B565_hw4

1

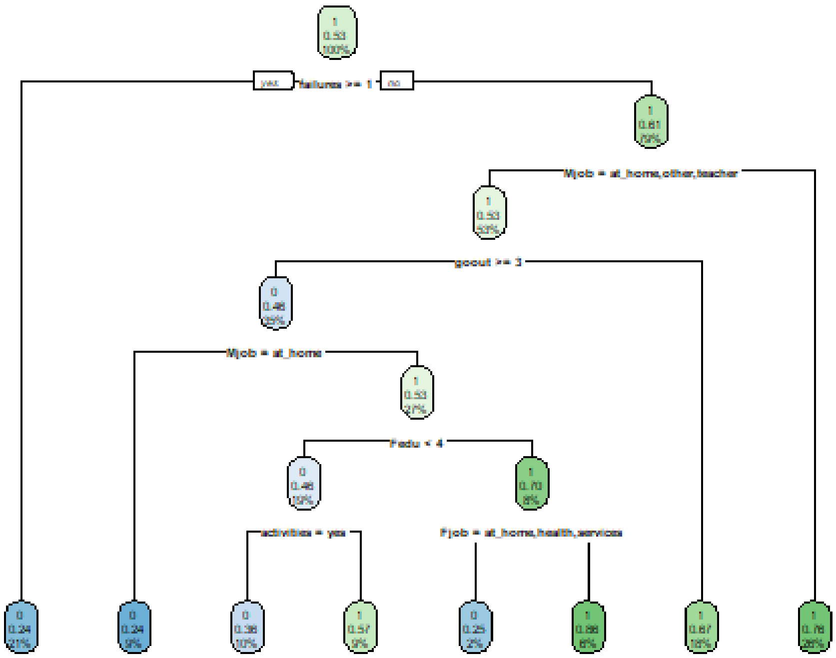
```
library(e1071)
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

# (a)
# Reading naive bayes binary file and factorizing the variables
df =  read.csv("D:/Data Mining/hw4/naive_bayes_binary.csv",colClasses =
c('factor','factor','factor','factor','factor','factor','factor','factor','fa
ctor','factor','factor'))

# Splitting the dataset into train and test
n = nrow(df)
train = df[1:(n/2),]
test = df[((n/2)+1):n,]

# function for prior probablities
prior = function(class,vector)
  return (p_prob = length(which(vector==class))/length(vector))

# function for conditional probabilities
cond_prob = function(class,value,v)
  return (c_prob = length(which(v[,1]==value & v[,2] ==
class))/length(which(v[,2]==class)))

# (b)
# Calculating the prior and conditional probablities using training dataset
and predicting the probabilities for each class using test data set

test$pred_1 = 0
test$pred_2 = 0
test$pred_3 = 0

for (i in 1:nrow(test))
  test[i,12] =
prior(1,train$V11)*cond_prob(1,test[i,1],train[,c(1,11)])*cond_prob(1,test[i,
2],train[,c(2,11)])*cond_prob(1,test[i,3],train[,c(3,11)])*cond_prob(1,test[i
,4],train[,c(4,11)])*cond_prob(1,test[i,5],train[,c(5,11)])*cond_prob(1,test[
i,6],train[,c(6,11)])*cond_prob(1,test[i,7],train[,c(7,11)])*cond_prob(1,test
[i,8],train[,c(8,11)])*cond_prob(1,test[i,9],train[,c(9,11)])*cond_prob(1,tes
t[i,10],train[,c(10,11)])
```

```
for (i in 1:nrow(test))
  test[i,13] =
prior(2,train$V11)*cond_prob(2,test[i,1],train[,c(1,11)])*cond_prob(2,test[i,
2],train[,c(2,11)])*cond_prob(2,test[i,3],train[,c(3,11)])*cond_prob(2,test[i
,4],train[,c(4,11)])*cond_prob(2,test[i,5],train[,c(5,11)])*cond_prob(2,test[
i,6],train[,c(6,11)])*cond_prob(2,test[i,7],train[,c(7,11)])*cond_prob(2,test
[i,8],train[,c(8,11)])*cond_prob(2,test[i,9],train[,c(9,11)])*cond_prob(2,tes
t[i,10],train[,c(10,11)])

for (i in 1:nrow(test))
  test[i,14] =
prior(3,train$V11)*cond_prob(3,test[i,1],train[,c(1,11)])*cond_prob(3,test[i,
2],train[,c(2,11)])*cond_prob(3,test[i,3],train[,c(3,11)])*cond_prob(3,test[i
,4],train[,c(4,11)])*cond_prob(3,test[i,5],train[,c(5,11)])*cond_prob(3,test[
i,6],train[,c(6,11)])*cond_prob(3,test[i,7],train[,c(7,11)])*cond_prob(3,test
[i,8],train[,c(8,11)])*cond_prob(3,test[i,9],train[,c(9,11)])*cond_prob(3,tes
t[i,10],train[,c(10,11)])

# Classify each vector to class having the maximum probablity
test$class = 0
for (i in  1:nrow(test))
