Credit_Card_project

Viswas, Revanth, Sayari

Credit_Card_project

Sayari

June 7, 2017

Develop a model to identify fraudulent transactions.

Data Wrangling

```
file=read.csv("creditcard.csv")
mymatrix=as.matrix(file)       # checking for NA
print(colnames(mymatrix)[colSums(is.na(mymatrix)) > 0])
```

```
## character(0)
```

```
head(file,2)
```

```
##   Time        V1          V2        V3        V4          V5          V6
## 1    0 -1.359807 -0.07278117 2.5363467 1.3781552 -0.33832077  0.46238778
## 2    0  1.191857  0.26615071 0.1664801 0.4481541  0.06001765 -0.08236081
##            V7         V8         V9         V10        V11        V12
## 1  0.23959855 0.09869790  0.3637870  0.09079417 -0.5515995 -0.6178009
## 2 -0.07880298 0.08510165 -0.2554251 -0.16697441  1.6127267  1.0652353
##          V13        V14       V15        V16        V17         V18
## 1 -0.9913898 -0.3111694 1.4681770 -0.4704005  0.2079712  0.02579058
## 2  0.4890950 -0.1437723 0.6355581  0.4639170 -0.1148047 -0.18336127
##         V19         V20         V21        V22        V23         V24
## 1  0.403993  0.25141210 -0.01830678  0.2778376 -0.1104739  0.06692807
## 2 -0.145783 -0.06908314 -0.22577525 -0.6386720  0.1012880 -0.33984648
##         V25        V26          V27         V28 Amount Class
## 1 0.1285394 -0.1891148  0.133558377 -0.02105305 149.62     0
## 2 0.1671704  0.1258945 -0.008983099  0.01472417   2.69     0
```

```
print(c(" no of frauds",length(which(file$Class==1))))
```

```
## [1] " no of frauds" "492"
```

```
file=file[,2:31]
```

```
head(file,2)
```

```
##          V1          V2        V3        V4          V5          V6
## 1 -1.359807 -0.07278117 2.5363467 1.3781552 -0.33832077  0.46238778
## 2  1.191857  0.26615071 0.1664801 0.4481541  0.06001765 -0.08236081
##            V7          V8         V9         V10        V11        V12
## 1  0.23959855 0.09869790  0.3637870  0.09079417 -0.5515995 -0.6178009
## 2 -0.07880298 0.08510165 -0.2554251 -0.16697441  1.6127267  1.0652353
##          V13        V14       V15        V16        V17         V18
## 1 -0.9913898 -0.3111694 1.4681770 -0.4704005  0.2079712  0.02579058
## 2  0.4890950 -0.1437723 0.6355581  0.4639170 -0.1148047 -0.18336127
##         V19         V20         V21        V22        V23         V24
## 1  0.403993  0.25141210 -0.01830678  0.2778376 -0.1104739  0.06692807
## 2 -0.145783 -0.06908314 -0.22577525 -0.6386720  0.1012880 -0.33984648
##         V25        V26          V27         V28 Amount Class
## 1 0.1285394 -0.1891148  0.133558377 -0.02105305 149.62     0
## 2 0.1671704  0.1258945 -0.008983099  0.01472417   2.69     0
```

```
seqno=seq(1,length(file[,1]))
idx=sample(seqno,200000)
 not_idx=setdiff(seqno,idx)
 train=file[idx,]
 test=file[not_idx,]
 dim(train)    # dimensions for the training data
```

```
## [1] 200000     30
```

```
 x_train=train[,1:29]
 y_train=train[,"Class"]
 x_test=test[,1:29]
 y_test=test[,"Class"]
```

Classification Algorithms - Naive Bayes

Before PCA

```
library(e1071)
bayesModel1 = naiveBayes(as.matrix(x_train),factor(y_train))
```

```
bayesModel1
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
```

```
## naiveBayes.default(x = as.matrix(x_train), y = factor(y_train))
##
## A-priori probabilities:
## factor(y_train)
##      0      1
## 0.9983 0.0017
##
## Conditional probabilities:
##               V1
## factor(y_train)        [,1]     [,2]
##               0  0.01222851 1.925536
##               1 -4.70331334 6.702104
##
##               V2
## factor(y_train)         [,1]     [,2]
##               0 -0.006351483 1.626194
##               1  3.813661934 4.220590
##
##               V3
## factor(y_train)        [,1]     [,2]
##               0  0.01227501 1.459395
##               1 -7.06725446 6.931178
##
##               V4
## factor(y_train)         [,1]     [,2]
##               0 -0.007834009 1.397836
##               1  4.489122088 2.854061
##
##               V5
## factor(y_train)         [,1]     [,2]
##               0  0.004971739 1.360567
##               1 -3.074967825 5.337522
##
##               V6
## factor(y_train)         [,1]     [,2]
##               0  0.001162553 1.330726
##               1 -1.376410346 1.948252
##
##               V7
## factor(y_train)         [,1]     [,2]
##               0  0.008871949 1.187548
##               1 -5.618393748 7.367950
##
##               V8
## factor(y_train)        [,1]     [,2]
##               0 -0.00087658 1.167538
```

```
##               1  0.23433738 7.571570
##
##                V9
## factor(y_train)          [,1]     [,2]
##               0  0.004905328 1.091622
##               1 -2.585144950 2.497433
##
##                V10
## factor(y_train)         [,1]     [,2]
##               0  0.01077754 1.046344
##               1 -5.75949610 4.871139
##
##                V11
## factor(y_train)          [,1]     [,2]
##               0 -0.005702064 1.003800
##               1  3.887435382 2.585502
##
##                V12
## factor(y_train)         [,1]      [,2]
##               0  0.01083298 0.9469475
##               1 -6.31149802 4.5968065
##
##                V13
## factor(y_train)          [,1]      [,2]
##               0  0.002217953 0.9944846
##               1 -0.169449586 1.1332543
##
##                V14
## factor(y_train)         [,1]     [,2]
##               0  0.01100557 0.897034
##               1 -7.14715373 4.246473
##
##                V15
## factor(y_train)           [,1]      [,2]
##               0  0.0004409582 0.9157867
##               1 -0.1233307319 1.0624572
##
##                V16
## factor(y_train)          [,1]      [,2]
##               0  0.007592594 0.8458842
##               1 -4.091858895 3.8429019
##
##                V17
## factor(y_train)         [,1]      [,2]
##               0  0.01029207 0.7519954
##               1 -6.50865358 6.9247315
```

```
##
##                  V18
## factor(y_train)          [,1]      [,2]
##               0  0.003467616 0.8261961
##               1 -2.175357241 2.8986403
##
##                  V19
## factor(y_train)          [,1]      [,2]
##               0 -0.001617277 0.8116022
##               1  0.621795908 1.5585301
##
##                  V20
## factor(y_train)           [,1]      [,2]
##               0 -0.0003497932 0.7679195
##               1  0.3372152528 1.4336489
##
##                  V21
## factor(y_train)          [,1]      [,2]
##               0 -0.003207543 0.7137173
##               1  0.828957388 4.4548565
##
##                  V22
## factor(y_train)           [,1]      [,2]
##               0 -0.0002709806 0.7226719
##               1 -0.0624512715 1.6680735
##
##                  V23
## factor(y_train)           [,1]     [,2]
##               0 -0.0001223251 0.621629
##               1 -0.0496280051 1.814370
##
##                  V24
## factor(y_train)           [,1]      [,2]
##               0 -4.815096e-05 0.6059082
##               1 -1.091646e-01 0.5018155
##
##                  V25
## factor(y_train)         [,1]      [,2]
##               0 0.001256657 0.5199837
##               1 0.059910099 0.8311824
##
##                  V26
## factor(y_train)           [,1]      [,2]
##               0 -0.0004758614 0.4825606
##               1  0.0552040639 0.4632689
##
```

```
##                V27
## factor(y_train)          [,1]      [,2]
##               0 -0.0007192794 0.4001839
##               1  0.1382431589 1.3582400
##
##                V28
## factor(y_train)          [,1]      [,2]
##               0 -0.0002016082 0.3218548
##               1  0.1012099580 0.5275893
##
##                Amount
## factor(y_train)      [,1]     [,2]
##               0  88.00765 249.5345
##               1 111.46082 232.0898
```

```
pred_Bayes1=predict(bayesModel1,newdata = as.matrix(x_test))
bayesConfusionMatrix1 = table(pred_Bayes1,factor(y_test))
```

```
bayesConfusionMatrix1
```

```
##
## pred_Bayes1     0     1
##           0 82862    33
##           1  1793   119
```

Chisq test

```
BayesChiSq1 = chisq.test(bayesConfusionMatrix1)
```

```
## Warning in chisq.test(bayesConfusionMatrix1): Chi-squared approximation may
## be incorrect
```

```
BayesChiSq1
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  bayesConfusionMatrix1
## X-squared = 3960.3, df = 1, p-value < 2.2e-16
```

Metrics

```
library(caret)
```

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

```
## Loading required package: ggplot2
```

```r
pred_Bayes1 = as.factor(pred_Bayes1)
y_test =  as.factor(y_test)

BayesAccuracy1=sum(diag(bayesConfusionMatrix1))/sum(bayesConfusionMatrix1)
print(c("BayesAccuracy1",BayesAccuracy1))
```

```
## [1] "BayesAccuracy1"    "0.978468758475126"
```

```r
Bayes_precision1 = posPredValue(pred_Bayes1, y_test)
print(c("Bayes_precision1",Bayes_precision1))
```

```
## [1] "Bayes_precision1"  "0.999601906025695"
```

```r
Bayes_recall1 = sensitivity(pred_Bayes1, y_test)
print(c("Bayes_recall1",Bayes_recall1))
```

```
## [1] "Bayes_recall1"     "0.978819916130175"
```

```r
Bayes_Spec1 = bayesConfusionMatrix1[2,2]/(bayesConfusionMatrix1[2,2] +
              bayesConfusionMatrix1[1,2])
print(c("Bayes_Spec1",Bayes_Spec1))
```

```
## [1] "Bayes_Spec1"       "0.782894736842105"
```

```r
Bayes_F1_1 = (2 * Bayes_precision1 * Bayes_recall1) / (Bayes_precision1 +
                                Bayes_recall1)
print(c("Bayes_F1_1",Bayes_F1_1))
```

```
## [1] "Bayes_F1_1"        "0.989101760668457"
```

checking bayes for cut off =1

```r
f = predict(bayesModel1,as.matrix(x_test),type="raw")

predict = matrix(f[,2],ncol=1)

idx = which(predict==1)

predict[idx] =1
seqno = seq(0,dim(f)[1])

not_idx = setdiff(seqno,idx)

predict[not_idx]=0
table(predict,factor(y_test))
```

```
##
## predict     0     1
##       0 83732    47
##       1   923   105
```

Linear discriminant analysis

```r
library(MASS)

x_train1 = x_train
ldaModel1 = lda((y_train)~as.matrix(x_train1))
x_train1 = x_test

ldaModel1
```

```
## Call:
## lda((y_train) ~ as.matrix(x_train1))
##
## Prior probabilities of groups:
##      0      1
## 0.9983 0.0017
##
## Group means:
##   as.matrix(x_train1)V1 as.matrix(x_train1)V2 as.matrix(x_train1)V3
## 0            0.01222851          -0.006351483            0.01227501
## 1           -4.70331334           3.813661934           -7.06725446
##   as.matrix(x_train1)V4 as.matrix(x_train1)V5 as.matrix(x_train1)V6
## 0          -0.007834009           0.004971739           0.001162553
## 1           4.489122088          -3.074967825          -1.376410346
##   as.matrix(x_train1)V7 as.matrix(x_train1)V8 as.matrix(x_train1)V9
## 0           0.008871949           -0.00087658           0.004905328
## 1          -5.618393748            0.23433738          -2.585144950
##   as.matrix(x_train1)V10 as.matrix(x_train1)V11 as.matrix(x_train1)V12
## 0             0.01077754           -0.005702064             0.01083298
## 1            -5.75949610            3.887435382            -6.31149802
##   as.matrix(x_train1)V13 as.matrix(x_train1)V14 as.matrix(x_train1)V15
## 0            0.002217953             0.01100557           0.0004409582
## 1           -0.169449586            -7.14715373          -0.1233307319
##   as.matrix(x_train1)V16 as.matrix(x_train1)V17 as.matrix(x_train1)V18
## 0            0.007592594             0.01029207            0.003467616
## 1           -4.091858895            -6.50865358           -2.175357241
##   as.matrix(x_train1)V19 as.matrix(x_train1)V20 as.matrix(x_train1)V21
## 0           -0.001617277          -0.0003497932           -0.003207543
## 1            0.621795908           0.3372152528            0.828957388
##   as.matrix(x_train1)V22 as.matrix(x_train1)V23 as.matrix(x_train1)V24
## 0          -0.0002709806          -0.0001223251          -4.815096e-05
## 1          -0.0624512715          -0.0496280051          -1.091646e-01
##   as.matrix(x_train1)V25 as.matrix(x_train1)V26 as.matrix(x_train1)V27
## 0            0.001256657          -0.0004758614          -0.0007192794
## 1            0.059910099           0.0552040639           0.1382431589
##   as.matrix(x_train1)V28 as.matrix(x_train1)Amount
```

```
## 0          -0.0002016082                 88.00765
## 1           0.1012099580                111.46082
##
## Coefficients of linear discriminants:
##                                      LD1
## as.matrix(x_train1)V1     -0.0894634674
## as.matrix(x_train1)V2      0.1416024724
## as.matrix(x_train1)V3     -0.2375625578
## as.matrix(x_train1)V4      0.1794117840
## as.matrix(x_train1)V5     -0.1053937013
## as.matrix(x_train1)V6     -0.0791837740
## as.matrix(x_train1)V7     -0.3278823302
## as.matrix(x_train1)V8      0.0293557657
## as.matrix(x_train1)V9     -0.1738556059
## as.matrix(x_train1)V10    -0.3891739028
## as.matrix(x_train1)V11     0.3074445356
## as.matrix(x_train1)V12    -0.5208828813
## as.matrix(x_train1)V13    -0.0124477247
## as.matrix(x_train1)V14    -0.6409119332
## as.matrix(x_train1)V15    -0.0088174497
## as.matrix(x_train1)V16    -0.4427915080
## as.matrix(x_train1)V17    -0.7541288998
## as.matrix(x_train1)V18    -0.2634534442
## as.matrix(x_train1)V19     0.0865749295
## as.matrix(x_train1)V20     0.0057530515
## as.matrix(x_train1)V21     0.1022759765
## as.matrix(x_train1)V22     0.0064839783
## as.matrix(x_train1)V23     0.0072426981
## as.matrix(x_train1)V24    -0.0260878329
## as.matrix(x_train1)V25     0.0246314864
## as.matrix(x_train1)V26     0.0207422565
## as.matrix(x_train1)V27     0.0737176271
## as.matrix(x_train1)V28     0.0582844056
## as.matrix(x_train1)Amount  0.0003737009
```

Confusion Matrix

```
pred_Lda1 = predict(ldaModel1,newdata = (x_train1))$class
ldaConfusionMatrix1 = table(pred_Lda1,y_test)
ldaConfusionMatrix1
```

```
##          y_test
## pred_Lda1     0     1
##         0 84642    37
```

```
##          1    13   115
```

Metrics

```r
library(caret)
pred_Lda1 = as.factor(pred_Lda1)
y_test =  as.factor(y_test)

Lda_Accuracy1=sum(diag(ldaConfusionMatrix1))/sum(ldaConfusionMatrix1)
print(c("Lda_Accuracy1",Lda_Accuracy1))
```

```
## [1] "Lda_Accuracy1"     "0.999410426026153"
```

```r
Lda_precision1 = posPredValue(pred_Lda1, y_test)
print(c("Lda_precision1",Lda_precision1))
```

```
## [1] "Lda_precision1"   "0.99956305577534"
```

```r
Lda_recall1 = sensitivity(pred_Lda1, y_test)
print(c("Lda_recall1",Lda_recall1))
```

```
## [1] "Lda_recall1"       "0.999846435532455"
```

```r
Lda_Spec1 = ldaConfusionMatrix1[2,2]/(ldaConfusionMatrix1[2,2] +
              ldaConfusionMatrix1[1,2])
print(c("Lda_Spec1",Lda_Spec1))
```

```
## [1] "Lda_Spec1"         "0.756578947368421"
```

```r
Lda_F1_1 = (2 * Lda_precision1 * Lda_recall1) / (Lda_precision1 +
                                    Lda_recall1)
print(c("Lda_F1_1",Lda_F1_1))
```

```
## [1] "Lda_F1_1"          "0.999704725571947"
```

