    predictor = p_YesNNET,
                    levels = rev(levels(testData$cvd)))

AUC_nnet<-pROC::auc(rocCurveNNET)
AUC_nnet
```

```
## Area under the curve: 0.7685
```

```
IC_AUC_nnet<-pROC::ci.auc(rocCurveNNET)
IC_AUC_nnet
```

```
## 95% CI: 0.7611-0.7758 (DeLong)
```

```
plot(rocCurveNNET)
```



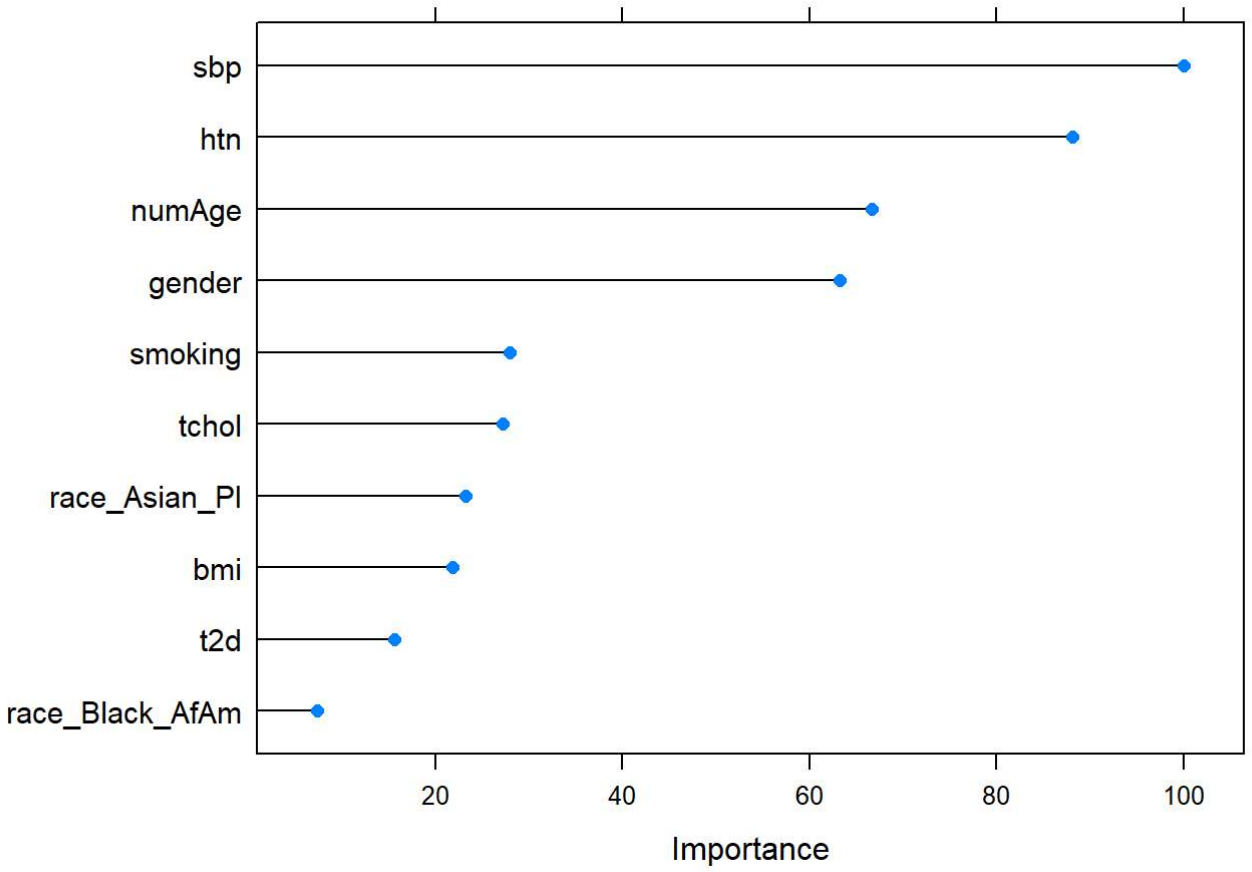
SVM

```
svmCVD <- train(cvd ~ ., data=trainData,
                method="svmLinear",
                trControl=ctrl,
                metric="ROC")
```

```
svmCVD
```

```
## Support Vector Machines with Linear Kernel
##
## 106535 samples
##     11 predictor
##      2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (1 reps, 85%)
## Summary of sample sizes: 90556
## Resampling results:
##
##   ROC        Sens       Spec       Accuracy   Kappa
##   0.7725491  0.9214833  0.3524827  0.7615621  0.3171002
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
varimp_svm<-varImp(svmCVD)
plot(varimp_svm, top=10,scales=list(y=list(cex=.95)))
```



Predicoes SVM

```
#Aplique o modelo ajustado ao banco de teste

#Predict cvd on test data
classPredSVM <- predict(svmCVD, newdata=select(testData,-cvd))

#calculate confusion Matrix and other measures of accuracy
confMatSVM <- confusionMatrix(testData$cvd, classPredSVM)
confMatSVM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     N     Y
##          N 12394  1122
##          Y  3373  1911
##
##                Accuracy : 0.7609
##                  95% CI : (0.7547, 0.767)
##     No Information Rate : 0.8387
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3202
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7861
##             Specificity : 0.6301
##          Pos Pred Value : 0.9170
##          Neg Pred Value : 0.3617
##              Prevalence : 0.8387
##          Detection Rate : 0.6593
##    Detection Prevalence : 0.7189
##       Balanced Accuracy : 0.7081
##
##        'Positive' Class : N
##
```

-Area under the ROC curve

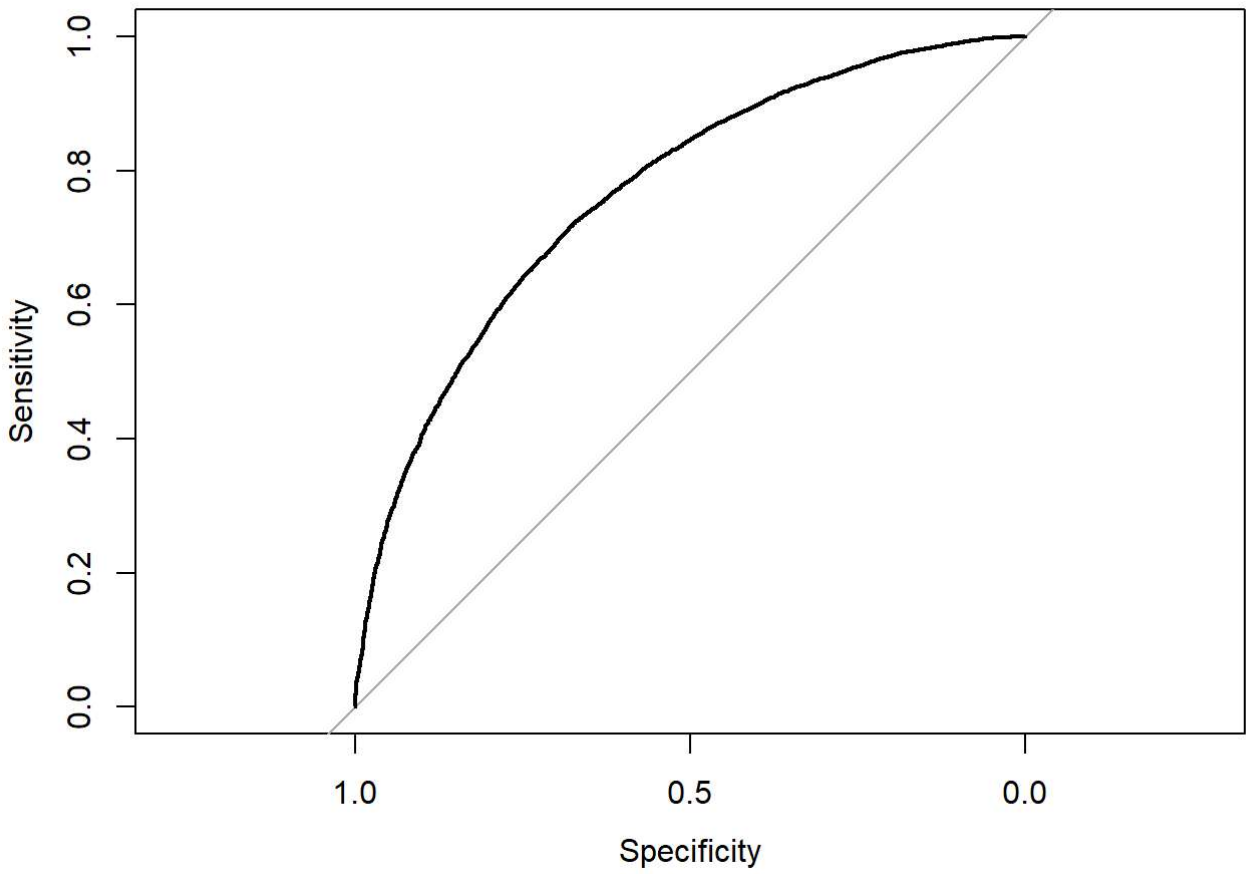
```
predicao_probSVM <- predict(svmCVD,select(testData,-cvd),type="prob")
p_YesSVM <- predicao_probSVM[,"Y"]
rocCurveSVM <- roc(response = testData$cvd,
                   predictor = p_YesSVM,
                   levels = rev(levels(testData$cvd)))