  test[i,15] = which.max(apply(test[i,c(12,13,14)],MARGIN=2,max))

test$class = as.factor(test$class)

# Creating the confusion matrix for test actual class and predicted class
t = confusionMatrix(test$class,test$V11)
t1 = as.table(t)
t1

##             Reference
## Prediction    1    2    3
##          1  177   16    6
##          2   23  206   13
##          3   55   31 1973
```

2

```
# (a)

library(rpart)
library(rpart.plot)

# reading the student performance dataset
d1=read.table("D:/Data Mining/hw4/student-mat.csv",sep=";",header=TRUE)

# creating a target variable using G3
d1$y = ifelse(d1$G3>10,1,0)
```

```r
# subsetting the dataset removing G1,G2,and G3 variable
df = subset(d1,select=-c(G1,G2,G3))

set.seed(100)

# Training the decision tree classifier using train dataset and checking the
value of cp where relative error is minimum
model = rpart(formula = y~.,data = df,method="class",control=rpart.control(cp
= 0))
printcp(model)

##
## Classification tree:
## rpart(formula = y ~ ., data = df, method = "class", control =
rpart.control(cp = 0))
##
## Variables actually used in tree construction:
##  [1] absences   activities failures   famsup     Fedu       Fjob
##  [7] freetime   goout      guardian   health     Mjob       reason
##
## Root node error: 186/395 = 0.47089
##
## n= 395
##
##          CP nsplit rel error  xerror     xstd
## 1 0.2311828      0   1.00000 1.00000 0.053336
## 2 0.0322581      1   0.76882 0.79570 0.051721
## 3 0.0268817      5   0.63978 0.85484 0.052401
## 4 0.0215054      6   0.61290 0.83333 0.052175
## 5 0.0161290      7   0.59140 0.84409 0.052291
## 6 0.0107527     10   0.54301 0.86022 0.052454
## 7 0.0080645     11   0.53226 0.89247 0.052742
## 8 0.0071685     13   0.51613 0.88710 0.052698
## 9 0.0000000     17   0.47849 0.91935 0.052943

# Pruning the tree and plotting the best tree model
modelp = prune(model,cp=0.021)
rpart.plot(modelp)
```

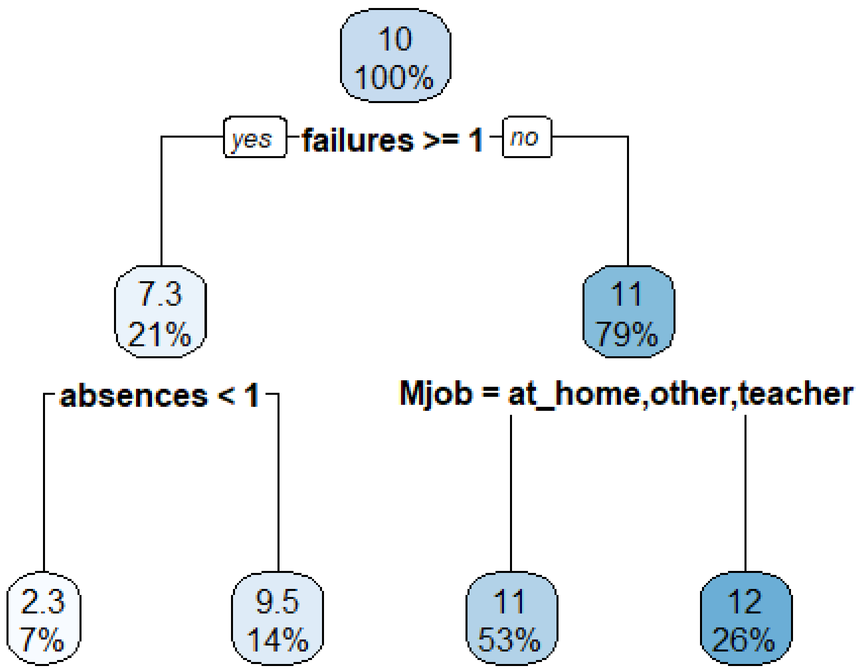



```
# (b)
#Calculating the training error and generalisation error
error_train = 0.47*0.59
error_gen = 0.47*0.84
error_train

## [1] 0.2773

error_gen

## [1] 0.3948

# Since the trees has been pruned in order to avoid overfiiting. For pruning,
the split on non-terminal nodes has been penalised with cp = 0.021 and cross
validated in order to get the minimum error rate on generalised dataset. The
error rate on training and generalisation error would not be exact since we
have trained on a different dataset and validated on different dataset but