Classification

```r
library(rpart)
library(rattle)
```

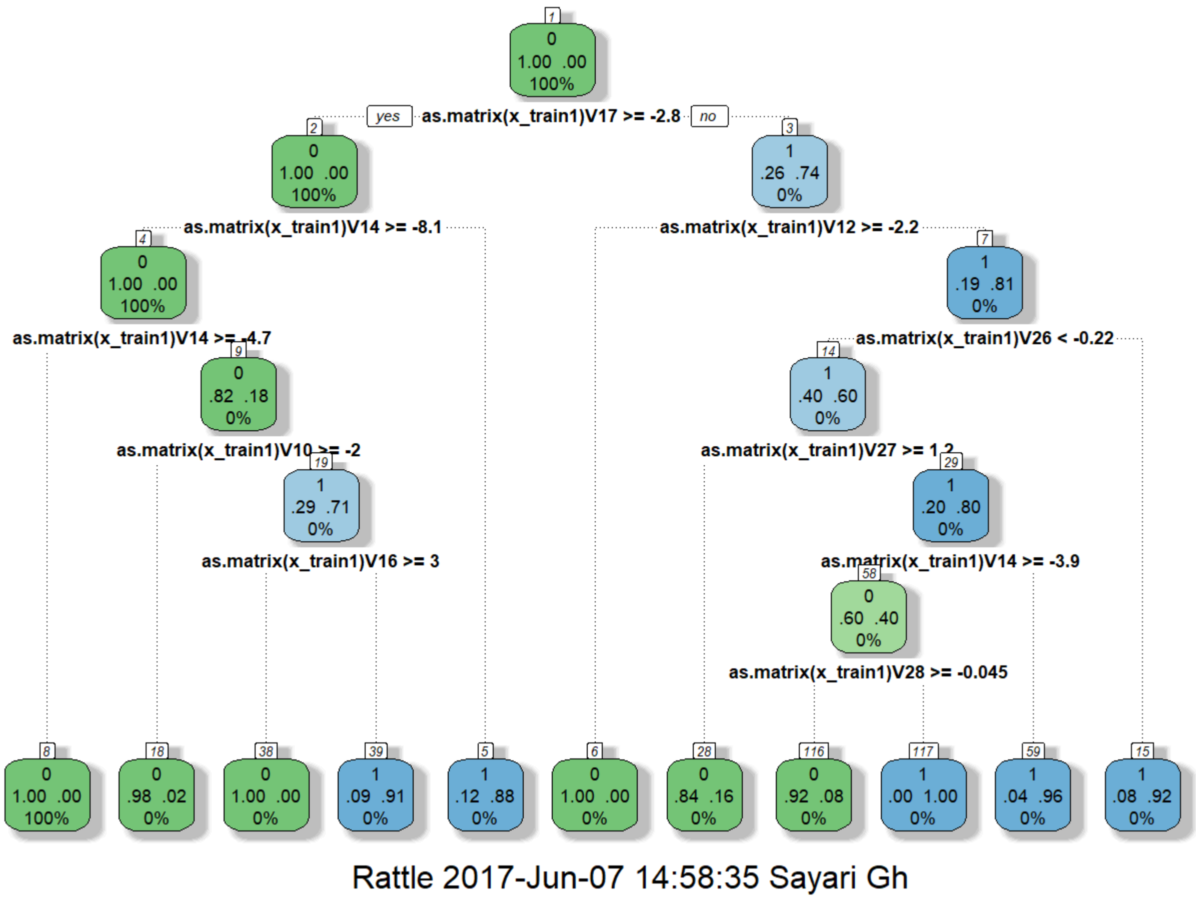
```
## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```r
library(rpart.plot)
library(RColorBrewer)

x_train1=x_train
```

```
cTreeModel1=rpart(factor(y_train)~as.matrix(x_train1),method = 'class')

fancyRpartPlot(cTreeModel1)
```



Confusion Matrix

```
x_train1=x_test
pred_CTree1 = round(predict(cTreeModel1,x_train1))
pred_CTree1=pred_CTree1[,2]
cTreeConfusionMatrix1 = table(pred_CTree1,y_test)
cTreeConfusionMatrix1
```

```
##              y_test
## pred_CTree1     0     1
```

```
##           0 84641    40
##           1    14   112
```

Chisq Test

```r
cTreeChisq1 = chisq.test(cTreeConfusionMatrix1)
```

```
## Warning in chisq.test(cTreeConfusionMatrix1): Chi-squared approximation may
## be incorrect
```

```r
cTreeChisq1
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  cTreeConfusionMatrix1
## X-squared = 55009, df = 1, p-value < 2.2e-16
```

Metrics

```r
library(caret)
pred_CTree1 = as.factor(pred_CTree1)
y_test =  as.factor(y_test)

cTree_accuracy1=sum(diag(cTreeConfusionMatrix1))/sum(cTreeConfusionMatrix1)
print(c("cTree_accuracy1",cTree_accuracy1))
```

```
## [1] "cTree_accuracy1"   "0.999363260108246"
```

```r
cTree_precision1 = posPredValue(pred_CTree1, y_test)
print(c("cTree_precision1",cTree_precision1))
```

```
## [1] "cTree_precision1" "0.99952763902174"
```

```r
cTree_recall1 = sensitivity(pred_CTree1, y_test)
print(c("cTree_recall1",cTree_recall1))
```

```
## [1] "cTree_recall1"     "0.999834622881106"
```

```r
cTree_Spec1 = cTreeConfusionMatrix1[2,2]/(cTreeConfusionMatrix1[2,2] +
              cTreeConfusionMatrix1[1,2])
print(c("cTree_Spec1",cTree_Spec1))
```

```
## [1] "cTree_Spec1"       "0.736842105263158"
```

```r
cTree_F1_1 = (2 * cTree_precision1 * cTree_recall1) / (cTree_precision1 +
                                 cTree_recall1)
print(c("cTree_F1_1",cTree_F1_1))
```

```
## [1] "cTree_F1_1"        "0.999681107384136"
```

C4.5

```
library(RWeka)

C4.5Model1 =J48(factor(y_train)~.,x_train,
        control = Weka_control(), options = NULL)
summary(C4.5Model1)
```

```
##
## === Summary ===
##
## Correctly Classified Instances      199945               99.9725 %
## Incorrectly Classified Instances        55                0.0275 %
## Kappa statistic                          0.9124
## Mean absolute error                      0.0005
## Root mean squared error                  0.0165
## Relative absolute error                 16.0737 %
## Root relative squared error             40.1214 %
## Total Number of Instances           200000
##
## === Confusion Matrix ===
##
##       a      b   <-- classified as
##  199658      2 |      a = 0
##      53    287 |      b = 1
```

Confusion Matrix

```
pred_C4.5_1 = predict(C4.5Model1,x_test)
C4.5ConfusionMatrix1 = table(pred_C4.5_1,y_test)
C4.5ConfusionMatrix1
```

```
##            y_test
## pred_C4.5_1     0     1
##           0 84645    39
##           1    10   113
```

Chisq Test

```
C4.5Chisq1 = chisq.test(C4.5ConfusionMatrix1)
```

```
## Warning in chisq.test(C4.5ConfusionMatrix1): Chi-squared approximation may
## be incorrect
```

```
C4.5Chisq1
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  C4.5ConfusionMatrix1
## X-squared = 57371, df = 1, p-value < 2.2e-16
```

Metrics

```
library(caret)
pred_C4.5_1 = as.factor(pred_C4.5_1)
y_test =  as.factor(y_test)

C4.5_accuracy1=sum(diag(C4.5ConfusionMatrix1))/sum(C4.5ConfusionMatrix1)
print(c("C4.5_accuracy1",C4.5_accuracy1))
```

```
## [1] "C4.5_accuracy1"   "0.99942221750563"
```

```
C4.5_precision1 = posPredValue(pred_C4.5_1, y_test)
print(c("C4.5_precision1",C4.5_precision1))
```

```
## [1] "C4.5_precision1"   "0.999539464361627"
```

```
C4.5_recall1 = sensitivity(pred_C4.5_1, y_test)
print(c("C4.5_recall1",C4.5_recall1))
```

```
## [1] "C4.5_recall1"      "0.999881873486504"
```

```
C4.5_Spec1 = C4.5ConfusionMatrix1[2,2]/(C4.5ConfusionMatrix1[2,2] +
              C4.5ConfusionMatrix1[1,2])
print(c("C4.5_Spec1",C4.5_Spec1))
```

```
## [1] "C4.5_Spec1"        "0.743421052631579"
```

```
C4.5_F1_1 = (2 * C4.5_precision1 * C4.5_recall1) / (C4.5_precision1 +
                                  C4.5_recall1)
print(c("C4.5_F1_1",C4.5_F1_1))
```

```
## [1] "C4.5_F1_1"        "0.99971063960458"
```

C5.0

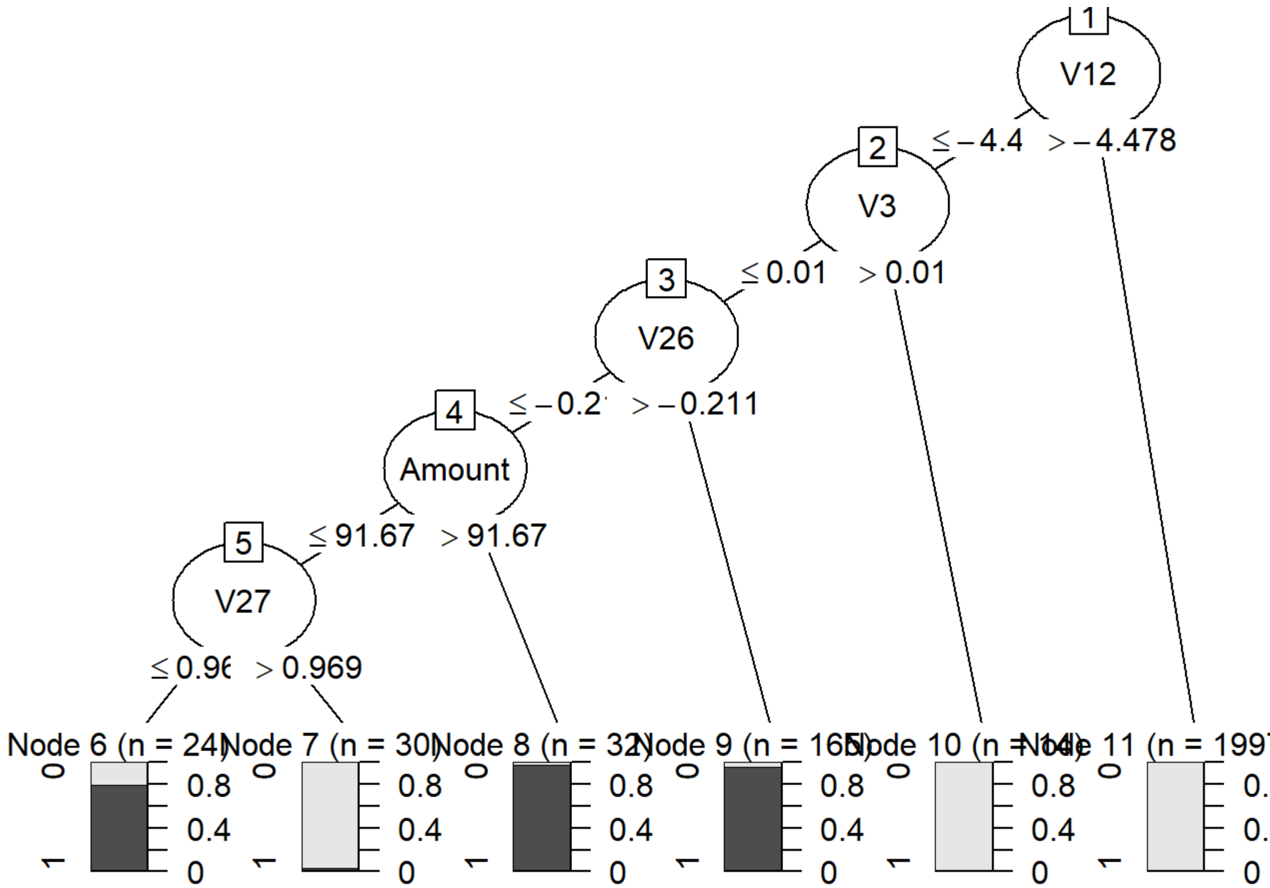
```
library(C50)
```

```r
C5.0Model1 =C5.0(x_train,
        factor(y_train),
        trials=20,rules = FALSE)

C5.0Model1
```

```
##
## Call:
## C5.0.default(x = x_train, y = factor(y_train), trials = 20, rules = FALSE)
##
## Classification Tree
## Number of samples: 200000
## Number of predictors: 29
##
## Number of boosting iterations: 20
## Average tree size: 6.9
##
## Non-standard options: attempt to group attributes
```

```r
plot(C5.0Model1)
```



Confusion Matrix

```
pred_C5.0_1 = predict(C5.0Model1,x_test)
C5.0ConfusionMatrix1 = table(pred_C5.0_1,y_test)
C5.0ConfusionMatrix1
```

```
##             y_test
## pred_C5.0_1     0     1
##           0 84640    37
##           1    15   115
```

Chisq Test

```
C5.0Chisq1 = chisq.test(C5.0ConfusionMatrix1)
```

```
## Warning in chisq.test(C5.0ConfusionMatrix1): Chi-squared approximation may
## be incorrect
```

```
C5.0Chisq1
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  C5.0ConfusionMatrix1
## X-squared = 56225, df = 1, p-value < 2.2e-16
```

Metrics

```
library(caret)
pred_C5.0_1 = as.factor(pred_C5.0_1)
y_test =  as.factor(y_test)

C5.0_Accuracy1=sum(diag(C5.0ConfusionMatrix1))/sum(C5.0ConfusionMatrix1)
print(c("C5.0_Accuracy1",C5.0_Accuracy1))
```

```
## [1] "C5.0_Accuracy1"  "0.9993868430672"
```

```
C5.0_precision1 = posPredValue(pred_C5.0_1, y_test)
print(c("C5.0_precision1",C5.0_precision1))
```

```
## [1] "C5.0_precision1"   "0.999563045455082"
```

```
C5.0_recall1 = sensitivity(pred_C5.0_1, y_test)
print(c("C5.0_recall1",C5.0_recall1))
```

```
## [1] "C5.0_recall1"      "0.999822810229756"
```

```
C5.0_Spec1 = C5.0ConfusionMatrix1[2,2]/(C5.0ConfusionMatrix1[2,2] +
              C5.0ConfusionMatrix1[1,2])
print(c("C5.0_Spec1",C5.0_Spec1))
```

```
## [1] "C5.0_Spec1"        "0.756578947368421"
```

```
C5.0_F1_1 = (2 * C5.0_precision1 * C5.0_recall1) / (C5.0_precision1 +
                                C5.0_recall1)
print(c("C5.0_F1_1",C5.0_F1_1))
```

```
## [1] "C5.0_F1_1"         "0.999692910967803"
```

Logit

```
x_train1=x_train
logitModel1 = glm(y_train~as.matrix(x_train1),
                  family=binomial(link="logit"))
x_train1=x_test
```

```
logitModel1
```

```
##
## Call:  glm(formula = y_train ~ as.matrix(x_train1), family = binomial(link = "logit"))
##
## Coefficients:
##              (Intercept)      as.matrix(x_train1)V1
##                -8.748882                   0.064460
##    as.matrix(x_train1)V2      as.matrix(x_train1)V3
##                 0.009153                   0.054222
##    as.matrix(x_train1)V4      as.matrix(x_train1)V5
##                 0.622932                   0.093040
##    as.matrix(x_train1)V6      as.matrix(x_train1)V7
##                -0.095085                  -0.110427
##    as.matrix(x_train1)V8      as.matrix(x_train1)V9
##                -0.179121                  -0.271157
##   as.matrix(x_train1)V10     as.matrix(x_train1)V11
##                -0.764732                   0.026824
##   as.matrix(x_train1)V12     as.matrix(x_train1)V13
##                 0.061078                  -0.375265
##   as.matrix(x_train1)V14     as.matrix(x_train1)V15
##                -0.595437                  -0.047490
##   as.matrix(x_train1)V16     as.matrix(x_train1)V17
##                -0.201086                   0.016916
##   as.matrix(x_train1)V18     as.matrix(x_train1)V19
##                -0.032660                   0.060423
##   as.matrix(x_train1)V20     as.matrix(x_train1)V21
##                -0.428706                   0.343328
##   as.matrix(x_train1)V22     as.matrix(x_train1)V23
##                 0.516923                  -0.130714
##   as.matrix(x_train1)V24     as.matrix(x_train1)V25
##                 0.219777                   0.054671
##   as.matrix(x_train1)V26     as.matrix(x_train1)V27
##                -0.038760                  -0.968604
##   as.matrix(x_train1)V28  as.matrix(x_train1)Amount
##                -0.452087                   0.000934
##
## Degrees of Freedom: 199999 Total (i.e. Null);  199970 Residual
## Null Deviance:       5016
```

```
## Residual Deviance: 1519  AIC: 1579
```

```
pred_logit1=round(predict(logitModel1,x_train1,type="response"),0)
logitConfusionMatrix1 = table(pred_logit1,y_test)
logitConfusionMatrix1
```

```
##             y_test
## pred_logit1     0     1
##           0 84646    63
##           1     9    89
```

```
lgChisq1 = chisq.test(logitConfusionMatrix1)
```

```
## Warning in chisq.test(logitConfusionMatrix1): Chi-squared approximation may
## be incorrect
```

```
lgChisq1
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  logitConfusionMatrix1
## X-squared = 44546, df = 1, p-value < 2.2e-16
```