AUC_svm<-pROC::auc(rocCurveSVM)
AUC_svm
```

```
## Area under the curve: 0.7666
```

```
IC_AUC_svm<-pROC::ci.auc(rocCurveSVM)
IC_AUC_svm
```

```
## 95% CI: 0.7592-0.774 (DeLong)
```

```
plot(rocCurveSVM)
```



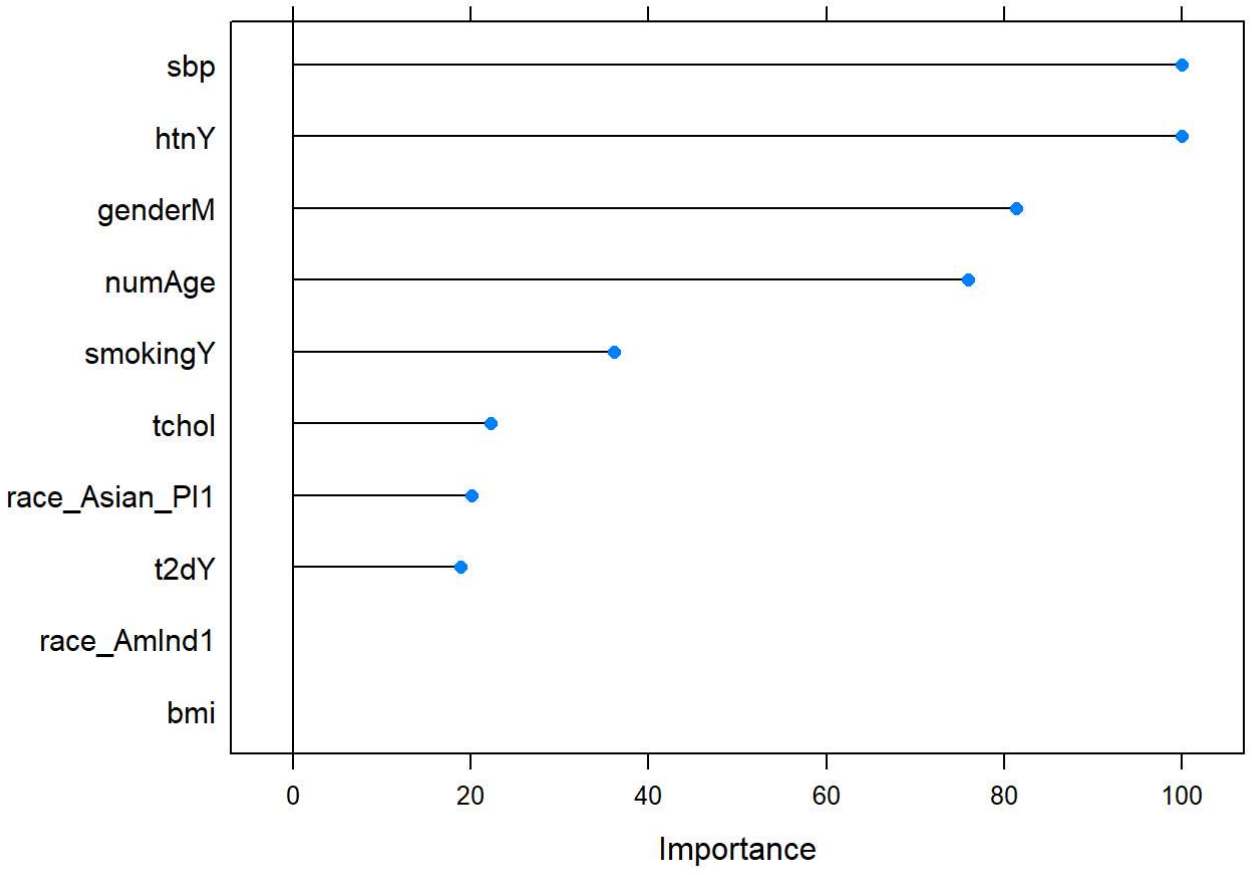
Árvore de classificação

```
set.seed(1)
rpartCVD <- train(cvd ~ ., data=trainData,
                  method="rpart",
                  trControl=ctrl,
                  metric="ROC")
```

```
rpartCVD
```

```
## CART
##
## 106535 samples
##     11 predictor
##      2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (1 reps, 85%)
## Summary of sample sizes: 90556
## Resampling results across tuning parameters:
##
##   cp           ROC        Sens       Spec       Accuracy   Kappa
##   0.005677454  0.6872049  0.9530815  0.2286796  0.7494837  0.2267321
##   0.021106770  0.6829124  0.9733635  0.1438432  0.7402215  0.1542153
##   0.026060626  0.6829124  0.9733635  0.1438432  0.7402215  0.1542153
##
## ROC was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.005677454.
```

```
varimp_rpart<-varImp(rpartCVD)
plot(varimp_rpart, top=10,scales=list(y=list(cex=.95)))
```



Predições Árvore de classificação

```
#Aplique o modelo ajustado ao banco de teste

#Predict cvd on test data
classPredRPART <- predict(rpartCVD, newdata=dplyr::select(testData,-cvd))