Metrics

```
library(caret)
pred_logit1 = as.factor(pred_logit1)
y_test =  as.factor(y_test)

lg_Accuracy1=sum(diag(logitConfusionMatrix1))/sum(logitConfusionMatrix1)
print(c("lg_Accuracy1",lg_Accuracy1))
```

```
## [1] "lg_Accuracy1"      "0.999151013477661"
```

```
lg_precision1 = posPredValue(pred_logit1, y_test)
print(c("lg_precision1",lg_precision1))
```

```
## [1] "lg_precision1"     "0.999256277373125"
```

```
lg_recall1 = sensitivity(pred_logit1, y_test)
print(c("lg_recall1",lg_recall1))
```

```
## [1] "lg_recall1"        "0.999893686137854"
```

```
lg_Spec1 = logitConfusionMatrix1[2,2]/(logitConfusionMatrix1[2,2] +
              logitConfusionMatrix1[1,2])
print(c("lg_Spec1",lg_Spec1))
```

```
## [1] "lg_Spec1"          "0.585526315789474"
```

```r
lg_F1_1 = (2 * lg_precision1 * lg_recall1) / (lg_precision1 +
                                lg_recall1)
print(c("lg_F1_1",lg_F1_1))
```

```
## [1] "lg_F1_1"           "0.999574880139817"
```

Probit

```r
x_train1=x_train
probitModel1 = glm(y_train~as.matrix(x_train1),
                  family=binomial(link="probit"))
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
x_train1=x_test
probitModel1
```

```
##
## Call:  glm(formula = y_train ~ as.matrix(x_train1), family = binomial(link = "probit"))
##
## Coefficients:
##               (Intercept)      as.matrix(x_train1)V1
##                 -3.771908                   0.024732
##     as.matrix(x_train1)V2      as.matrix(x_train1)V3
##                  0.008223                   0.012737
##     as.matrix(x_train1)V4      as.matrix(x_train1)V5
##                  0.216370                   0.036620
##     as.matrix(x_train1)V6      as.matrix(x_train1)V7
##                 -0.045331                  -0.047107
##     as.matrix(x_train1)V8      as.matrix(x_train1)V9
##                 -0.069558                  -0.140866
##    as.matrix(x_train1)V10     as.matrix(x_train1)V11
##                 -0.238934                   0.004900
##    as.matrix(x_train1)V12     as.matrix(x_train1)V13
##                 -0.011756                  -0.134907
##    as.matrix(x_train1)V14     as.matrix(x_train1)V15
##                 -0.240516                  -0.010408
##    as.matrix(x_train1)V16     as.matrix(x_train1)V17
##                 -0.059503                   0.014708
##    as.matrix(x_train1)V18     as.matrix(x_train1)V19
##                 -0.021147                   0.020285
##    as.matrix(x_train1)V20     as.matrix(x_train1)V21
##                 -0.155494                   0.109455
##    as.matrix(x_train1)V22     as.matrix(x_train1)V23
```

```
##                 0.163374                   -0.045075
##    as.matrix(x_train1)V24     as.matrix(x_train1)V25
##                 0.052587                    0.010146
##    as.matrix(x_train1)V26     as.matrix(x_train1)V27
##                -0.038022                   -0.344297
##    as.matrix(x_train1)V28  as.matrix(x_train1)Amount
##                -0.156487                    0.000407
##
## Degrees of Freedom: 199999 Total (i.e. Null);  199970 Residual
## Null Deviance:       5016
## Residual Deviance: 1458  AIC: 1518
```

```
pred_probit1=round(predict(probitModel1,x_train1,type="response"),0)
probitConfusionMatrix1 = table(pred_probit1,y_test)
probitConfusionMatrix1
```

```
##             y_test
## pred_probit1     0     1
##            0 84646    75
##            1     9    77
```

```
pbChisq1 = chisq.test(probitConfusionMatrix1)
```

```
## Warning in chisq.test(probitConfusionMatrix1): Chi-squared approximation
## may be incorrect
```

```
pbChisq1
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  probitConfusionMatrix1
## X-squared = 37921, df = 1, p-value < 2.2e-16
```

Metrics

```
library(caret)
pred_probit1 = as.factor(pred_probit1)
y_test =  as.factor(y_test)

pb_Accuracy1=sum(diag(probitConfusionMatrix1))/sum(probitConfusionMatrix1)
print(c("pb_Accuracy1",pb_Accuracy1))
```

```
## [1] "pb_Accuracy1"      "0.999009515723938"
```

```
pb_precision1 = posPredValue(pred_probit1, y_test)
print(c("pb_precision1",pb_precision1))
```

```
## [1] "pb_precision1"     "0.999114741327416"
```

```
pb_recall1 = sensitivity(pred_probit1, y_test)
print(c("pb_recall1",pb_recall1))
```

```
## [1] "pb_recall1"        "0.999893686137854"
```

```
pb_Spec1 = probitConfusionMatrix1[2,2]/(probitConfusionMatrix1[2,2] +
              probitConfusionMatrix1[1,2])
print(c("pb_Spec1",pb_Spec1))
```

```
## [1] "pb_Spec1"          "0.506578947368421"
```

```
pb_F1_1 = (2 * pb_precision1 * pb_recall1) / (pb_precision1 +
                                 pb_recall1)
print(c("pb_F1_1",pb_F1_1))
```

```
## [1] "pb_F1_1"           "0.999504061968638"
```

Boosting

```
library(xgboost)
objectives=c("reg:linear","reg:logistic","binary:logistic")

for (x in objectives)
{
  print(x)
  xgboostModel1 = xgboost(data=(as.matrix(x_train)),
                  label=y_train,
                  objective = x, nrounds=10)
  print(names(xgboostModel1))

  pred_Boost1 = round(predict(xgboostModel1,as.matrix(x_test)),0)

  boostConfusionMatrix1 = table(pred_Boost1,y_test)
  print(boostConfusionMatrix1)

  Boostchisq1 = chisq.test(boostConfusionMatrix1)
  print(Boostchisq1)
  Boost_Accuracy1=sum(diag(boostConfusionMatrix1))/sum(boostConfusionMatrix1)
  print(Boost_Accuracy1)
}
```

```
## [1] "reg:linear"
## [1]  train-rmse:0.350234
## [2]  train-rmse:0.245467
## [3]  train-rmse:0.172230
```

```
## [4]  train-rmse:0.121104
## [5]  train-rmse:0.085505
## [6]  train-rmse:0.060873
## [7]  train-rmse:0.044039
## [8]  train-rmse:0.032744
## [9]  train-rmse:0.025426
## [10] train-rmse:0.020847
## [1] "handle"         "raw"            "niter"          "evaluation_log"
## [5] "call"           "params"         "callbacks"
##            y_test
## pred_Boost1     0     1
##           0 84643    37
##           1    12   115

## Warning in chisq.test(boostConfusionMatrix1): Chi-squared approximation may
## be incorrect

##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  boostConfusionMatrix1
## X-squared = 57557, df = 1, p-value < 2.2e-16
##
## [1] 0.9994222
## [1] "reg:logistic"
## [1]  train-rmse:0.354602
## [2]  train-rmse:0.256924
## [3]  train-rmse:0.187989
## [4]  train-rmse:0.138422
## [5]  train-rmse:0.102502
## [6]  train-rmse:0.076396
## [7]  train-rmse:0.057473
## [8]  train-rmse:0.043850
## [9]  train-rmse:0.034174
## [10] train-rmse:0.027438
## [1] "handle"         "raw"            "niter"          "evaluation_log"
## [5] "call"           "params"         "callbacks"
##            y_test
## pred_Boost1     0     1
##           0 84646    41
##           1     9   111

## Warning in chisq.test(boostConfusionMatrix1): Chi-squared approximation may
## be incorrect

##
##  Pearson's Chi-squared test with Yates' continuity correction
##
```

```
## data:  boostConfusionMatrix1
## X-squared = 56733, df = 1, p-value < 2.2e-16
##
## [1] 0.9994104
## [1] "binary:logistic"
## [1]  train-error:0.000350
## [2]  train-error:0.000325
## [3]  train-error:0.000330
## [4]  train-error:0.000315
## [5]  train-error:0.000280
## [6]  train-error:0.000265
## [7]  train-error:0.000265
## [8]  train-error:0.000260
## [9]  train-error:0.000255
## [10] train-error:0.000240
## [1] "handle"         "raw"            "niter"          "evaluation_log"
## [5] "call"           "params"         "callbacks"
##            y_test
## pred_Boost1     0     1
##           0 84646    41
##           1     9   111
```

```
## Warning in chisq.test(boostConfusionMatrix1): Chi-squared approximation may
## be incorrect
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  boostConfusionMatrix1
## X-squared = 56733, df = 1, p-value < 2.2e-16
##
## [1] 0.9994104
```

Sampling 20000 data points for training in order to reduce time for the list of algo : KNN, SVM, random forest

```
seqno=seq(0,length(file[,1]))
idx=sample(seqno,20000)
 not_idx=setdiff(seqno,idx)
 train_t=file[idx,]
 test_t=file[not_idx,]
 dim(train_t)
```

```
## [1] 20000    30
```

```
 x_train_t=train_t[,1:29]
 y_train_t=train_t[,"Class"]
 x_test_t=test_t[,1:29]
 y_test_t=test_t[,"Class"]
```

Support Vector Machine

```
SVMmodel1 = svm(x_train_t,y_train_t,type = 'C-classification'
                ,kernel = 'radial')
SVMmodel1
```

```
##
## Call:
## svm.default(x = x_train_t, y = y_train_t, type = "C-classification",
##     kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.03448276
##
## Number of Support Vectors:  670
```

```
pred_svm1=predict(SVMmodel1,newdata = x_test_t)

SVMConfusionMatrix1 = table(pred_svm1,y_test_t)
SVMConfusionMatrix1
```

```
##          y_test_t
## pred_svm1      0      1
##         0 264327    367
##         1     15     98
```

```
SVMChisq1 = chisq.test(SVMConfusionMatrix1)
```

```
## Warning in chisq.test(SVMConfusionMatrix1): Chi-squared approximation may
## be incorrect
```

```
SVMChisq1
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  SVMConfusionMatrix1
## X-squared = 47817, df = 1, p-value < 2.2e-16
```

Metrics

```
library(caret)
pred_svm1 = as.factor(pred_svm1)
```

```r
y_test_t =  as.factor(y_test_t)

SVM_Accuracy1=sum(diag(SVMConfusionMatrix1))/sum(SVMConfusionMatrix1)
print(c("SVM_Accuracy1",SVM_Accuracy1))
```

```
## [1] "SVM_Accuracy1"     "0.998557439946829"
```

```r
svm_precision1 = posPredValue(pred_svm1, y_test_t)
print(c("svm_precision1",svm_precision1))
```

```
## [1] "svm_precision1"    "0.998613493316811"
```

```r
svm_recall1 = sensitivity(pred_probit1, y_test_t)
```

```
## Warning in complete.cases(data) & complete.cases(reference): longer object
## length is not a multiple of shorter object length

## Warning in data %in% positive & reference %in% positive: longer object
## length is not a multiple of shorter object length
```

```r
print(c("svm_recall1",svm_recall1))
```

```
## [1] "svm_recall1"       "0.998918068260057"
```

```r
svm_Spec1 = SVMConfusionMatrix1[2,2]/(SVMConfusionMatrix1[2,2] +
              SVMConfusionMatrix1[1,2])
print(c("svm_Spec1",svm_Spec1))
```

```
## [1] "svm_Spec1"         "0.210752688172043"
```

```r
svm_F1_1 = (2 * svm_precision1 * svm_recall1) / (svm_precision1 +
                                 svm_recall1)
print(c("svm_F1_1",svm_F1_1))
```

```
## [1] "svm_F1_1"          "0.998765757568301"
```

Random Forest

```r
library(randomForest)
```

```
## randomForest 4.6-12

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

##
## Attaching package: 'randomForest'

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

```r
rfModel1 <- randomForest(x_train_t,y_train_t,
                        importance=TRUE,ntree=500)
```

```
## Warning in randomForest.default(x_train_t, y_train_t, importance = TRUE, :
## The response has five or fewer unique values. Are you sure you want to do
## regression?
```

```r
rfModel1
```

```
##
## Call:
##  randomForest(x = x_train_t, y = y_train_t, ntree = 500, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 9
##
##           Mean of squared residuals: 0.0003403886
##                     % Var explained: 74.75
```

```r
pred_rf1 = round(predict(rfModel1,(x_test_t),type="response"),0)
```

```r
rfConfusionMatrix1 = table(pred_rf1,y_test_t)
rfConfusionMatrix1
```

```
##         y_test_t
## pred_rf1      0      1
##        0 264259    121
##        1     83    344
```

ChiSq test

```r
rfChisq1 = chisq.test(rfConfusionMatrix1)
```

```
## Warning in chisq.test(rfConfusionMatrix1): Chi-squared approximation may be
## incorrect
```

```r
rfChisq1
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  rfConfusionMatrix1
## X-squared = 157210, df = 1, p-value < 2.2e-16
```

Metrics

```
library(caret)
pred_rf1 = as.factor(pred_rf1)
y_test_t =  as.factor(y_test_t)

rf_Accuracy1=sum(diag(rfConfusionMatrix1))/sum(rfConfusionMatrix1)
print(c("rf_Accuracy1",rf_Accuracy1))
```

```
## [1] "rf_Accuracy1"      "0.999229627615584"
```

```
rf_precision1 = posPredValue(pred_rf1, y_test_t)
print(c("rf_precision1",rf_precision1))
```

```
## [1] "rf_precision1"     "0.999542325440654"
```

```
rf_recall1 = sensitivity(pred_probit1, y_test_t)
```

```
## Warning in complete.cases(data) & complete.cases(reference): longer object
## length is not a multiple of shorter object length

## Warning in data %in% positive & reference %in% positive: longer object
## length is not a multiple of shorter object length
```

```
print(c("rf_recall1",rf_recall1))
```

```
## [1] "rf_recall1"        "0.998918068260057"
```

```
rf_Spec1 = rfConfusionMatrix1[2,2]/(rfConfusionMatrix1[2,2] +
              rfConfusionMatrix1[1,2])
print(c("rf_Spec1",rf_Spec1))
```

```
## [1] "rf_Spec1"          "0.739784946236559"
```

```
rf_F1_1 = (2 * rf_precision1 * rf_recall1) / (rf_precision1 +
                                ref_recall1)
print(c("rf_F1_1",rf_F1_1))
```

```
## [1] "rf_F1_1"           "0.999230099351043"
```

KNN

```
library(class)
knnModel1 = knn(x_train_t, x_test_t, y_train_t, k = 3,
                  prob = FALSE, use.all = TRUE)

knnConfusionMatrix1 = table(knnModel1,y_test_t)
knnConfusionMatrix1
```

```
##          y_test_t
## knnModel1      0      1
```

```
##         0 264296    244
##         1     46    221
```

Chisq test

```r
knnChisq1 = chisq.test(knnConfusionMatrix1)
```

```
## Warning in chisq.test(knnConfusionMatrix1): Chi-squared approximation may
## be incorrect
```

```r
knnChisq1
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  knnConfusionMatrix1
## X-squared = 103550, df = 1, p-value < 2.2e-16
```

Metrics

```r
library(caret)

knn_Accuracy1=sum(diag(knnConfusionMatrix1))/sum(knnConfusionMatrix1)
print(c("knn_Accuracy1",knn_Accuracy1))
```

```
## [1] "knn_Accuracy1"    "0.99890486278686"
```

```r
knn_precision1 = posPredValue(knnModel1, y_test_t)
print(c("knn_precision1",knn_precision1))
```

```
## [1] "knn_precision1"    "0.999077644212595"
```

```r
knn_recall1 = sensitivity(knnModel1, y_test_t)
print(c("knn_recall1",knn_recall1))
```

```
## [1] "knn_recall1"       "0.999825983006862"
```

```r
knn_Spec1 = knnConfusionMatrix1[2,2]/(knnConfusionMatrix1[2,2] +
              knnConfusionMatrix1[1,2])
print(c("knn_Spec1",knn_Spec1))
```

```
## [1] "knn_Spec1"         "0.475268817204301"
```

```r
knn_F1_1 = (2 * knn_precision1 * knn_recall1) / (knn_precision1 +
                                knn_recall1)
print(c("knn_F1_1",knn_F1_1))
```

```
## [1] "knn_F1_1"          "0.999451673530201"
```

creating data frame

```
algorithms1 = c('Naive Bayes','Linear Discriminant Analysis','Support Vector Machine','KNN',

Accuracy1 = c(BayesAccuracy1,Lda_Accuracy1,SVM_Accuracy1,knn_Accuracy1,
             pb_Accuracy1,lg_Accuracy1,cTree_accuracy1,C4.5_accuracy1,
             C5.0_Accuracy1,rf_Accuracy1)

Precision1 = c(Bayes_precision1,Lda_precision1,svm_precision1,knn_precision1,
              pb_precision1,lg_precision1,cTree_precision1,C4.5_precision1,
              C5.0_precision1,rf_precision1)

Specificity1 = c(Bayes_Spec1,Lda_Spec1,svm_Spec1,knn_Spec1,pb_Spec1,lg_Spec1,
                cTree_Spec1,C4.5_Spec1,C5.0_Spec1,rf_Spec1)

Recall1 = c(Bayes_recall1,Lda_recall1,svm_recall1,knn_recall1,pb_recall1,
           lg_recall1,cTree_recall1,C4.5_recall1,C5.0_recall1,rf_recall1)

F1Score1 = c(Bayes_F1_1,Lda_F1_1,svm_F1_1,knn_F1_1,pb_F1_1,lg_F1_1,cTree_F1_1,
            C4.5_F1_1,C5.0_F1_1,rf_F1_1)
```

populating the dataframe

```
statistics1 = data.frame(algorithms1,Accuracy1,Precision1,Specificity1,
                        Recall1,F1Score1)
statistics1
```

```
##                     algorithms1 Accuracy1 Precision1 Specificity1
## 1                   Naive Bayes 0.9784688  0.9996019    0.7828947
## 2  Linear Discriminant Analysis 0.9994104  0.9995631    0.7565789
## 3        Support Vector Machine 0.9985574  0.9986135    0.2107527
## 4                           KNN 0.9989049  0.9990776    0.4752688
## 5                        Probit 0.9990095  0.9991147    0.5065789
## 6                         Logit 0.9991510  0.9992563    0.5855263
## 7           Classification Tree 0.9993633  0.9995276    0.7368421
## 8                          C4.5 0.9994222  0.9995395    0.7434211
## 9                          C5.0 0.9993868  0.9995630    0.7565789
## 10                Random Forest 0.9992296  0.9995423    0.7397849
##      Recall1  F1Score1
## 1  0.9788199 0.9891018
## 2  0.9998464 0.9997047
## 3  0.9989181 0.9987658
## 4  0.9998260 0.9994517
## 5  0.9998937 0.9995041
## 6  0.9998937 0.9995749
## 7  0.9998346 0.9996811
## 8  0.9998819 0.9997106
```

```
## 9  0.9998228 0.9996929
## 10 0.9989181 0.9992301
```

PCA

```r
cc_train = x_train
head(cc_train)
```

```
##                V1         V2         V3         V4         V5          V6
## 54022   0.6733988 -0.5357197  1.2897471  2.8321018 -0.5631682  1.70274213
## 116469 -0.9393101  0.1466542  1.7052063  1.5088613  0.8860643 -0.40599665
## 209474 -1.5743327 -2.1070156  0.6765146 -0.1702220  2.0938298 -1.21211528
## 128277 -0.8183604  1.0429703  0.6079387  0.7385025  0.2147287 -0.12228971
## 75080  -0.3321915  0.9642144  0.8844730 -0.1707300  0.5065874 -0.05553086
## 230923  0.2310598  1.3449951 -0.9386236  1.3325102  0.7890464 -0.97916866
##                 V7          V8         V9        V10        V11
## 54022  -0.65961499  0.59124090  0.1862881  0.3923875  0.7807621
## 116469  0.02762129  0.18461270 -0.3051010 -0.2580589 -0.7551252
## 209474 -1.23550465  0.26837248 -1.2474921  0.6996593  0.1432209
## 128277  0.71071822  0.28521864 -0.9493856 -0.5887222 -1.4583286
## 75080   0.54850377  0.18577471 -0.1617466 -0.5227185 -0.9449852
## 230923  1.07175424 -0.04942182 -2.0090043  0.5332889 -1.1985766
##                V12        V13        V14        V15         V16
## 54022   1.15157118 -0.3092220 -0.4160618 -1.6518744 -0.33999409
## 116469 -0.22785050 -0.9114778  0.2324321  0.3705841 -1.19955144
## 209474 -0.01049607 -0.7582146  0.3578606 -0.9998794 -2.35341383
## 128277 -0.05534435  0.8096637  0.5018056  1.0524197  0.07730675
## 75080  -0.90796484 -0.9500314 -0.2129764  1.3385769  0.39174771
## 230923  0.28024476  1.4206501  0.3354349 -1.4875345  0.33864888
##                V17        V18        V19         V20         V21
## 54022   0.32651504 -0.6780389 -1.0517678  0.07867677  0.22475588
## 116469  0.71515064 -0.5593137  0.8020653  0.14785723  0.03822154
## 209474  0.45558188  1.7005523  0.6571442  0.15193751  0.07315895
## 128277 -0.39266510  0.4442852  0.8184100  0.18667433  0.16904201
## 75080   0.04969246 -0.1655179 -0.1259222  0.04080272 -0.30288142
## 230923 -0.19063805 -1.1855936 -0.4146082 -0.13352311  0.16825975
##                V22          V23         V24        V25        V26
## 54022   0.67598112 -0.221283791 -0.23795037  0.3774976  0.2385702
## 116469  0.09416356 -0.095969970  0.07950861  0.1265636 -0.1882977
## 209474  0.19728141  0.165432586  0.73406377 -0.0497924  0.8346007
## 128277  0.23930864 -0.145555445 -0.59892424  0.2716507 -0.2177920
## 75080  -0.82758503 -0.065933715 -0.85143717 -0.1603140  0.1575317
## 230923  0.44432583 -0.002921632  0.05270039 -0.5149684  2.3684868
##                V27         V28 Amount
```

```
## 54022   0.04737106  0.03758307 160.20
## 116469  0.11969312  0.12406712   1.00
## 209474  0.03578263  0.17885057  11.50
## 128277 -0.01717166  0.04105559  87.66
## 75080   0.25190828  0.08406908   6.99
## 230923 -0.30801924 -0.11653943  15.73
```

```
cc_train$Amount = (cc_train$Amount - mean(cc_train$Amount))/sd(cc_train$Amount)
head(cc_train)
```

```
##                V1         V2         V3         V4         V5          V6
## 54022   0.6733988 -0.5357197  1.2897471  2.8321018 -0.5631682  1.70274213
## 116469 -0.9393101  0.1466542  1.7052063  1.5088613  0.8860643 -0.40599665
## 209474 -1.5743327 -2.1070156  0.6765146 -0.1702220  2.0938298 -1.21211528
## 128277 -0.8183604  1.0429703  0.6079387  0.7385025  0.2147287 -0.12228971
## 75080  -0.3321915  0.9642144  0.8844730 -0.1707300  0.5065874 -0.05553086
## 230923  0.2310598  1.3449951 -0.9386236  1.3325102  0.7890464 -0.97916866
##                 V7          V8         V9        V10        V11
## 54022  -0.65961499  0.59124090  0.1862881  0.3923875  0.7807621
## 116469  0.02762129  0.18461270 -0.3051010 -0.2580589 -0.7551252
## 209474 -1.23550465  0.26837248 -1.2474921  0.6996593  0.1432209
## 128277  0.71071822  0.28521864 -0.9493856 -0.5887222 -1.4583286
## 75080   0.54850377  0.18577471 -0.1617466 -0.5227185 -0.9449852
## 230923  1.07175424 -0.04942182 -2.0090043  0.5332889 -1.1985766
##                V12        V13        V14        V15         V16
## 54022   1.15157118 -0.3092220 -0.4160618 -1.6518744 -0.33999409
## 116469 -0.22785050 -0.9114778  0.2324321  0.3705841 -1.19955144
## 209474 -0.01049607 -0.7582146  0.3578606 -0.9998794 -2.35341383
## 128277 -0.05534435  0.8096637  0.5018056  1.0524197  0.07730675
## 75080  -0.90796484 -0.9500314 -0.2129764  1.3385769  0.39174771
## 230923  0.28024476  1.4206501  0.3354349 -1.4875345  0.33864888
##                V17        V18        V19         V20         V21
## 54022   0.32651504 -0.6780389 -1.0517678  0.07867677  0.22475588
## 116469  0.71515064 -0.5593137  0.8020653  0.14785723  0.03822154
## 209474  0.45558188  1.7005523  0.6571442  0.15193751  0.07315895
## 128277 -0.39266510  0.4442852  0.8184100  0.18667433  0.16904201
## 75080   0.04969246 -0.1655179 -0.1259222  0.04080272 -0.30288142
## 230923 -0.19063805 -1.1855936 -0.4146082 -0.13352311  0.16825975
##                V22          V23         V24        V25        V26
## 54022   0.67598112 -0.221283791 -0.23795037  0.3774976  0.2385702
## 116469  0.09416356 -0.095969970  0.07950861  0.1265636 -0.1882977
## 209474  0.19728141  0.165432586  0.73406377 -0.0497924  0.8346007
## 128277  0.23930864 -0.145555445 -0.59892424  0.2716507 -0.2177920
## 75080  -0.82758503 -0.065933715 -0.85143717 -0.1603140  0.1575317
## 230923  0.44432583 -0.002921632  0.05270039 -0.5149684  2.3684868
##                V27         V28      Amount
## 54022   0.04737106  0.03758307  0.289179943
```

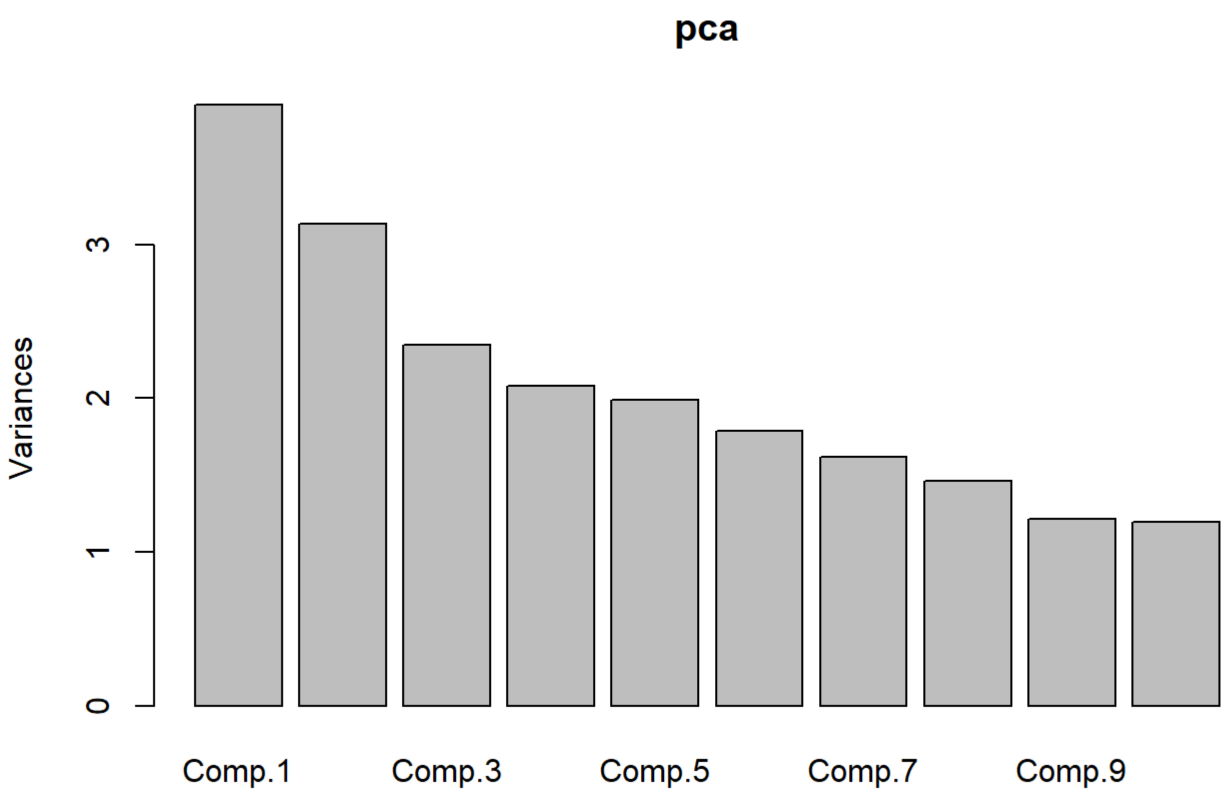
```
## 116469  0.11969312  0.12406712 -0.348877816
## 209474  0.03578263  0.17885057 -0.306794861
## 128277 -0.01717166  0.04105559 -0.001553159
## 75080   0.25190828  0.08406908 -0.324870492
## 230923 -0.30801924 -0.11653943 -0.289841442
```

Principal Component Analysis

```
pca = princomp(cc_train[,1:29])
summary(pca)
```

```
## Importance of components:
##                           Comp.1     Comp.2     Comp.3     Comp.4
## Standard deviation     1.9777268 1.77086124 1.53214836 1.44287896
## Proportion of Variance 0.1232381 0.09880558 0.07396292 0.06559522
## Cumulative Proportion  0.1232381 0.22204368 0.29600660 0.36160182
##                            Comp.5     Comp.6     Comp.7     Comp.8
## Standard deviation     1.41157272 1.33812348 1.27306744 1.20985672
## Proportion of Variance 0.06277965 0.05641632 0.05106405 0.04611904
## Cumulative Proportion  0.42438147 0.48079779 0.53186184 0.57798088
##                            Comp.9    Comp.10    Comp.11    Comp.12
## Standard deviation     1.10146243 1.09244846 1.02127312 0.99957104
## Proportion of Variance 0.03822538 0.03760229 0.03286217 0.03148036
## Cumulative Proportion  0.61620626 0.65380855 0.68667071 0.71815108
##                           Comp.13    Comp.14   Comp.15    Comp.16
## Standard deviation     0.99461064 0.96003254 0.9160977 0.87623542
## Proportion of Variance 0.03116869 0.02903918 0.0264421 0.02419101
## Cumulative Proportion  0.74931977 0.77835895 0.8048010 0.82899206
##                           Comp.17    Comp.18    Comp.19    Comp.20
## Standard deviation     0.84757009 0.84092148 0.82680169 0.81076844
## Proportion of Variance 0.02263412 0.02228042 0.02153848 0.02071124
## Cumulative Proportion  0.85162618 0.87390659 0.89544507 0.91615631
##                           Comp.21   Comp.22    Comp.23    Comp.24
## Standard deviation     0.73977701 0.7261621 0.63168564 0.60570986
## Proportion of Variance 0.01724305 0.0166142 0.01257229 0.01155957
## Cumulative Proportion  0.93339936 0.9500136 0.96258586 0.97414543
##                            Comp.25     Comp.26    Comp.27     Comp.28
## Standard deviation     0.522047999 0.482528426 0.40429174 0.322298502
## Proportion of Variance 0.008586838 0.007335981 0.00514994 0.003272872
## Cumulative Proportion  0.982732264 0.990068245 0.99521819 0.998491057
##                            Comp.29
## Standard deviation     0.218841799
## Proportion of Variance 0.001508943
## Cumulative Proportion  1.000000000
```

```r
screeplot(pca)
```




```r
pca$loadings
```

```
##
## Loadings:
##        Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9
## V1      0.962  0.237
## V2      0.134 -0.825  0.328  0.296               -0.113
## V3            -0.159 -0.902  0.363
## V4                           0.371  0.920
## V5            -0.186 -0.186 -0.700  0.363  0.389 -0.287
## V6                           0.232         0.904  0.276
## V7             0.136         0.218         0.130 -0.895
## V8                                                      -0.990
## V9                                                              0.971
```

```
## V10                                                             0.222
## V11
## V12
## V13
## V14
## V15
## V16
## V17
## V18
## V19
## V20
## V21
## V22
## V23
## V24
## V25
## V26
## V27
## V28
## Amount -0.216  0.413  0.137  0.210               -0.135
##        Comp.10 Comp.11 Comp.12 Comp.13 Comp.14 Comp.15 Comp.16 Comp.17
## V1
## V2
## V3
## V4
## V5
## V6
## V7
## V8
## V9      0.230
## V10    -0.967
## V11             0.999
## V12                     0.990  -0.140
## V13                    -0.139  -0.989
## V14                                    -0.998
## V15                                            -0.999
## V16                                                    -0.999
## V17                                                            -0.979
## V18                                                             0.198
## V19
## V20
## V21
## V22
## V23
## V24
## V25
```