#calculate confusion Matrix and other measures of accuracy
confMatRPART <- confusionMatrix(testData$cvd, classPredRPART)
confMatRPART
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     N     Y
##          N 13005   511
##          Y  4239  1045
##
##                Accuracy : 0.7473
##                  95% CI : (0.7411, 0.7535)
##     No Information Rate : 0.9172
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2037
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7542
##             Specificity : 0.6716
##          Pos Pred Value : 0.9622
##          Neg Pred Value : 0.1978
##              Prevalence : 0.9172
##          Detection Rate : 0.6918
##    Detection Prevalence : 0.7189
##       Balanced Accuracy : 0.7129
##
##        'Positive' Class : N
##
```

-Area under the ROC curve

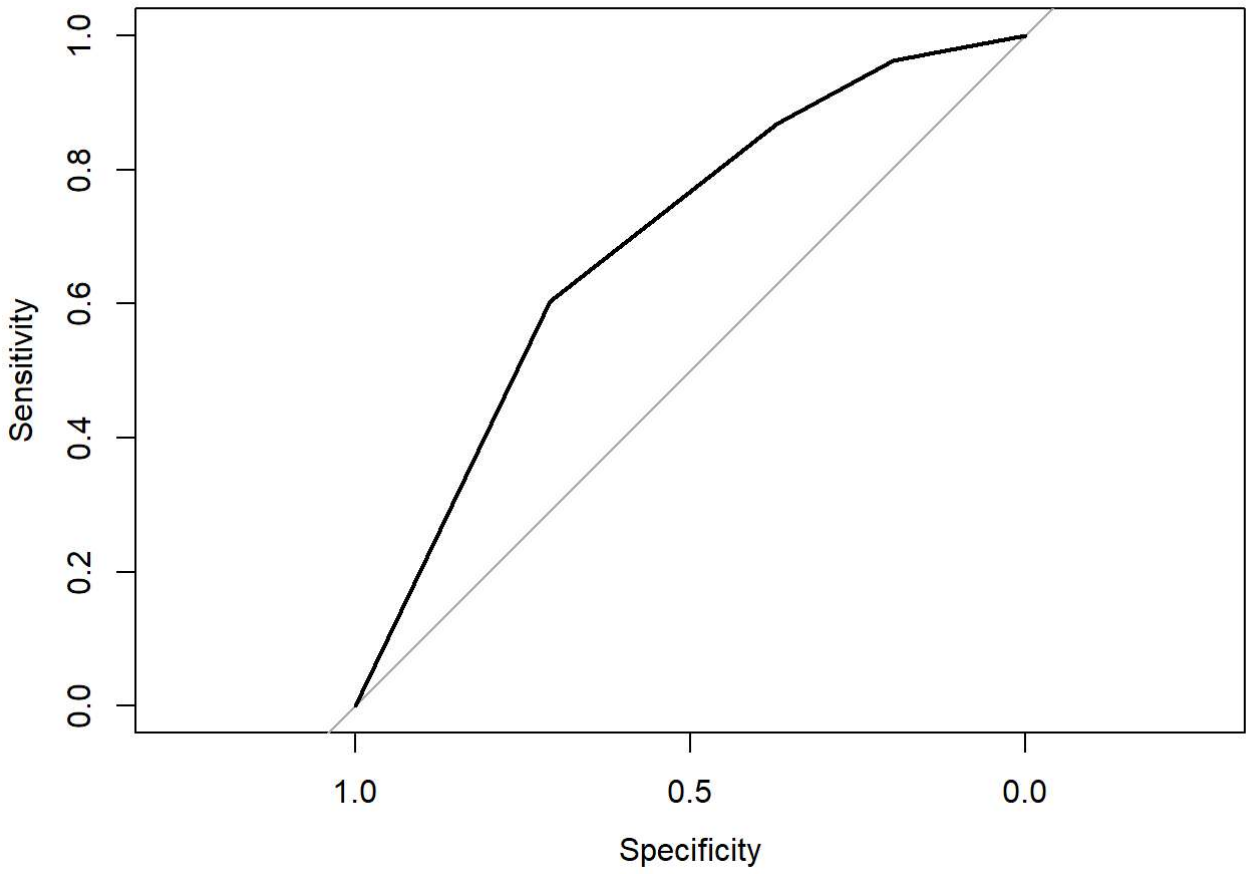
```
predicao_probRPART <- predict(rpartCVD,dplyr::select(testData,-cvd),type="prob")
p_YesRPART <- predicao_probRPART[,"Y"]
rocCurveRPART <- roc(response = testData$cvd,
                     predictor = p_YesRPART,
                     levels = rev(levels(testData$cvd)))

AUC_rpart<-pROC::auc(rocCurveRPART)
AUC_rpart
```

```
## Area under the curve: 0.689
```

```
IC_AUC_rpart<-pROC::ci.auc(rocCurveRPART)
IC_AUC_rpart
```

```
## 95% CI: 0.6809-0.6971 (DeLong)
```

```
plot(rocCurveRPART)
```



Boosting

```
set.seed(1)
xgbCVD <- train(cvd ~ ., data=trainData,
                method="xgbTree",
                trControl=ctrl,
                metric="ROC")
```

```
xgbCVD
```

```
## eXtreme Gradient Boosting
##
## 106535 samples
##     11 predictor
##      2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (1 reps, 85%)
## Summary of sample sizes: 90556
## Resampling results across tuning parameters:
##
##   eta  max_depth  colsample_bytree  subsample  nrounds  ROC
##   0.3  1          0.6               0.50        50      0.7724637
##   0.3  1          0.6               0.50       100      0.7745678
##   0.3  1          0.6               0.50       150      0.7747005
##   0.3  1          0.6               0.75        50      0.7717635
##   0.3  1          0.6               0.75       100      0.7744404
##   0.3  1          0.6               0.75       150      0.7747823
##   0.3  1          0.6               1.00        50      0.7719233
##   0.3  1          0.6               1.00       100      0.7743852
##   0.3  1          0.6               1.00       150      0.7746723
##   0.3  1          0.8               0.50        50      0.7717859
##   0.3  1          0.8               0.50       100      0.7741831
##   0.3  1          0.8               0.50       150      0.7742506
##   0.3  1          0.8               0.75        50      0.7726929
##   0.3  1          0.8               0.75       100      0.7745596
##   0.3  1          0.8               0.75       150      0.7748806
##   0.3  1          0.8               1.00        50      0.7721068
##   0.3  1          0.8               1.00       100      0.7744185