```
## V26
## V27
## V28
## Amount
##        Comp.18 Comp.19 Comp.20 Comp.21 Comp.22 Comp.23 Comp.24 Comp.25
## V1
## V2             -0.108
## V3
## V4
## V5             -0.100
## V6
## V7              0.130
## V8
## V9
## V10
## V11
## V12
## V13
## V14
## V15
## V16
## V17     0.195
## V18     0.912   0.348
## V19    -0.109   0.485  -0.865
## V20     0.288  -0.710  -0.463   0.203           0.107
## V21            -0.123          -0.969  -0.145
## V22                             0.124  -0.986
## V23                                             0.984
## V24                                                    -0.999
## V25                                                            -0.996
## V26
## V27
## V28
## Amount  0.130  -0.253  -0.112
##        Comp.26 Comp.27 Comp.28 Comp.29
## V1
## V2                             -0.250
## V3                             -0.109
## V4
## V5                             -0.221
## V6                              0.128
## V7                              0.255
## V8
## V9
## V10
## V11
```

```
## V12
## V13
## V14
## V15
## V16
## V17
## V18
## V19
## V20                             0.362
## V21                             0.120
## V22
## V23                            -0.157
## V24
## V25
## V26     1.000
## V27            -0.997
## V28                   -0.999
## Amount                         -0.760
##
##                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## SS loadings     1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
## Proportion Var  0.034  0.034  0.034  0.034  0.034  0.034  0.034  0.034
## Cumulative Var  0.034  0.069  0.103  0.138  0.172  0.207  0.241  0.276
##                Comp.9 Comp.10 Comp.11 Comp.12 Comp.13 Comp.14 Comp.15
## SS loadings     1.000   1.000   1.000   1.000   1.000   1.000   1.000
## Proportion Var  0.034   0.034   0.034   0.034   0.034   0.034   0.034
## Cumulative Var  0.310   0.345   0.379   0.414   0.448   0.483   0.517
##                Comp.16 Comp.17 Comp.18 Comp.19 Comp.20 Comp.21 Comp.22
## SS loadings      1.000   1.000   1.000   1.000   1.000   1.000   1.000
## Proportion Var   0.034   0.034   0.034   0.034   0.034   0.034   0.034
## Cumulative Var   0.552   0.586   0.621   0.655   0.690   0.724   0.759
##                Comp.23 Comp.24 Comp.25 Comp.26 Comp.27 Comp.28 Comp.29
## SS loadings      1.000   1.000   1.000   1.000   1.000   1.000   1.000
## Proportion Var   0.034   0.034   0.034   0.034   0.034   0.034   0.034
## Cumulative Var   0.793   0.828   0.862   0.897   0.931   0.966   1.000
```

Taking a subset of 15 variables on the basis of PCA to get 80% of the total proportion

```
x_train_15 = x_train[1:15]
x_test_15  = x_test[1:15]
```

Naive Bayes

```r
library(e1071)
bayesModel2 = naiveBayes(as.matrix(x_train_15),factor(y_train))

bayesModel2
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = as.matrix(x_train_15), y = factor(y_train))
##
## A-priori probabilities:
## factor(y_train)
##      0      1
## 0.9983 0.0017
##
## Conditional probabilities:
##                V1
## factor(y_train)        [,1]     [,2]
##               0  0.01222851 1.925536
##               1 -4.70331334 6.702104
##
##                V2
## factor(y_train)         [,1]     [,2]
##               0 -0.006351483 1.626194
##               1  3.813661934 4.220590
##
##                V3
## factor(y_train)        [,1]     [,2]
##               0  0.01227501 1.459395
##               1 -7.06725446 6.931178
##
##                V4
## factor(y_train)         [,1]     [,2]
##               0 -0.007834009 1.397836
##               1  4.489122088 2.854061
##
##                V5
## factor(y_train)         [,1]     [,2]
##               0  0.004971739 1.360567
##               1 -3.074967825 5.337522
##
##                V6
## factor(y_train)         [,1]     [,2]
```

```
##                0  0.001162553 1.330726
##                1 -1.376410346 1.948252
##
##                 V7
## factor(y_train)         [,1]     [,2]
##                0  0.008871949 1.187548
##                1 -5.618393748 7.367950
##
##                 V8
## factor(y_train)        [,1]     [,2]
##                0 -0.00087658 1.167538
##                1  0.23433738 7.571570
##
##                 V9
## factor(y_train)         [,1]     [,2]
##                0  0.004905328 1.091622
##                1 -2.585144950 2.497433
##
##                 V10
## factor(y_train)        [,1]     [,2]
##                0  0.01077754 1.046344
##                1 -5.75949610 4.871139
##
##                 V11
## factor(y_train)         [,1]     [,2]
##                0 -0.005702064 1.003800
##                1  3.887435382 2.585502
##
##                 V12
## factor(y_train)        [,1]      [,2]
##                0  0.01083298 0.9469475
##                1 -6.31149802 4.5968065
##
##                 V13
## factor(y_train)         [,1]      [,2]
##                0  0.002217953 0.9944846
##                1 -0.169449586 1.1332543
##
##                 V14
## factor(y_train)        [,1]     [,2]
##                0  0.01100557 0.897034
##                1 -7.14715373 4.246473
##
##                 V15
## factor(y_train)          [,1]      [,2]
##                0  0.0004409582 0.9157867
```

```
##               1 -0.1233307319 1.0624572
```

```
pred_Bayes2=predict(bayesModel2,newdata = as.matrix(x_test_15))
bayesConfusionMatrix2 = table(pred_Bayes2,factor(y_test))

bayesConfusionMatrix2
```

```
##
## pred_Bayes2     0     1
##           0 83266    34
##           1  1389   118
```

Chisq test

```
BayesChiSq2 = chisq.test(bayesConfusionMatrix2)
```

```
## Warning in chisq.test(bayesConfusionMatrix2): Chi-squared approximation may
## be incorrect
```

```
BayesChiSq2
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  bayesConfusionMatrix2
## X-squared = 4976.4, df = 1, p-value < 2.2e-16
```

Metrics

```
library(caret)
pred_Bayes2 = as.factor(pred_Bayes2)
y_test =  as.factor(y_test)

BayesAccuracy2=sum(diag(bayesConfusionMatrix2))/sum(bayesConfusionMatrix2)
print(c("BayesAccuracy2",BayesAccuracy2))
```

```
## [1] "BayesAccuracy2"    "0.983220724704329"
```

```
Bayes_precision2 = posPredValue(pred_Bayes2, y_test)
print(c("Bayes_precision2",Bayes_precision2))
```

```
## [1] "Bayes_precision2"  "0.999591836734694"
```

```
Bayes_recall2 = sensitivity(pred_Bayes2, y_test)
print(c("Bayes_recall2",Bayes_recall2))
```

```
## [1] "Bayes_recall2"     "0.983592227275412"
```

```r
Bayes_Spec2 = bayesConfusionMatrix2[2,2]/(bayesConfusionMatrix2[2,2] +
              bayesConfusionMatrix2[1,2])
print(c("Bayes_Spec2",Bayes_Spec2))
```

```
## [1] "Bayes_Spec2"       "0.776315789473684"
```

```r
Bayes_F1_2 = (2 * Bayes_precision2 * Bayes_recall2) /
                 (Bayes_precision2 + Bayes_recall2)
print(c("Bayes_F1_2",Bayes_F1_2))
```

```
## [1] "Bayes_F1_2"        "0.991527492483106"
```

Linear discriminant analysis

```r
library(MASS)

x_train1_15 = x_train_15
ldaModel2 = lda((y_train)~as.matrix(x_train1_15))
x_train1_15 = x_test_15

ldaModel2
```

```
## Call:
## lda((y_train) ~ as.matrix(x_train1_15))
##
## Prior probabilities of groups:
##      0      1
## 0.9983 0.0017
##
## Group means:
##   as.matrix(x_train1_15)V1 as.matrix(x_train1_15)V2
## 0               0.01222851             -0.006351483
## 1              -4.70331334              3.813661934
##   as.matrix(x_train1_15)V3 as.matrix(x_train1_15)V4
## 0               0.01227501             -0.007834009
## 1              -7.06725446              4.489122088
##   as.matrix(x_train1_15)V5 as.matrix(x_train1_15)V6
## 0              0.004971739              0.001162553
## 1             -3.074967825             -1.376410346
##   as.matrix(x_train1_15)V7 as.matrix(x_train1_15)V8
## 0              0.008871949              -0.00087658
## 1             -5.618393748               0.23433738
##   as.matrix(x_train1_15)V9 as.matrix(x_train1_15)V10
## 0              0.004905328                0.01077754
## 1             -2.585144950               -5.75949610
##   as.matrix(x_train1_15)V11 as.matrix(x_train1_15)V12
```

```
## 0              -0.005702064               0.01083298
## 1               3.887435382              -6.31149802
##   as.matrix(x_train1_15)V13 as.matrix(x_train1_15)V14
## 0               0.002217953               0.01100557
## 1              -0.169449586              -7.14715373
##   as.matrix(x_train1_15)V15
## 0              0.0004409582
## 1             -0.1233307319
##
## Coefficients of linear discriminants:
##                                   LD1
## as.matrix(x_train1_15)V1  -0.10579577
## as.matrix(x_train1_15)V2   0.11696603
## as.matrix(x_train1_15)V3  -0.26265688
## as.matrix(x_train1_15)V4   0.19205367
## as.matrix(x_train1_15)V5  -0.13880974
## as.matrix(x_train1_15)V6  -0.06791846
## as.matrix(x_train1_15)V7  -0.31078450
## as.matrix(x_train1_15)V8   0.02407633
## as.matrix(x_train1_15)V9  -0.18284256
## as.matrix(x_train1_15)V10 -0.41217021
## as.matrix(x_train1_15)V11  0.31912927
## as.matrix(x_train1_15)V12 -0.54060020
## as.matrix(x_train1_15)V13 -0.01072884
## as.matrix(x_train1_15)V14 -0.66181044
## as.matrix(x_train1_15)V15 -0.01143533
```

Confusion Matrix

```
pred_Lda2 = predict(ldaModel2,newdata = (x_train1_15))$class
ldaConfusionMatrix2 = table(pred_Lda2,y_test)
ldaConfusionMatrix2
```

```
##          y_test
## pred_Lda2     0     1
##         0 84637    47
##         1    18   105
```

Metrics

```
library(caret)
pred_Lda2 = as.factor(pred_Lda2)
y_test =  as.factor(y_test)
```

```r
Lda_Accuracy2=sum(diag(ldaConfusionMatrix2))/sum(ldaConfusionMatrix2)
print(c("Lda_Accuracy2",Lda_Accuracy2))
```

```
## [1] "Lda_Accuracy2"  "0.999233553834"
```

```r
Lda_precision2 = posPredValue(pred_Lda2, y_test)
print(c("Lda_precision2",Lda_precision2))
```

```
## [1] "Lda_precision2"   "0.99944499551273"
```

```r
Lda_recall2 = sensitivity(pred_Lda2, y_test)
print(c("Lda_recall2",Lda_recall2))
```

```
## [1] "Lda_recall2"       "0.999787372275707"
```

```r
Lda_Spec2 = ldaConfusionMatrix2[2,2]/(ldaConfusionMatrix2[2,2] +
              ldaConfusionMatrix2[1,2])
print(c("Lda_Spec2",Lda_Spec2))
```

```
## [1] "Lda_Spec2"         "0.690789473684211"
```

```r
Lda_F1_2 = (2 * Lda_precision2 * Lda_recall2) / (Lda_precision2 +
                                Lda_recall2)
print(c("Lda_F1_2",Lda_F1_2))
```

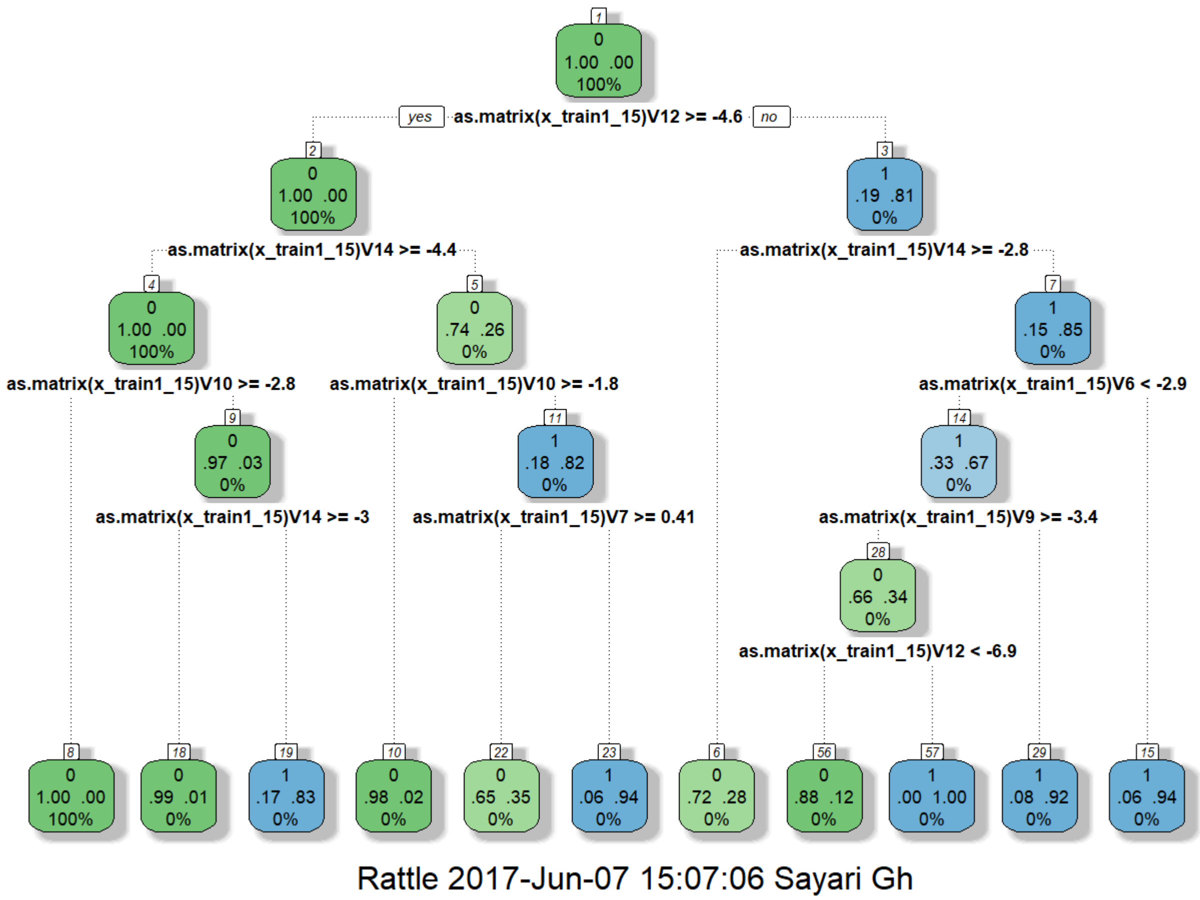
```
## [1] "Lda_F1_2"          "0.999616154577504"
```

Classification

```r
library(rpart)
library(rattle)
library(rpart.plot)
library(RColorBrewer)

x_train1_15 = x_train_15
cTreeModel2=rpart(factor(y_train)~as.matrix(x_train1_15),method = 'class')

fancyRpartPlot(cTreeModel2)
```



Confusion Matrix

```
x_train1_15=x_test_15
pred_CTree2 = round(predict(cTreeModel2,x_train1_15))
pred_CTree2=pred_CTree2[,2]
cTreeConfusionMatrix2 = table(pred_CTree2,y_test)
cTreeConfusionMatrix2
```

```
##              y_test
## pred_CTree2     0     1
##           0 84646    41
##           1     9   111
```

Chisq Test

```
cTreeChisq2 = chisq.test(cTreeConfusionMatrix2)
```

```
## Warning in chisq.test(cTreeConfusionMatrix2): Chi-squared approximation may
## be incorrect
```

```
cTreeChisq2
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  cTreeConfusionMatrix2
## X-squared = 56733, df = 1, p-value < 2.2e-16
```

Metrics

```
library(caret)
pred_CTree2 = as.factor(pred_CTree2)
y_test =  as.factor(y_test)

cTree_accuracy2=sum(diag(cTreeConfusionMatrix2))/sum(cTreeConfusionMatrix2)
print(c("cTree_accuracy2",cTree_accuracy2))
```

```
## [1] "cTree_accuracy2"   "0.999410426026153"
```

```
cTree_precision2 = posPredValue(pred_CTree2, y_test)
print(c("cTree_precision2",cTree_precision2))
```

```
## [1] "cTree_precision2"  "0.999515864300306"
```

```
cTree_recall2 = sensitivity(pred_CTree2, y_test)
print(c("cTree_recall2",cTree_recall2))
```