##   0.3  1          0.8               1.00       150      0.7745842
##   0.3  2          0.6               0.50        50      0.7756607
##   0.3  2          0.6               0.50       100      0.7760181
##   0.3  2          0.6               0.50       150      0.7761534
##   0.3  2          0.6               0.75        50      0.7762000
##   0.3  2          0.6               0.75       100      0.7766464
##   0.3  2          0.6               0.75       150      0.7766751
##   0.3  2          0.6               1.00        50      0.7757741
##   0.3  2          0.6               1.00       100      0.7762081
##   0.3  2          0.6               1.00       150      0.7762243
##   0.3  2          0.8               0.50        50      0.7764895
##   0.3  2          0.8               0.50       100      0.7766890
##   0.3  2          0.8               0.50       150      0.7767390
##   0.3  2          0.8               0.75        50      0.7758784
##   0.3  2          0.8               0.75       100      0.7762149
##   0.3  2          0.8               0.75       150      0.7761541
##   0.3  2          0.8               1.00        50      0.7757347
##   0.3  2          0.8               1.00       100      0.7763252
##   0.3  2          0.8               1.00       150      0.7763098
##   0.3  3          0.6               0.50        50      0.7767273
##   0.3  3          0.6               0.50       100      0.7763260
##   0.3  3          0.6               0.50       150      0.7758665
##   0.3  3          0.6               0.75        50      0.7771896
##   0.3  3          0.6               0.75       100      0.7766912
##   0.3  3          0.6               0.75       150      0.7762459
##   0.3  3          0.6               1.00        50      0.7766581
##   0.3  3          0.6               1.00       100      0.7765422
##   0.3  3          0.6               1.00       150      0.7764816
```

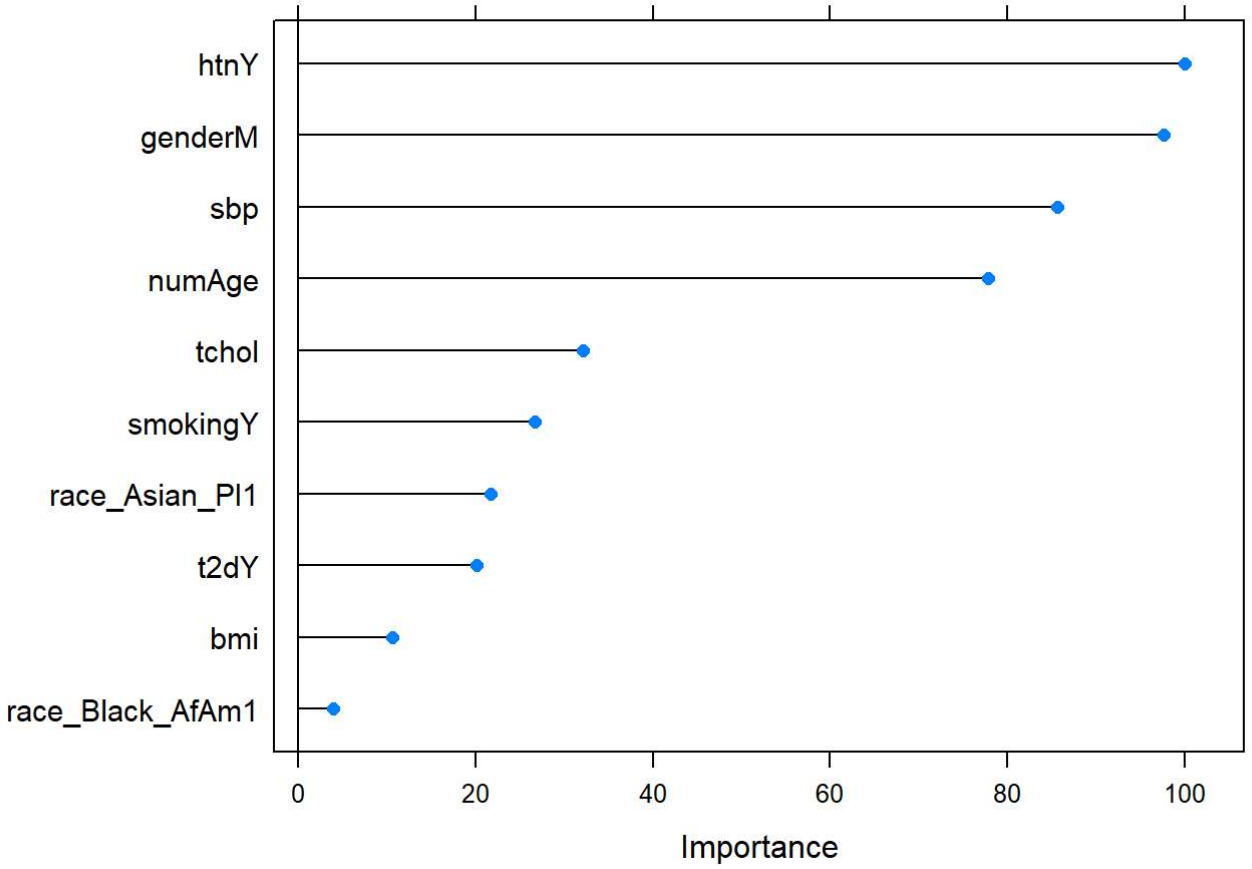
```
##   0.3  3          0.8          0.50         50       0.7760850
##   0.3  3          0.8          0.50        100       0.7754196
##   0.3  3          0.8          0.50        150       0.7753324
##   0.3  3          0.8          0.75         50       0.7761460
##   0.3  3          0.8          0.75        100       0.7762776
##   0.3  3          0.8          0.75        150       0.7755181
##   0.3  3          0.8          1.00         50       0.7764517
##   0.3  3          0.8          1.00        100       0.7764305
##   0.3  3          0.8          1.00        150       0.7762154
##   0.4  1          0.6          0.50         50       0.7728144
##   0.4  1          0.6          0.50        100       0.7738640
##   0.4  1          0.6          0.50        150       0.7741875
##   0.4  1          0.6          0.75         50       0.7734288
##   0.4  1          0.6          0.75        100       0.7742756
##   0.4  1          0.6          0.75        150       0.7743911
##   0.4  1          0.6          1.00         50       0.7727938
##   0.4  1          0.6          1.00        100       0.7743827
##   0.4  1          0.6          1.00        150       0.7745119
##   0.4  1          0.8          0.50         50       0.7731779
##   0.4  1          0.8          0.50        100       0.7741418
##   0.4  1          0.8          0.50        150       0.7743136
##   0.4  1          0.8          0.75         50       0.7728009
##   0.4  1          0.8          0.75        100       0.7743003
##   0.4  1          0.8          0.75        150       0.7743136
##   0.4  1          0.8          1.00         50       0.7735710
##   0.4  1          0.8          1.00        100       0.7746024
##   0.4  1          0.8          1.00        150       0.7747367
##   0.4  2          0.6          0.50         50       0.7761899
##   0.4  2          0.6          0.50        100       0.7764127
##   0.4  2          0.6          0.50        150       0.7759243
##   0.4  2          0.6          0.75         50       0.7764582
##   0.4  2          0.6          0.75        100       0.7763643
##   0.4  2          0.6          0.75        150       0.7761943
##   0.4  2          0.6          1.00         50       0.7762100
##   0.4  2          0.6          1.00        100       0.7760164
##   0.4  2          0.6          1.00        150       0.7759979
##   0.4  2          0.8          0.50         50       0.7749085
##   0.4  2          0.8          0.50        100       0.7753909
##   0.4  2          0.8          0.50        150       0.7757186
##   0.4  2          0.8          0.75         50       0.7762920
##   0.4  2          0.8          0.75        100       0.7759812
##   0.4  2          0.8          0.75        150       0.7757514
##   0.4  2          0.8          1.00         50       0.7761056
##   0.4  2          0.8          1.00        100       0.7760292
##   0.4  2          0.8          1.00        150       0.7759028
##   0.4  3          0.6          0.50         50       0.7757973
##   0.4  3          0.6          0.50        100       0.7749590
##   0.4  3          0.6          0.50        150       0.7743182
##   0.4  3          0.6          0.75         50       0.7758495
##   0.4  3          0.6          0.75        100       0.7754433
##   0.4  3          0.6          0.75        150       0.7748412
##   0.4  3          0.6          1.00         50       0.7766137
##   0.4  3          0.6          1.00        100       0.7761221
##   0.4  3          0.6          1.00        150       0.7754596
##   0.4  3          0.8          0.50         50       0.7747467
##   0.4  3          0.8          0.50        100       0.7735055
##   0.4  3          0.8          0.50        150       0.7718986
##   0.4  3          0.8          0.75         50       0.7759760
```

```
##   0.4  3          0.8                0.75       50      0.7799760
##   0.4  3          0.8                0.75       100     0.7756361
##   0.4  3          0.8                0.75       150     0.7745679
##   0.4  3          0.8                1.00        50     0.7762684
##   0.4  3          0.8                1.00       100     0.7753796
##   0.4  3          0.8                1.00       150     0.7756188
##   Sens       Spec       Accuracy   Kappa
##   0.9275766  0.3286573  0.7592465  0.3007544
##   0.9180014  0.3576041  0.7604982  0.3174283
##   0.9150418  0.3649521  0.7604356  0.3208039
##   0.9321031  0.3197506  0.7599975  0.2977937
##   0.9167827  0.3578268  0.7596846  0.3159110
##   0.9139102  0.3653975  0.7597472  0.3196451
##   0.9302751  0.3226453  0.7594968  0.2982419
##   0.9206999  0.3542641  0.7614995  0.3178312
##   0.9180014  0.3593854  0.7609988  0.3192804
##   0.9325383  0.3175239  0.7596846  0.2960315
##   0.9181755  0.3573814  0.7605607  0.3174467
##   0.9144325  0.3629481  0.7594343  0.3178591
##   0.9322772  0.3184146  0.7597472  0.2966101
##   0.9173050  0.3580494  0.7601227  0.3168916
##   0.9133008  0.3656201  0.7593717  0.3190046
##   0.9334958  0.3130706  0.7591213  0.2926213
##   0.9215703  0.3491427  0.7606859  0.3137291
##   0.9178273  0.3558228  0.7598723  0.3153239
##   0.9228760  0.3500334  0.7618750  0.3165481
##   0.9198294  0.3562681  0.7614369  0.3186669
##   0.9203517  0.3569361  0.7620001  0.3201159
##   0.9238336  0.3482521  0.7620627  0.3160649
##   0.9204387  0.3602761  0.7630014  0.3237174
##   0.9200905  0.3613894  0.7630640  0.3243715
##   0.9254875  0.3437987  0.7620001  0.3137769
##   0.9215703  0.3551548  0.7623756  0.3200174
##   0.9207869  0.3584948  0.7627511  0.3223673
##   0.9233983  0.3489201  0.7619375  0.3161361
##   0.9200035  0.3596081  0.7625008  0.3223948
##   0.9188719  0.3616121  0.7622505  0.3228453
##   0.9249652  0.3482521  0.7628763  0.3177048
##   0.9199164  0.3560454  0.7614369  0.3185602
##   0.9212221  0.3587174  0.7631266  0.3232279
##   0.9257486  0.3475841  0.7632518  0.3181400
##   0.9226149  0.3549321  0.7630640  0.3212954
##   0.9221797  0.3578268  0.7635647  0.3236852
##   0.9200905  0.3569361  0.7618124  0.3197393
##   0.9205258  0.3573814  0.7622505  0.3208310
##   0.9201776  0.3502561  0.7599975  0.3128889
##   0.9234854  0.3500334  0.7623130  0.3174296
##   0.9214833  0.3551548  0.7623130  0.3198917
##   0.9205258  0.3527054  0.7609362  0.3159521
##   0.9233983  0.3511467  0.7625634  0.3184703
##   0.9223538  0.3556001  0.7630640  0.3216150
##   0.9224408  0.3542641  0.7627511  0.3203458
##   0.9190460  0.3578268  0.7613117  0.3191620
##   0.9195682  0.3549321  0.7608736  0.3168979
##   0.9202646  0.3547094  0.7613117  0.3176686
##   0.9227890  0.3533734  0.7627511  0.3199186
##   0.9221797  0.3567134  0.7632518  0.3225246
##   0.9200905  0.3611668  0.7630014  0.3241403
##   0.9221797  0.3491427  0.7611240  0.3146083
```

```
##   0.9220056  0.3522601  0.7618750  0.3176215
##   0.9211351  0.3538187  0.7616872  0.3179940
##   0.9193942  0.3524827  0.7600601  0.3140905
##   0.9143454  0.3640615  0.7596846  0.3188865
##   0.9133879  0.3640615  0.7589962  0.3175185
##   0.9247040  0.3437987  0.7614369  0.3126417
##   0.9153029  0.3647295  0.7605607  0.3209478
##   0.9152159  0.3640615  0.7603104  0.3201322
##   0.9220056  0.3464707  0.7602478  0.3115531
##   0.9150418  0.3645068  0.7603104  0.3203435
##   0.9137361  0.3631708  0.7589962  0.3170945
##   0.9234854  0.3422400  0.7601227  0.3092369
##   0.9132138  0.3611668  0.7580575  0.3142748
##   0.9121692  0.3676241  0.7591213  0.3194573
##   0.9201776  0.3482521  0.7594343  0.3107898
##   0.9132138  0.3631708  0.7586207  0.3163490
##   0.9126045  0.3653975  0.7588084  0.3177809
##   0.9272284  0.3364507  0.7611866  0.3085355
##   0.9160864  0.3596081  0.7596846  0.3167638
##   0.9147806  0.3600534  0.7588710  0.3153576
##   0.9193942  0.3607214  0.7623756  0.3226732
##   0.9192201  0.3613894  0.7624382  0.3231159
##   0.9190460  0.3620574  0.7625008  0.3235583
##   0.9220056  0.3531507  0.7621253  0.3185529
##   0.9206128  0.3549321  0.7616246  0.3184029
##   0.9213962  0.3549321  0.7621879  0.3195335
##   0.9203517  0.3598308  0.7628137  0.3231288
##   0.9181755  0.3611668  0.7616246  0.3213807
##   0.9185237  0.3596081  0.7614369  0.3202629
##   0.9237465  0.3502561  0.7625634  0.3180412
##   0.9220056  0.3549321  0.7626259  0.3204140
##   0.9206999  0.3562681  0.7620627  0.3199220
##   0.9238336  0.3504787  0.7626885  0.3184007
##   0.9231372  0.3556001  0.7636273  0.3227494
##   0.9205258  0.3598308  0.7629389  0.3233801
##   0.9220926  0.3544868  0.7625634  0.3200748
##   0.9213092  0.3551548  0.7621879  0.3196402
##   0.9213092  0.3507014  0.7609362  0.3149853
##   0.9213962  0.3531507  0.7616872  0.3176729
##   0.9207869  0.3518147  0.7608736  0.3153973
##   0.9220056  0.3469161  0.7603730  0.3120208
##   0.9199164  0.3593854  0.7623756  0.3220377
##   0.9216574  0.3562681  0.7627511  0.3213048
##   0.9214833  0.3556001  0.7624382  0.3203564
##   0.9235724  0.3531507  0.7632518  0.3208203
##   0.9221797  0.3544868  0.7626259  0.3202007
##   0.9220056  0.3553774  0.7627511  0.3208789
##   0.9168698  0.3616121  0.7608111  0.3199665
##   0.9220926  0.3527054  0.7620627  0.3182131
##   0.9186978  0.3556001  0.7604356  0.3163422
##   0.9209610  0.3522601  0.7611240  0.3161140
##   0.9180014  0.3600534  0.7611866  0.3199744
##   0.9191330  0.3556001  0.7607485  0.3169681
##   0.9233113  0.3515921  0.7626259  0.3188107
##   0.9221797  0.3544868  0.7626259  0.3202007
##   0.9223538  0.3549321  0.7628763  0.3209176
##
## Tuning parameter 'gamma' was held constant at a value of 0
```