```
## [1] "cTree_recall2"     "0.999893686137854"
```

```
cTree_Spec2 = cTreeConfusionMatrix2[2,2]/(cTreeConfusionMatrix2[2,2] +
              cTreeConfusionMatrix2[1,2])
print(c("cTree_Spec2",cTree_Spec2))
```

```
## [1] "cTree_Spec2"       "0.730263157894737"
```

```
cTree_F1_2 = (2 * cTree_precision2 * cTree_recall2) / (cTree_precision2 +
                                cTree_recall2)
print(c("cTree_F1_2",cTree_F1_2))
```

```
## [1] "cTree_F1_2"        "0.999704739521206"
```

C4.5

```r
library(RWeka)

C4.5Model2 =J48(factor(y_train)~.,x_train_15,
        control = Weka_control(), options = NULL)
summary(C4.5Model2)
```

```
##
## === Summary ===
##
## Correctly Classified Instances      199939               99.9695 %
## Incorrectly Classified Instances       61                0.0305 %
## Kappa statistic                          0.9032
## Mean absolute error                      0.0006
## Root mean squared error                  0.0174
## Relative absolute error                 17.7187 %
## Root relative squared error             42.1244 %
## Total Number of Instances           200000
##
## === Confusion Matrix ===
##
##       a      b   <-- classified as
##  199654      6 |      a = 0
##      55    285 |      b = 1
```

Confusion Matrix

```r
pred_C4.5_2 = predict(C4.5Model2,x_test_15)
C4.5ConfusionMatrix2 = table(pred_C4.5_2,y_test)
C4.5ConfusionMatrix2
```

```
##            y_test
## pred_C4.5_2     0     1
##           0 84648    47
##           1     7   105
```

Chisq Test

```r
C4.5Chisq2 = chisq.test(C4.5ConfusionMatrix2)
```

```
## Warning in chisq.test(C4.5ConfusionMatrix2): Chi-squared approximation may
## be incorrect
```

```
C4.5Chisq2
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  C4.5ConfusionMatrix2
## X-squared = 54361, df = 1, p-value < 2.2e-16
```

Metrics

```
library(caret)
pred_C4.5_2 = as.factor(pred_C4.5_2)
y_test =  as.factor(y_test)

C4.5_accuracy2=sum(diag(C4.5ConfusionMatrix2))/sum(C4.5ConfusionMatrix2)
print(c("C4.5_accuracy2",C4.5_accuracy2))
```

```
## [1] "C4.5_accuracy2"    "0.999363260108246"
```

```
C4.5_precision2 = posPredValue(pred_C4.5_2, y_test)
print(c("C4.5_precision2",C4.5_precision2))
```

```
## [1] "C4.5_precision2"  "0.99944506759549"
```

```
C4.5_recall2 = sensitivity(pred_C4.5_2, y_test)
print(c("C4.5_recall2",C4.5_recall2))
```

```
## [1] "C4.5_recall2"      "0.999917311440553"
```

```
C4.5_Spec2 = C4.5ConfusionMatrix2[2,2]/(C4.5ConfusionMatrix2[2,2] +
              C4.5ConfusionMatrix2[1,2])
print(c("C4.5_Spec2",C4.5_Spec2))
```

```
## [1] "C4.5_Spec2"        "0.690789473684211"
```

```
C4.5_F1_2 = (2 * C4.5_precision2 * C4.5_recall2) / (C4.5_precision2 +
                                C4.5_recall2)
print(c("C4.5_F1_2",C4.5_F1_2))
```

```
## [1] "C4.5_F1_2"         "0.999681133746678"
```

C5.0

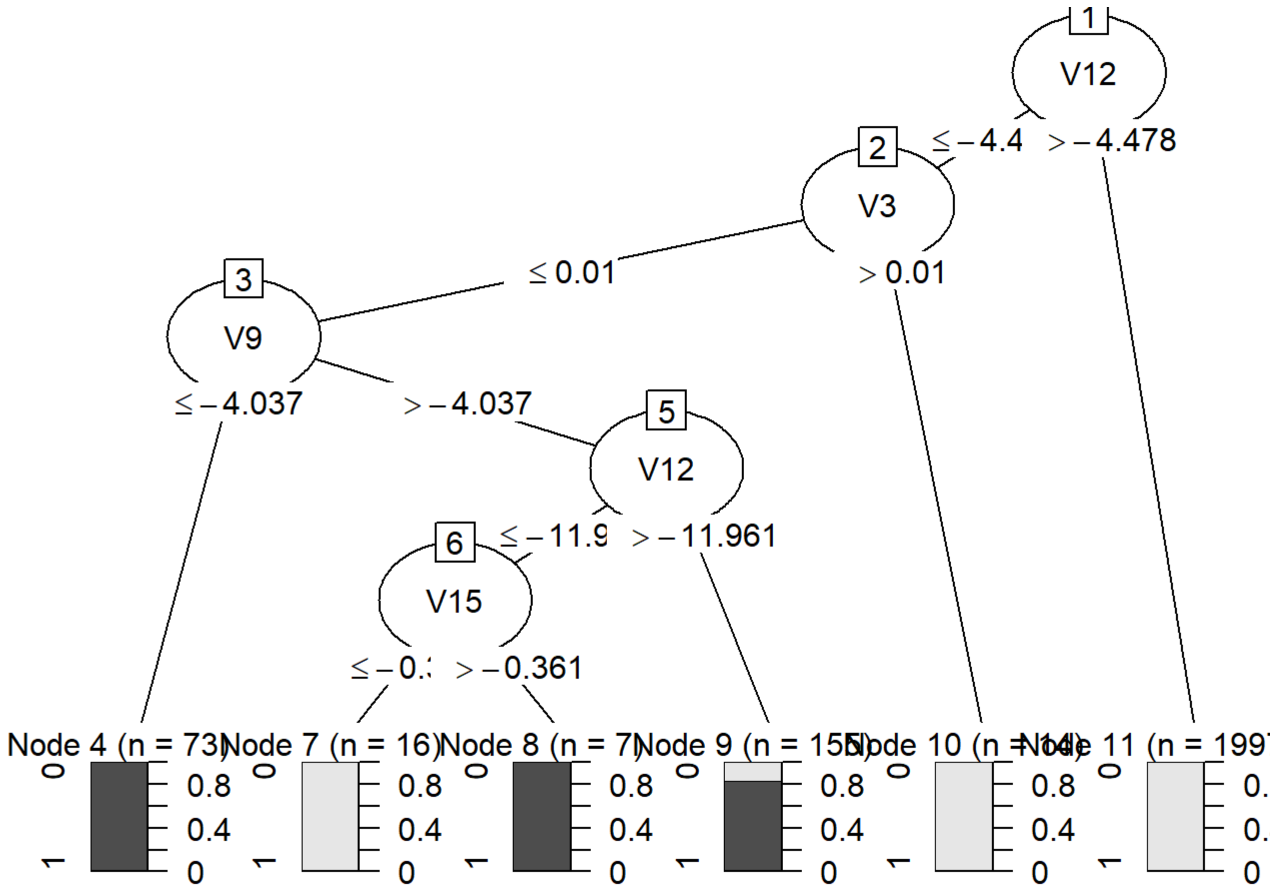
```
library(C50)

C5.0Model2 =C5.0(x_train_15,
        factor(y_train),
```

```r
       trials=20,rules = FALSE)
C5.0Model2
```

```
##
## Call:
## C5.0.default(x = x_train_15, y = factor(y_train), trials = 20, rules
##  = FALSE)
##
## Classification Tree
## Number of samples: 200000
## Number of predictors: 15
##
## Number of boosting iterations: 20
## Average tree size: 6.8
##
## Non-standard options: attempt to group attributes
```

```r
plot(C5.0Model2)
```



Confusion Matrix

```
pred_C5.0_2 = predict(C5.0Model2,x_test_15)
C5.0ConfusionMatrix2 = table(pred_C5.0_2,y_test)
C5.0ConfusionMatrix2
```

```
##              y_test
## pred_C5.0_2      0      1
##            0 84642     45
##            1    13    107
```

Chisq Test

```
C5.0Chisq2 = chisq.test(C5.0ConfusionMatrix2)
```

```
## Warning in chisq.test(C5.0ConfusionMatrix2): Chi-squared approximation may
## be incorrect
```

```
C5.0Chisq2
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  C5.0ConfusionMatrix2
## X-squared = 52692, df = 1, p-value < 2.2e-16
```

Metrics

```
library(caret)
pred_C5.0_2 = as.factor(pred_C5.0_2)
y_test =  as.factor(y_test)

C5.0_Accuracy2=sum(diag(C5.0ConfusionMatrix2))/sum(C5.0ConfusionMatrix2)
print(c("C5.0_Accuracy2",C5.0_Accuracy2))
```

```
## [1] "C5.0_Accuracy2"    "0.999316094190338"
```

```
C5.0_precision2 = posPredValue(pred_C5.0_2, y_test)
print(c("C5.0_precision2",C5.0_precision2))
```

```
## [1] "C5.0_precision2"   "0.999468631549116"
```

```
C5.0_recall2 = sensitivity(pred_C5.0_2, y_test)
print(c("C5.0_recall2",C5.0_recall2))
```

```
## [1] "C5.0_recall2"      "0.999846435532455"
```

```
C5.0_Spec2 = C5.0ConfusionMatrix2[2,2]/(C5.0ConfusionMatrix2[2,2] +
              C5.0ConfusionMatrix2[1,2])
print(c("C5.0_Spec2",C5.0_Spec2))
```

```
## [1] "C5.0_Spec2"        "0.703947368421053"
```

```
C5.0_F1_2 = (2 * C5.0_precision2 * C5.0_recall2) / (C5.0_precision2 +
                                C5.0_recall2)
print(c("C5.0_F1_2",C5.0_F1_2))
```

```
## [1] "C5.0_F1_2"         "0.999657497844598"
```

Logit

```
x_train1_15=x_train_15
logitModel2 = glm(y_train~as.matrix(x_train1_15),
                 family=binomial(link="logit"))
x_train1_15=x_test_15
```

```
logitModel2
```

```
##
## Call:  glm(formula = y_train ~ as.matrix(x_train1_15), family = binomial(link = "logit"))
##
## Coefficients:
##              (Intercept)   as.matrix(x_train1_15)V1
##                -8.274314                  -0.057639
##  as.matrix(x_train1_15)V2   as.matrix(x_train1_15)V3
##                 0.001914                   0.141343
##  as.matrix(x_train1_15)V4   as.matrix(x_train1_15)V5
##                 0.431584                   0.006730
##  as.matrix(x_train1_15)V6   as.matrix(x_train1_15)V7
##                -0.051143                  -0.037858
##  as.matrix(x_train1_15)V8   as.matrix(x_train1_15)V9
##                -0.153047                  -0.007118
## as.matrix(x_train1_15)V10  as.matrix(x_train1_15)V11
##                -0.387008                   0.150794
## as.matrix(x_train1_15)V12  as.matrix(x_train1_15)V13
##                -0.242899                  -0.276545
## as.matrix(x_train1_15)V14  as.matrix(x_train1_15)V15
##                -0.691250                  -0.033000
##
## Degrees of Freedom: 199999 Total (i.e. Null);  199984 Residual
## Null Deviance:       5016
## Residual Deviance: 1608  AIC: 1640
```

```
pred_logit2=round(predict(logitModel2,x_train1_15,type="response"),0)
logitConfusionMatrix2 = table(pred_logit2,y_test)
logitConfusionMatrix2
```

```
##            y_test
## pred_logit2     0     1
##           0 84643    68
##           1    12    84
```

```
lgChisq2 = chisq.test(logitConfusionMatrix2)
```

```
## Warning in chisq.test(logitConfusionMatrix2): Chi-squared approximation may
## be incorrect
```

```
lgChisq2
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  logitConfusionMatrix2
## X-squared = 40473, df = 1, p-value < 2.2e-16
```

Metrics

```
library(caret)
pred_logit2 = as.factor(pred_logit2)
y_test =  as.factor(y_test)

lg_Accuracy2=sum(diag(logitConfusionMatrix2))/sum(logitConfusionMatrix2)
print(c("lg_Accuracy2",lg_Accuracy2))
```

```
## [1] "lg_Accuracy2"      "0.999056681641846"
```

```
lg_precision2 = posPredValue(pred_logit2, y_test)
print(c("lg_precision2",lg_precision2))
```

```
## [1] "lg_precision2"    "0.99919727072045"
```

```
lg_recall2 = sensitivity(pred_logit2, y_test)
print(c("lg_recall2",lg_recall2))
```

```
## [1] "lg_recall2"        "0.999858248183805"
```

```
lg_Spec2 = logitConfusionMatrix2[2,2]/(logitConfusionMatrix2[2,2] +
              logitConfusionMatrix2[1,2])
print(c("lg_Spec2",lg_Spec2))
```

```
## [1] "lg_Spec2"          "0.552631578947368"
```

```
lg_F1_2 = (2 * lg_precision2 * lg_recall2) / (lg_precision2 +
                                 lg_recall2)
print(c("lg_F1_2",lg_F1_2))
```

```
## [1] "lg_F1_2"           "0.999527650177722"
```

Probit

```
x_train1_15=x_train_15
probitModel2 = glm(y_train~as.matrix(x_train1_15),
                  family=binomial(link="probit"))
```

```
## Warning: glm.fit: algorithm did not converge
```

```r
x_train1_15=x_test_15
probitModel2
```

```
##
## Call:  glm(formula = y_train ~ as.matrix(x_train1_15), family = binomial(link = "probit")
##
## Coefficients:
##              (Intercept)   as.matrix(x_train1_15)V1
##                -3.618776                  -0.019850
##  as.matrix(x_train1_15)V2   as.matrix(x_train1_15)V3
##                 0.004864                   0.024803
##  as.matrix(x_train1_15)V4   as.matrix(x_train1_15)V5
##                 0.166056                   0.012633
##  as.matrix(x_train1_15)V6   as.matrix(x_train1_15)V7
##                -0.030683                  -0.003240
##  as.matrix(x_train1_15)V8   as.matrix(x_train1_15)V9
##                -0.068070                  -0.056438
## as.matrix(x_train1_15)V10  as.matrix(x_train1_15)V11
##                -0.116532                   0.031832
## as.matrix(x_train1_15)V12  as.matrix(x_train1_15)V13
##                -0.084283                  -0.110288
## as.matrix(x_train1_15)V14  as.matrix(x_train1_15)V15
##                -0.270502                  -0.010057
##
## Degrees of Freedom: 199999 Total (i.e. Null);  199984 Residual
## Null Deviance:      5016
## Residual Deviance: 1532  AIC: 1564
```

```r
pred_probit2=round(predict(probitModel2,x_train1_15,type="response"),0)
probitConfusionMatrix2 = table(pred_probit2,y_test)
probitConfusionMatrix2
```

```
##             y_test
## pred_probit2     0     1
##            0 84647    78
##            1     8    74
```

```r
pbChisq2 = chisq.test(probitConfusionMatrix2)
```

```
## Warning in chisq.test(probitConfusionMatrix2): Chi-squared approximation
## may be incorrect
```

```r
pbChisq2
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  probitConfusionMatrix2
## X-squared = 36712, df = 1, p-value < 2.2e-16
```

Metrics

```
library(caret)
pred_probit2 = as.factor(pred_probit2)
y_test =  as.factor(y_test)

pb_Accuracy2=sum(diag(probitConfusionMatrix2))/sum(probitConfusionMatrix2)
print(c("pb_Accuracy2",pb_Accuracy2))
```

```
## [1] "pb_Accuracy2"      "0.998985932764984"
```

```
pb_precision2 = posPredValue(pred_probit2, y_test)
print(c("pb_precision2",pb_precision2))
```

```
## [1] "pb_precision2"     "0.999079374446739"
```

```
pb_recall2 = sensitivity(pred_probit2, y_test)
print(c("pb_recall2",pb_recall2))
```

```
## [1] "pb_recall2"        "0.999905498789203"
```

```
pb_Spec2 = probitConfusionMatrix2[2,2]/(probitConfusionMatrix2[2,2] +
              probitConfusionMatrix2[1,2])
print(c("pb_Spec2",pb_Spec2))
```

```
## [1] "pb_Spec2"          "0.486842105263158"
```

```
pb_F1_2 = (2 * pb_precision2 * pb_recall2) / (pb_precision2 +
                                pb_recall2)
print(c("pb_F1_2",pb_F1_2))
```

```
## [1] "pb_F1_2"           "0.999492265910969"
```

creating data frame

```
algorithms2 = c('Naive Bayes','Linear Discriminant Analysis',
               'Probit','Logit','Classification Tree','C4.5','C5.0')

Accuracy2 = c(BayesAccuracy2,Lda_Accuracy2,pb_Accuracy2,lg_Accuracy2,
             cTree_accuracy2,C4.5_accuracy2,C5.0_Accuracy2)

Precision2 = c(Bayes_precision2,Lda_precision2,pb_precision2,lg_precision2,
              cTree_precision2,C4.5_precision2,C5.0_precision2)