```
##
## Tuning parameter 'min_child_weight' was held constant at a value of 1
## ROC was used to select the optimal model using  the largest value.
## The final values used for the model were nrounds = 50, max_depth = 3,
##  eta = 0.3, gamma = 0, colsample_bytree = 0.6, min_child_weight = 1
##  and subsample = 0.75.
```

```
varimp_XGB<-varImp(xgbCVD)
plot(varimp_XGB, top=10,scales=list(y=list(cex=.95)))
```



Predições Boosting

```
#Aplique o modelo ajustado ao banco de teste

#Predict cvd on test data
classPredXGB <- predict(xgbCVD, newdata=dplyr::select(testData,-cvd))

#calculate confusion Matrix and other measures of accuracy
confMatXGB <- confusionMatrix(testData$cvd, classPredXGB)
confMatXGB
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     N     Y
##          N 12440  1076
##          Y  3389  1895
##
##                Accuracy : 0.7625
##                  95% CI : (0.7564, 0.7686)
##     No Information Rate : 0.842
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3219
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7859
##             Specificity : 0.6378
##          Pos Pred Value : 0.9204
##          Neg Pred Value : 0.3586
##              Prevalence : 0.8420
##          Detection Rate : 0.6617
##    Detection Prevalence : 0.7189
##       Balanced Accuracy : 0.7119
##
##        'Positive' Class : N
##
```

-Area under the ROC curve

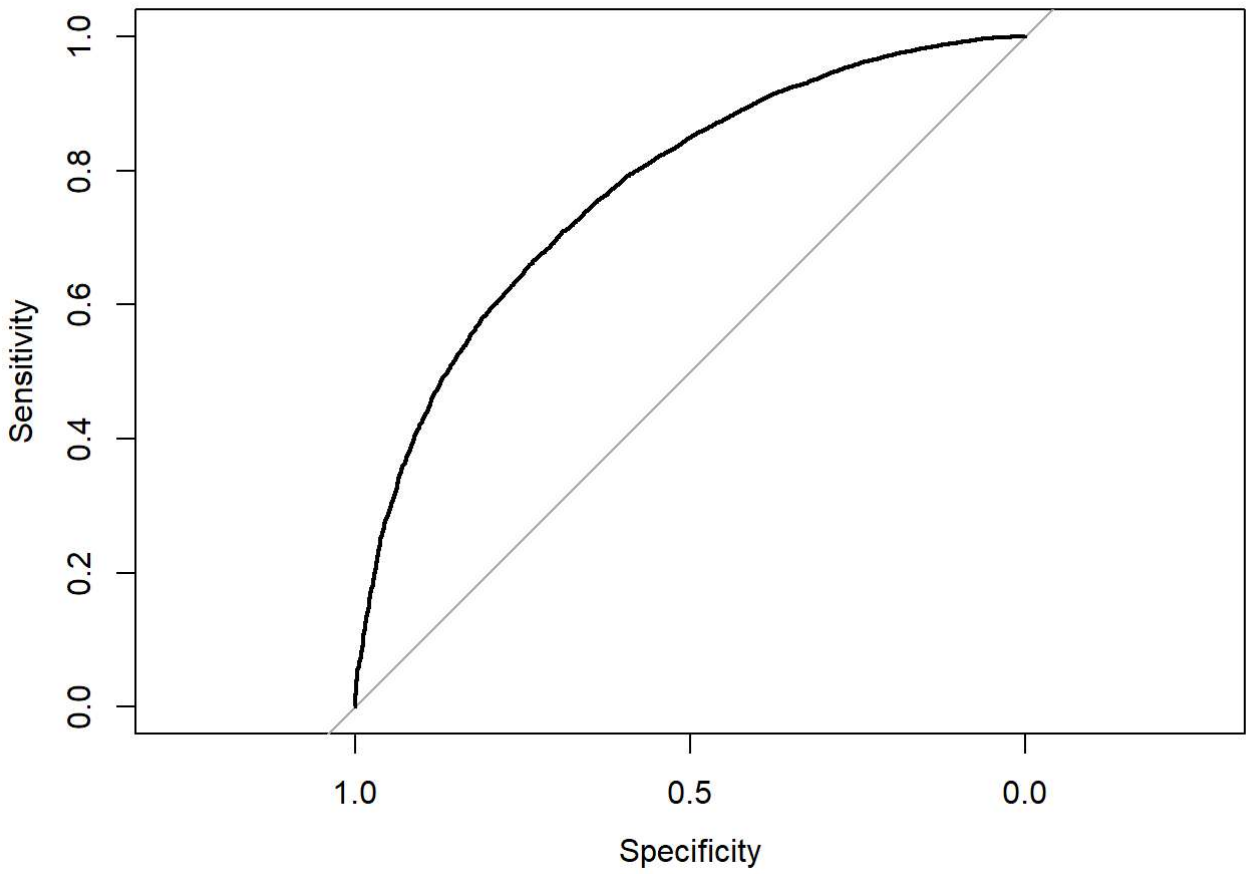
```
predicao_probXGB <- predict(xgbCVD,dplyr::select(testData,-cvd),type="prob")
p_YesXGB <- predicao_probXGB[,"Y"]
rocCurveXGB <- roc(response = testData$cvd,
                   predictor = p_YesXGB,
                   levels = rev(levels(testData$cvd)))

AUC_xgb<-pROC::auc(rocCurveXGB)
AUC_xgb
```

```
## Area under the curve: 0.7727
```

```
IC_AUC_xgb<-pROC::ci.auc(rocCurveXGB)
IC_AUC_xgb
```

```
## 95% CI: 0.7654-0.78 (DeLong)
```

```
plot(rocCurveXGB)
```



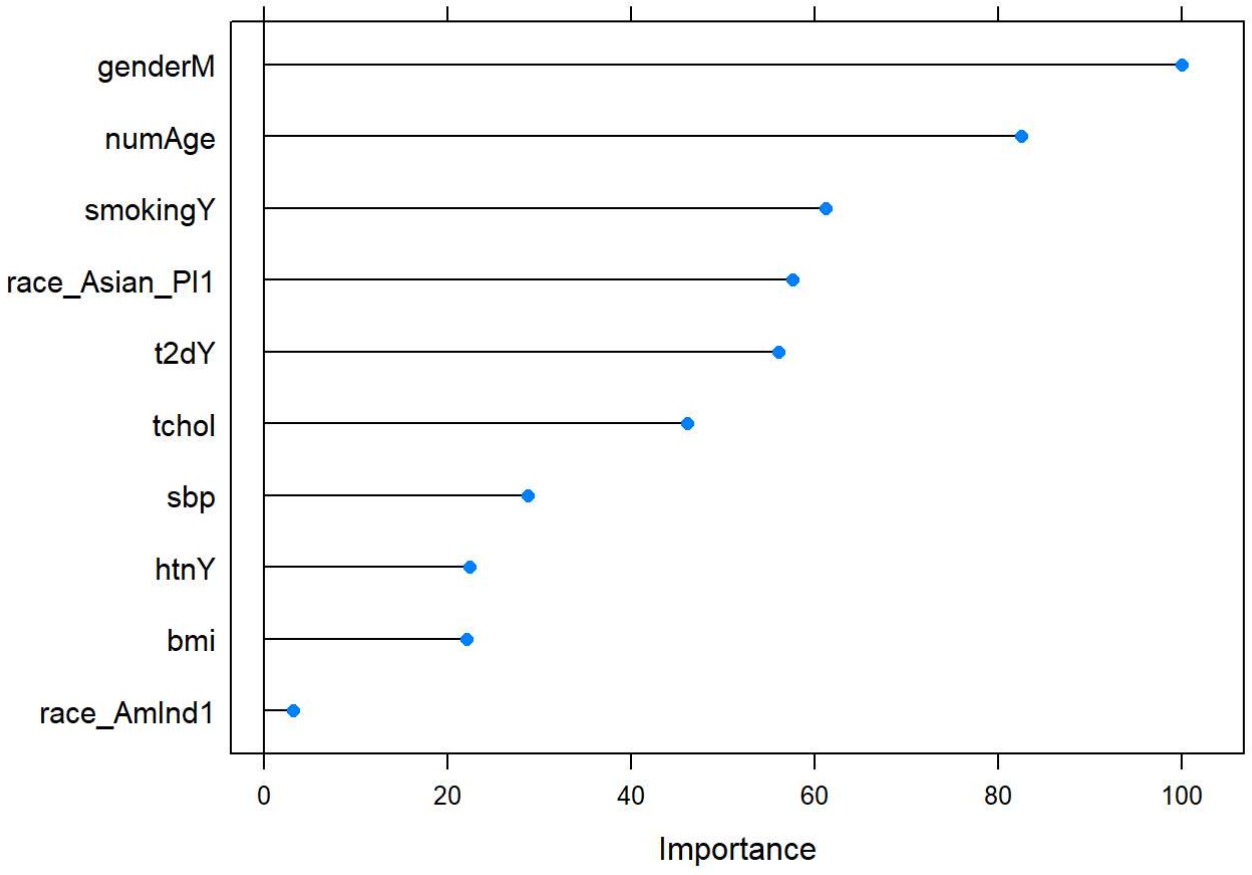
Random Forest

```
rfCVD <- train(cvd ~ ., data=trainData,
               method="rf",
               trControl=ctrl,
               metric="ROC",
               importance=T)
```

```
rfCVD
```

```
## Random Forest
##
## 106535 samples
##     11 predictor
##      2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (1 reps, 85%)
## Summary of sample sizes: 90556
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec       Accuracy   Kappa
##    2    0.7501689  0.9382834  0.2950345  0.7574942  0.2799623
##    6    0.7360185  0.8797006  0.3805389  0.7394080  0.2875364
##   11    0.7257378  0.8605501  0.3943442  0.7295200  0.2758035
##
## ROC was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
varimp_RF<-varImp(rfCVD)
plot(varimp_RF, top=10,scales=list(y=list(cex=.95)))
```



Predições Random Forest

```
#Aplique o modelo ajustado ao banco de teste

#Predict cvd on test data
classPredRF <- predict(rfCVD, newdata=dplyr::select(testData,-cvd))

#calculate confusion Matrix and other measures of accuracy
confMatRF <- confusionMatrix(testData$cvd, classPredRF)
confMatRF
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     N     Y
##          N 12725   791
##          Y  3714  1570
##
##                Accuracy : 0.7604
##                  95% CI : (0.7542, 0.7665)
##     No Information Rate : 0.8744
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2869
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7741
##             Specificity : 0.6650
##          Pos Pred Value : 0.9415
##          Neg Pred Value : 0.2971
##              Prevalence : 0.8744
##          Detection Rate : 0.6769
##    Detection Prevalence : 0.7189
##       Balanced Accuracy : 0.7195
##
##        'Positive' Class : N
##
```

-Area under the ROC curve

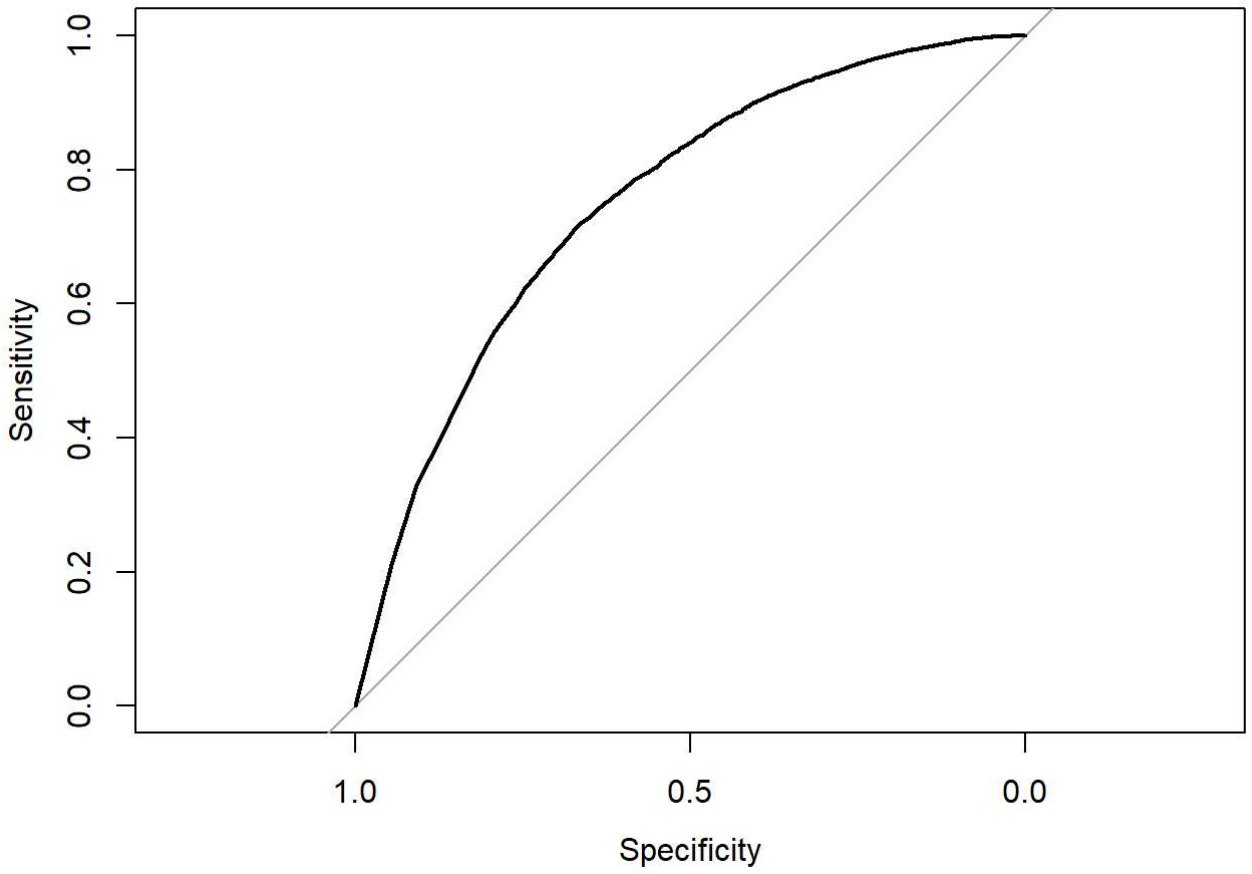
```
predicao_probRF <- predict(rfCVD,dplyr::select(testData,-cvd),type="prob")
p_YesRF <- predicao_probRF[,"Y"]
rocCurveRF <- roc(response = testData$cvd,
                  predictor = p_YesRF,
                  levels = rev(levels(testData$cvd)))