Specificity2 = c(Bayes_Spec2,Lda_Spec2,pb_Spec2,lg_Spec2,
                cTree_Spec2,C4.5_Spec2,C5.0_Spec2)

Recall2 = c(Bayes_recall2,Lda_recall2,pb_recall2,
```

```
          lg_recall2,cTree_recall2,C4.5_recall2,C5.0_recall2)

F1Score2 = c(Bayes_F1_2,Lda_F1_2,pb_F1_2,lg_F1_2,cTree_F1_2,
            C4.5_F1_2,C5.0_F1_2)
```

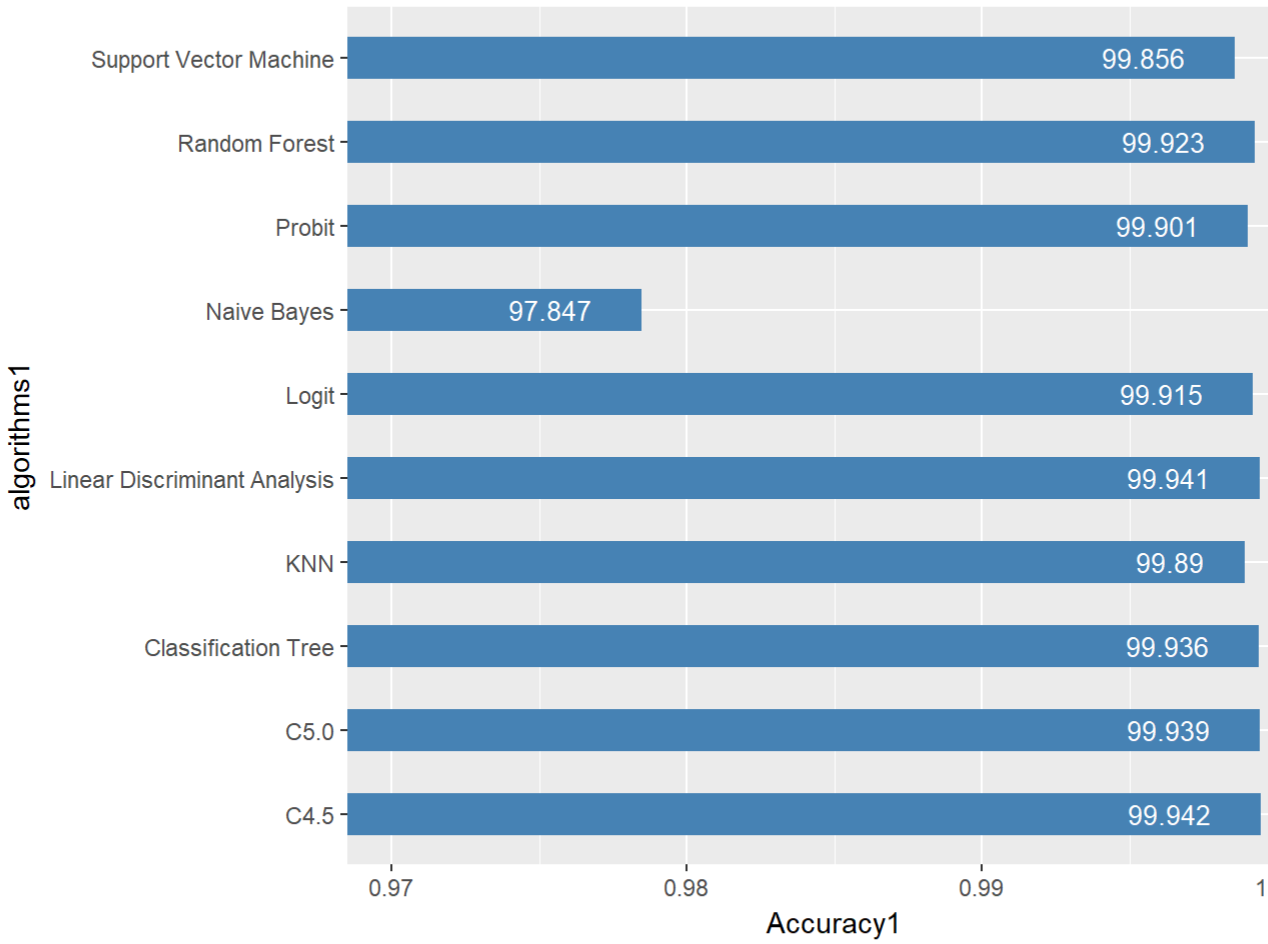
populating the dataframe

```
statistics2 = data.frame(algorithms2,Accuracy2,Precision2,Specificity2,
                        Recall2,F1Score2)
statistics2
```

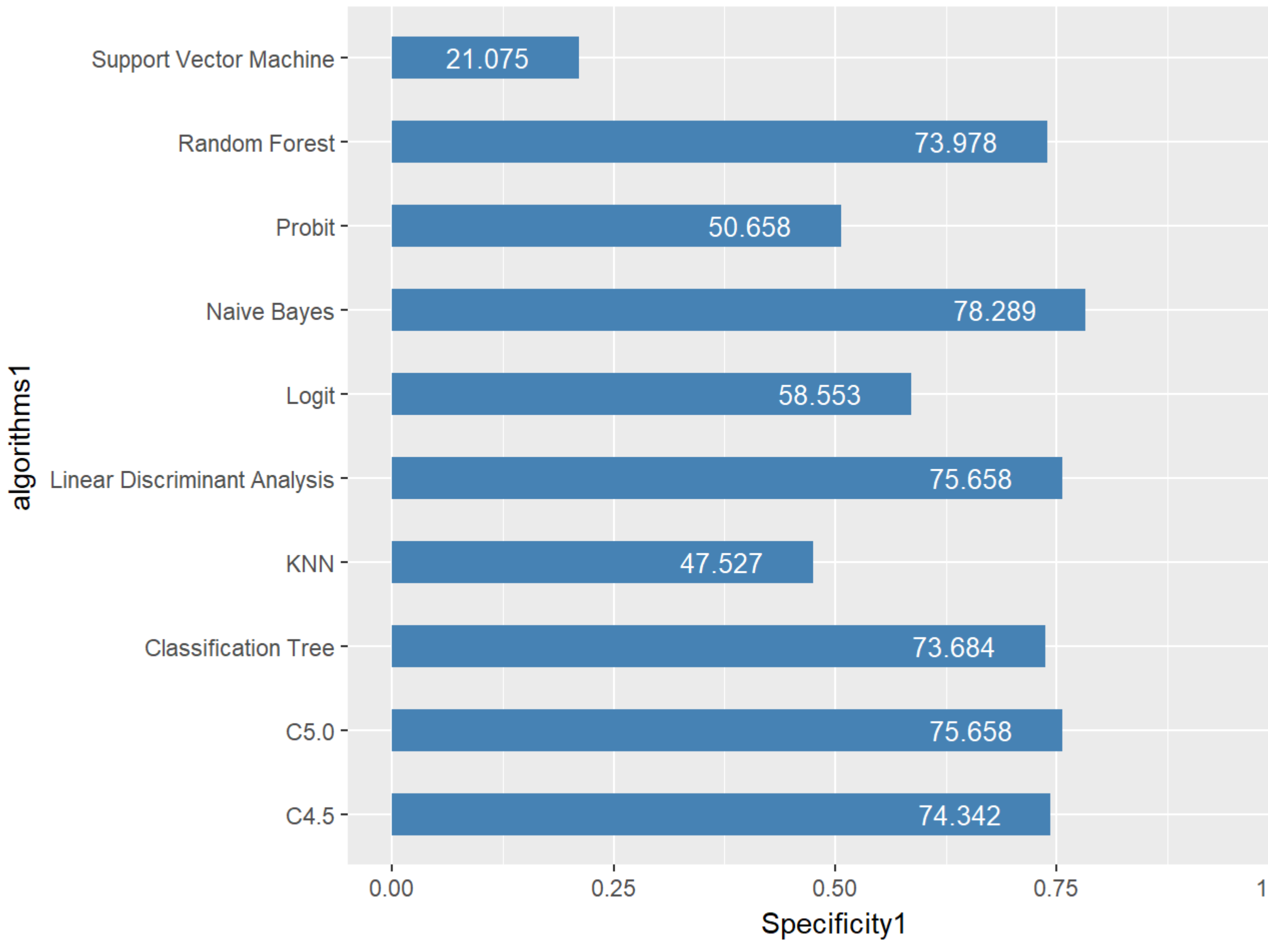
```
##                    algorithms2 Accuracy2 Precision2 Specificity2   Recall2
## 1                  Naive Bayes 0.9832207  0.9995918    0.7763158 0.9835922
## 2 Linear Discriminant Analysis 0.9992336  0.9994450    0.6907895 0.9997874
## 3                       Probit 0.9989859  0.9990794    0.4868421 0.9999055
## 4                        Logit 0.9990567  0.9991973    0.5526316 0.9998582
## 5          Classification Tree 0.9994104  0.9995159    0.7302632 0.9998937
## 6                         C4.5 0.9993633  0.9994451    0.6907895 0.9999173
## 7                         C5.0 0.9993161  0.9994686    0.7039474 0.9998464
##    F1Score2
## 1 0.9915275
## 2 0.9996162
## 3 0.9994923
## 4 0.9995277
## 5 0.9997047
## 6 0.9996811
## 7 0.9996575
```

Before PCA

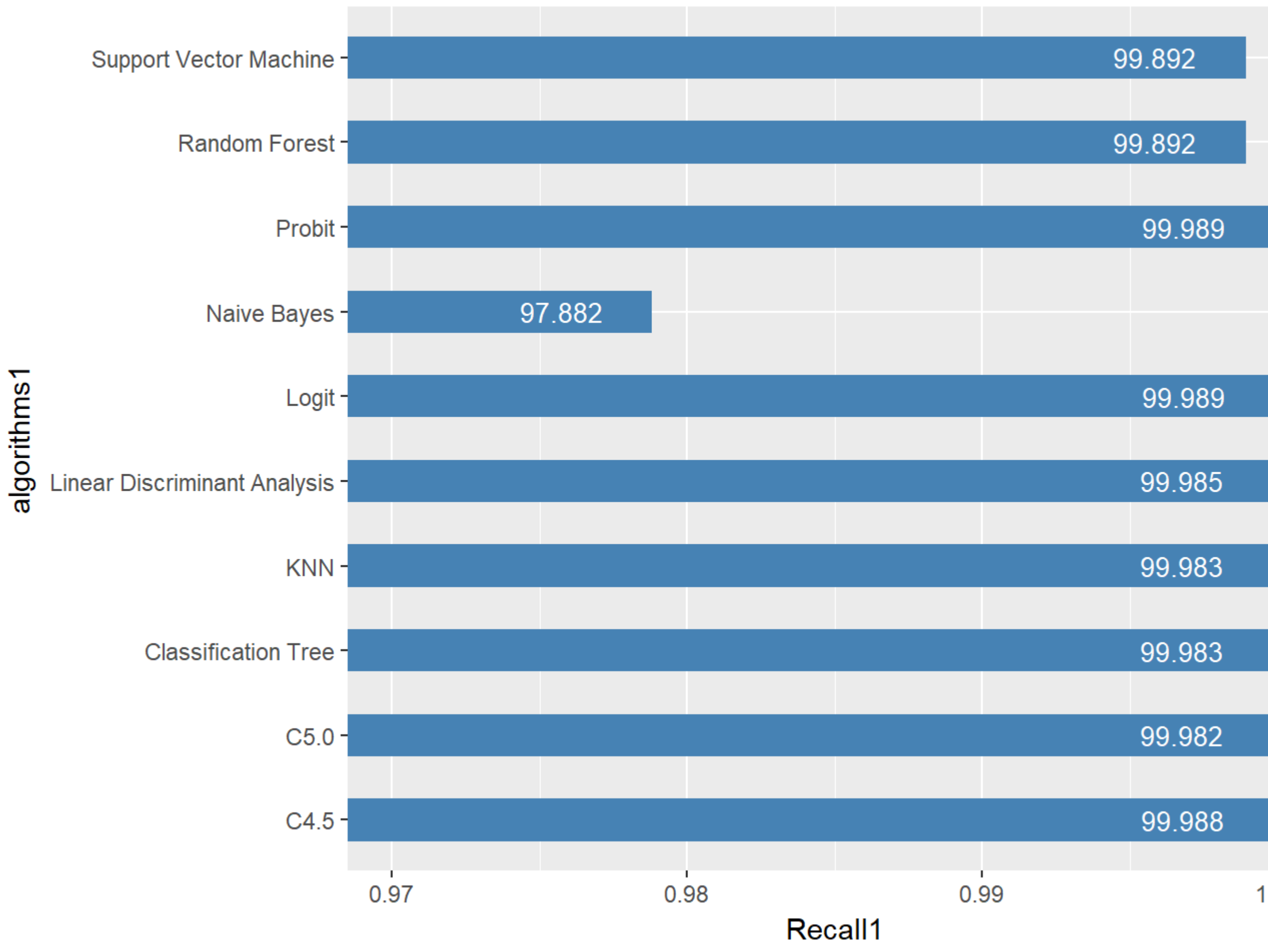
```
ggplot(data=statistics1, aes(x=algorithms1, y=Accuracy1))  + geom_bar(stat="identity", fill=
  geom_text(aes(label=round(Accuracy1 * 100,digits = 3)), hjust=1.6, color="white", size=3.5
```




```
ggplot(data=statistics1, aes(x=algorithms1, y=Specificity1))  + geom_bar(stat="identity", fi
  geom_text(aes(label=round(Specificity1 * 100,digits = 3)), hjust=1.6, color="white", size=
```

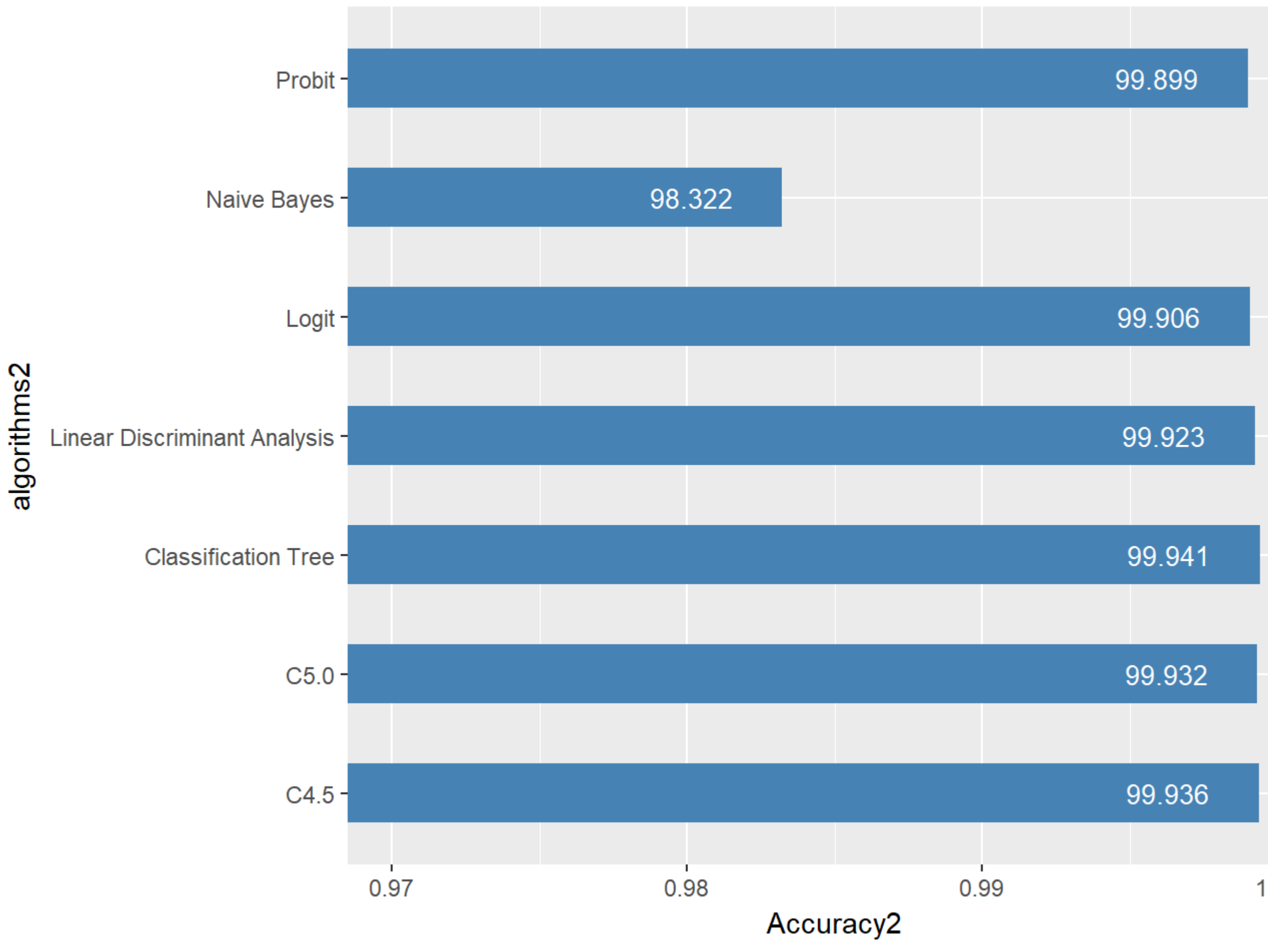