AUC_rf<-pROC::auc(rocCurveRF)
AUC_rf
```

```
## Area under the curve: 0.7513
```

```
IC_AUC_rf<-pROC::ci.auc(rocCurveRF)
IC_AUC_rf
```

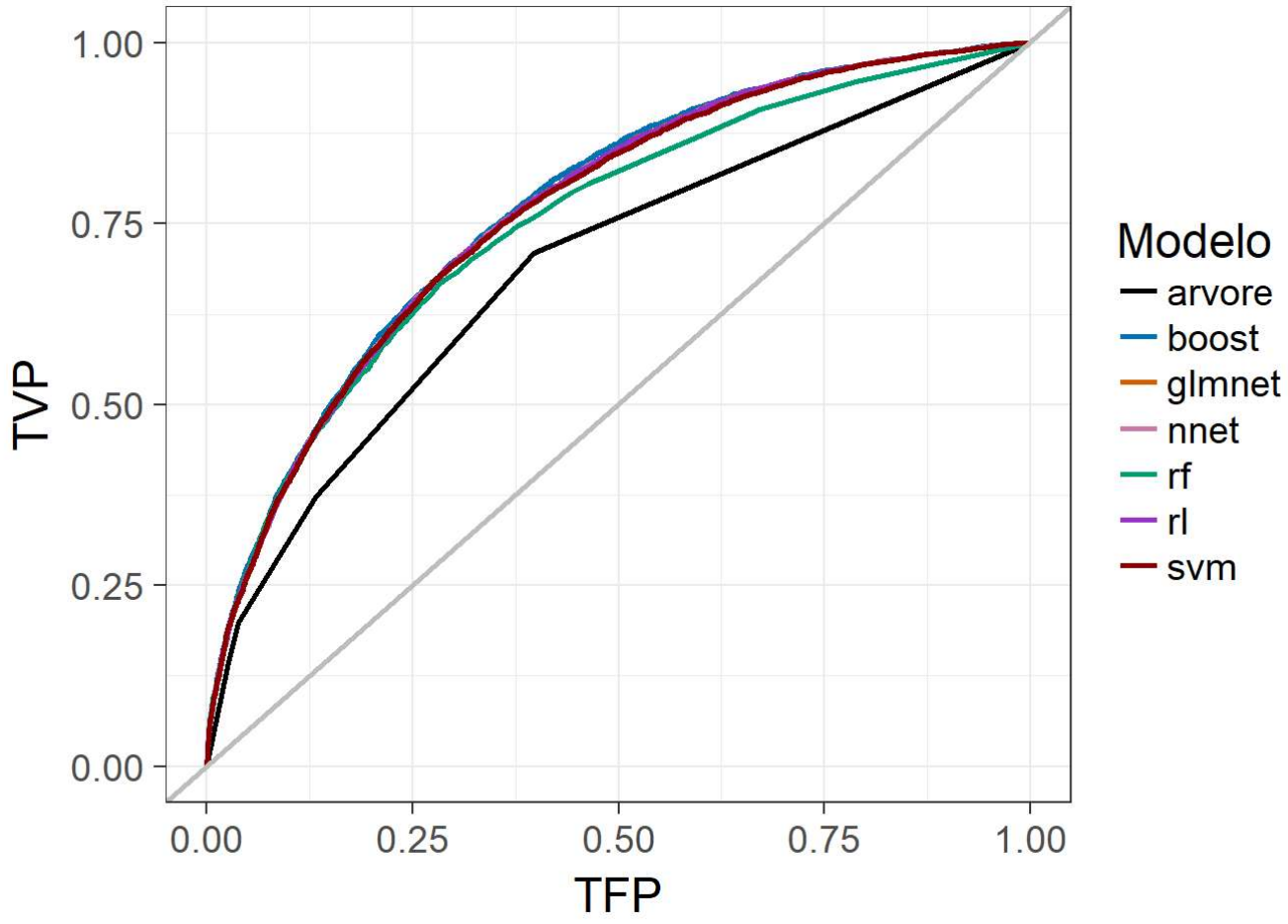
```
## 95% CI: 0.7434-0.7591 (DeLong)
```

```
plot(rocCurveRF)
```



AUC ROC para todos os modelos ajustados

```
test_roc <- function(model, data){
                    roc(data$cvd,
                    predict(model, data, type = "prob")[, "Y"])
            }

# Examine results for test set
model_list <- list(rl=reglogCVD,
                   glmnet=glmnetCVD,
                   nnet=nnetCVD,
                   svm=svmCVD,
                   arvore=rpartCVD,
                   boost=xgbCVD,
                   rf=rfCVD)

model_list_roc <- model_list %>%
                  map(test_roc, data = testData)

results_list_roc <- list(NA)
num_mod <- 1

for(the_roc in model_list_roc){
    results_list_roc[[num_mod]] <-
    data_frame(TVP = the_roc$sensitivities,
               TFP = 1 - the_roc$specificities,
               Modelo = names(model_list)[num_mod])
    num_mod <- num_mod + 1
}

results_df_roc <- bind_rows(results_list_roc)

# Plot ROC curve for all 7 models
custom_col <- c("#000000","#0072B2","#D55E00","#CC79A7","#009E73","#9932CC","#8B0000")

ggplot(aes(x = TFP,  y = TVP, group = Modelo), data = results_df_roc) +
          geom_line(aes(color = Modelo), size = 1) +
          scale_color_manual(values = custom_col) +
          geom_abline(intercept = 0, slope = 1, color = "gray", size = 1) +
          theme_bw(base_size = 18)
```



Performance (AUC e IC95%)

```
AUC<-as.data.frame(rbind(AUC_rl,AUC_glmnet,AUC_nnet,AUC_svm,AUC_rpart,AUC_xgb,AUC_rf),row.nam
es=F)
names(AUC)[1]<-"AUC"
IC_AUC<-as.data.frame(rbind(IC_AUC_rl,IC_AUC_glmnet,IC_AUC_nnet,IC_AUC_svm,IC_AUC_rpart,IC_AU
C_xgb,IC_AUC_rf),row.names=F)
names(IC_AUC)[1:3]<-c("int_INF","AUC","int_SUP")
IC_AUC<-select(IC_AUC,-AUC)
Algorithm<-c("RL","GLMNET","NNET","SVMl","RPART","Boosting","RF")
performance<-cbind(Algorithm,AUC,IC_AUC)
performance[order(AUC,decreasing = T),]
```

```
##   Algorithm       AUC   int_INF   int_SUP
## 6  Boosting 0.7727194 0.7654248 0.7800141
## 2    GLMNET 0.7686742 0.7613179 0.7760306
## 1        RL 0.7686092 0.7612518 0.7759666
## 3      NNET 0.7684823 0.7611244 0.7758402
## 4      SVMl 0.7665654 0.7591539 0.7739768
## 7        RF 0.7512702 0.7434186 0.7591217
## 5     RPART 0.6889887 0.6808923 0.6970850
```