```
ggplot(data=statistics1, aes(x=algorithms1, y=Recall1))  + geom_bar(stat="identity", fill="s
  geom_text(aes(label=round(Recall1 * 100,digits = 3)), hjust=1.6, color="white", size=3.5)
```

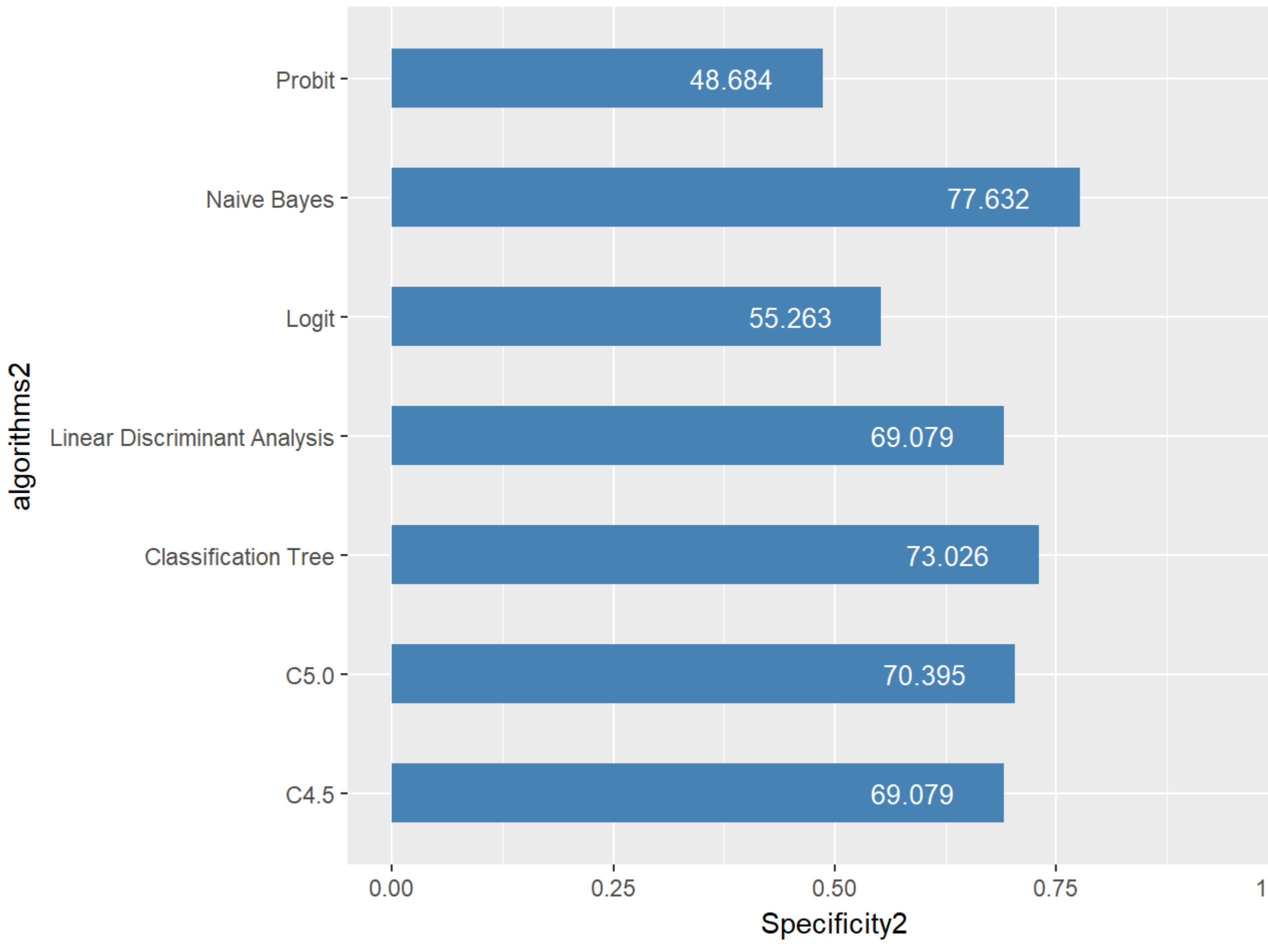


After PCA

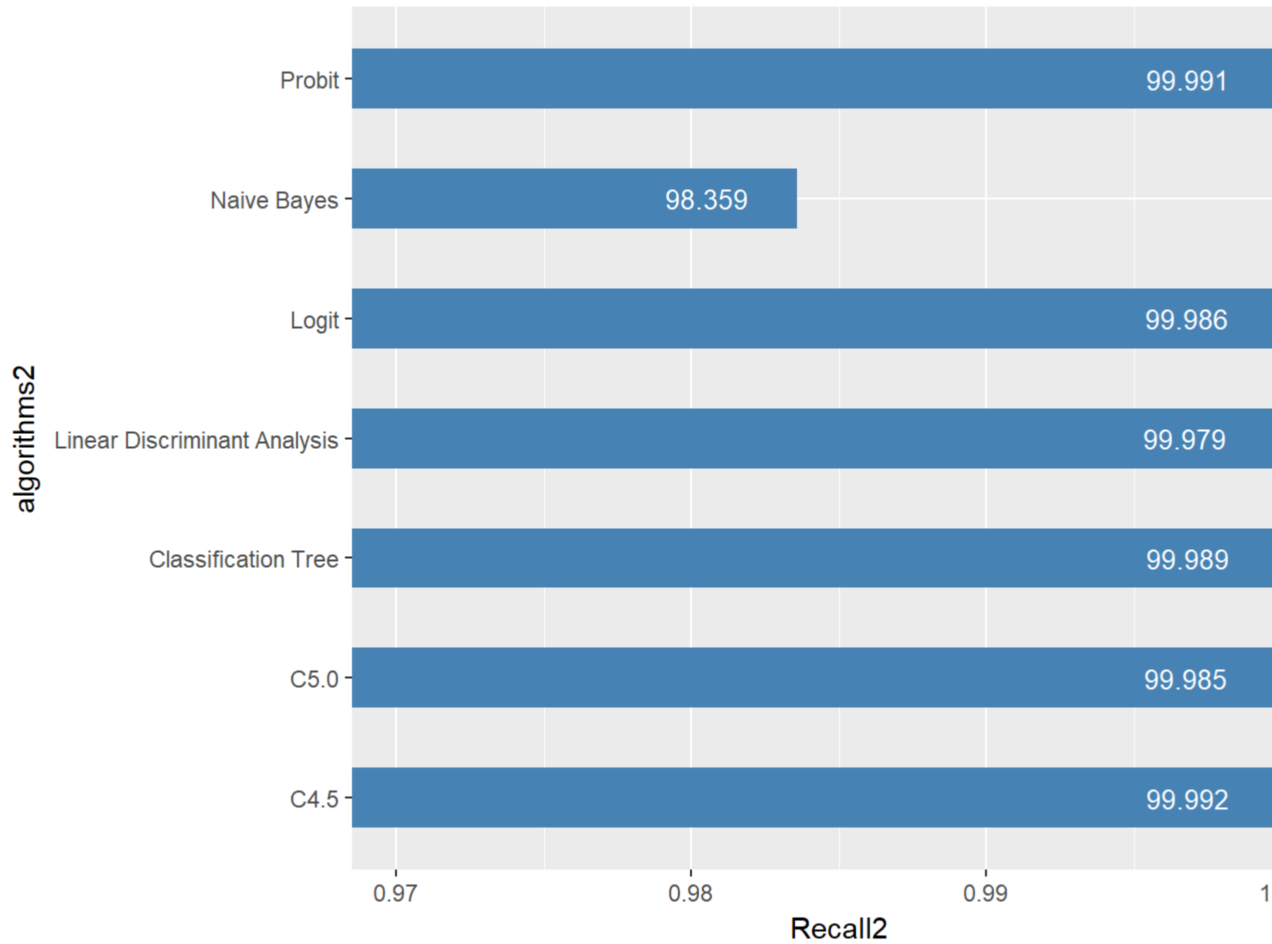
```
ggplot(data=statistics2, aes(x=algorithms2, y=Accuracy2))  + geom_bar(stat="identity", fill=
  geom_text(aes(label=round(Accuracy2 * 100,digits = 3)), hjust=1.6, color="white", size=3.5
```




```
ggplot(data=statistics2, aes(x=algorithms2, y=Specificity2))  + geom_bar(stat="identity", fi
  geom_text(aes(label=round(Specificity2 * 100,digits = 3)), hjust=1.6, color="white", size=
```




```
ggplot(data=statistics2, aes(x=algorithms2, y=Recall2))  + geom_bar(stat="identity", fill="s
  geom_text(aes(label=round(Recall2 * 100,digits = 3)), hjust=1.6, color="white", size=3.5)
```




```
library(ROCR)
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
pred_Bayes1 = as.numeric(pred_Bayes1)
pred_LDA1 = as.numeric(pred_Lda1)
pred_CTree1 = as.numeric(pred_CTree1)
pred_C4.5_1 = as.numeric(pred_C4.5_1)
pred_C5.0_1 = as.numeric(pred_C5.0_1)
pred_logit1 = as.numeric(pred_logit1)
```

```
pred_probit1 = as.numeric(pred_probit1)
pred_Boost1 = as.numeric(pred_Boost1)
pred_svm1 = as.numeric(pred_svm1)
pred_rf1 = as.numeric(pred_rf1)

y_test =  as.numeric(y_test)
y_test_t =  as.numeric(y_test_t)

pr <- prediction(pred_Bayes1, y_test)
prf <- performance(pr, "tpr", "fpr")
pr2 <- prediction(pred_LDA1, y_test)
prf2 <- performance(pr2, "tpr", "fpr")
pr3 <- prediction(pred_CTree1, y_test)
prf3 <- performance(pr3, "tpr", "fpr")
pr4 <- prediction(pred_C4.5_1, y_test)
prf4 <- performance(pr4, "tpr", "fpr")
pr5 <- prediction(pred_C5.0_1, y_test)
prf5 <- performance(pr5, "tpr", "fpr")
pr6 <- prediction(pred_logit1, y_test)
prf6 <- performance(pr6, "tpr", "fpr")
pr7 <- prediction(pred_probit1, y_test)
prf7 <- performance(pr7, "tpr", "fpr")
pr8 <- prediction(pred_Boost1, y_test)
prf8 <- performance(pr8, "tpr", "fpr")
pr9 <- prediction(pred_svm1, y_test_t)
prf9 <- performance(pr9, "tpr", "fpr")
pr10 <- prediction(pred_rf1, y_test_t)
prf10 <- performance(pr10, "tpr", "fpr")
auc_bayes = as.numeric(performance(pr, "auc")@y.values)
auc_lda = as.numeric(performance(pr2, "auc")@y.values)
auc_ctree = as.numeric(performance(pr3, "auc")@y.values)
auc_c45 = as.numeric(performance(pr4, "auc")@y.values)
auc_c50 = as.numeric(performance(pr5, "auc")@y.values)
auc_log = as.numeric(performance(pr6, "auc")@y.values)
auc_prob = as.numeric(performance(pr7, "auc")@y.values)
auc_boost = as.numeric(performance(pr8, "auc")@y.values)
auc_svm = as.numeric(performance(pr9, "auc")@y.values)
auc_rf = as.numeric(performance(pr10, "auc")@y.values)
print("auc_bayes: ", auc_bayes)
```

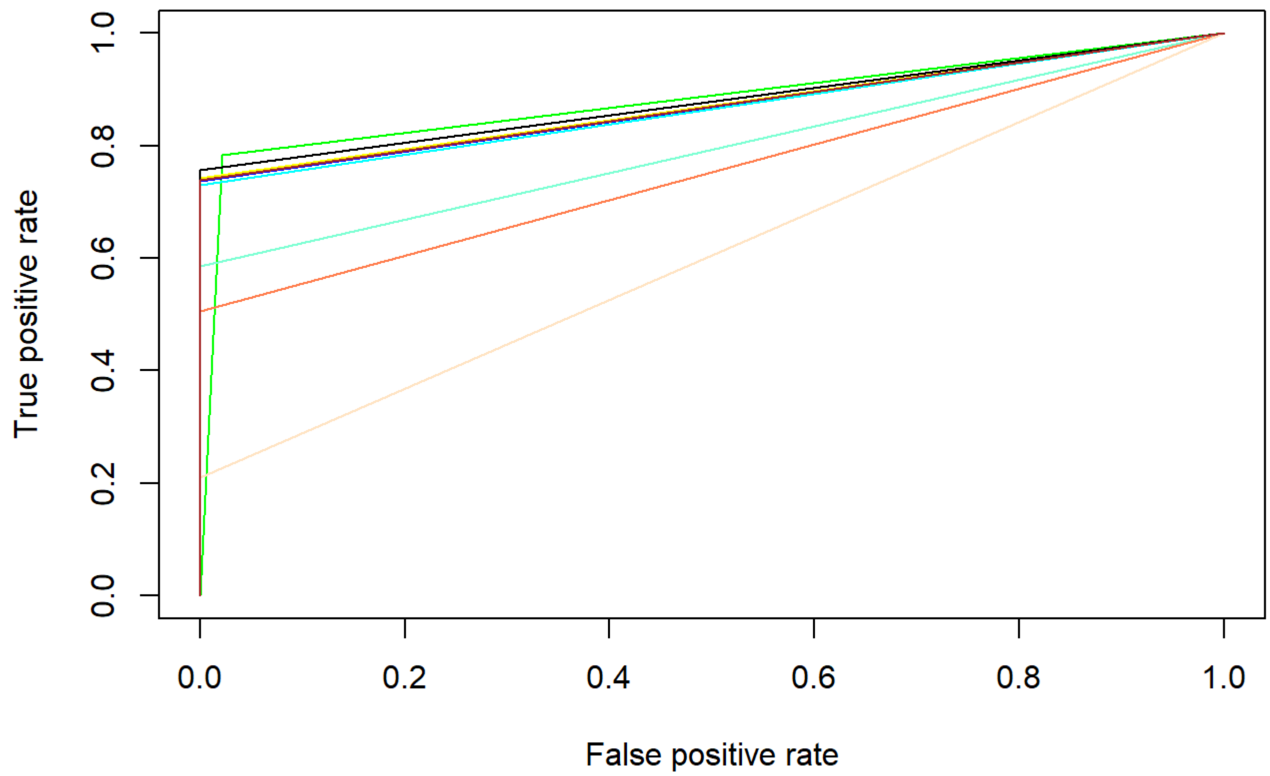
```
## [1] "auc_bayes: "
```

```
print("auc_lda: ", auc_lda)
```

```
## [1] "auc_lda: "
```

```
print("auc_ctree: ", auc_ctree)
```

```
## [1] "auc_ctree: "
print("auc_c45: ", auc_c45)
## [1] "auc_c45: "
print("auc_c50: ", auc_c50)
## [1] "auc_c50: "
print("auc_logit: ", auc_log)
## [1] "auc_logit: "
print("auc_probit: ", auc_prob)
## [1] "auc_probit: "
print("auc_boost: ", auc_boost)
## [1] "auc_boost: "
print("auc_svm: ", auc_svm)
## [1] "auc_svm: "
print("auc_rf: ", auc_rf)
## [1] "auc_rf: "
plot(prf, col='green', legend.title="4 bootstrap-crossvalidation steps")
plot(prf2, add=TRUE, col= 'red', legend.title="4 bootstrap-crossvalidation steps")
plot(prf3, add=TRUE, col= 'blue', legend.title="4 bootstrap-crossvalidation steps")
plot(prf4, add=TRUE, col= 'yellow', legend.title="4 bootstrap-crossvalidation steps")
plot(prf5, add=TRUE, col= 'black', legend.title="4 bootstrap-crossvalidation steps")
plot(prf6, add=TRUE, col= 'aquamarine', legend.title="4 bootstrap-crossvalidation steps")
plot(prf7, add=TRUE, col= 'coral', legend.title="4 bootstrap-crossvalidation steps")
plot(prf8, add=TRUE, col= 'cyan', legend.title="4 bootstrap-crossvalidation steps")
plot(prf9, add=TRUE, col= 'bisque', legend.title="4 bootstrap-crossvalidation steps")
plot(prf10, add=TRUE, col= 'brown', legend.title="4 bootstrap-crossvalidation steps")
```


True positive rate
0.0
0.2
0.4
0.6
0.8
1.0
False positive rate