the error rate on validated dataset would be minimum given the optimal split
penalty we have taken.

# (c)
# calculating the most important variable
var_imp = modelp$variable.importance
var_imp

##   failures       Mjob       Fjob      goout        age       Fedu
## 17.4497032 12.6357332  5.7207035  4.2857143  3.7577091  2.8242800
##   guardian   absences activities     higher       Medu     reason
```

```
##   2.1023739  1.7909561  1.6522778  1.2614243  1.0612058  0.5522727
##      school       paid    address     famrel   freetime traveltime
##   0.5522727  0.3572493  0.3289374  0.2256932  0.1836735  0.1300068

# 'failures' is the most important variable

# (d)
# Repeating the steps above

df1 = subset(d1,select=-c(G1,G2,y))

set.seed(100)

model1 = rpart(formula = G3~.,data =
df1,method="anova",control=rpart.control(cp = 0))
printcp(model1)

##
## Regression tree:
## rpart(formula = G3 ~ ., data = df1, method = "anova", control =
rpart.control(cp = 0))
##
## Variables actually used in tree construction:
##  [1] absences  address   Dalc      failures  Fedu      Fjob      freetime
##  [8] guardian  health    Mjob      reason    sex       studytime Walc
##
## Root node error: 8269.9/395 = 20.936
##
## n= 395
##
##             CP nsplit rel error  xerror     xstd
## 1  0.1260889      0   1.00000 1.00829 0.078406
## 2  0.1142592      1   0.87391 0.95750 0.077501
## 3  0.0233634      2   0.75965 0.77074 0.068788
## 4  0.0145405      3   0.73629 0.83468 0.073739
## 5  0.0128850     10   0.63404 0.91586 0.078230
## 6  0.0090577     13   0.59538 0.92824 0.080640
## 7  0.0078754     14   0.58633 0.92267 0.080267
## 8  0.0072754     15   0.57845 0.92034 0.081359
## 9  0.0061414     16   0.57118 0.92562 0.081122
## 10 0.0052215     17   0.56503 0.96698 0.084300
## 11 0.0048656     19   0.55459 0.97808 0.084923
## 12 0.0047132     20   0.54973 0.98040 0.084900
## 13 0.0046436     21   0.54501 0.98808 0.084993
## 14 0.0045709     22   0.54037 0.98794 0.085002
## 15 0.0043780     23   0.53580 0.99117 0.085228
## 16 0.0032130     24   0.53142 0.99372 0.085280
## 17 0.0031490     25   0.52821 0.99141 0.084869
## 18 0.0028614     26   0.52506 0.98970 0.084783
## 19 0.0028278     28   0.51934 0.98899 0.084785
```

```
## 20 0.0024273     29   0.51651 0.99139 0.084973
## 21 0.0022010     30   0.51408 0.99750 0.084982
## 22 0.0000000     31   0.51188 0.99458 0.084601
```

```r
modelp1 = prune(model1,cp=0.023)
rpart.plot(modelp1)
```



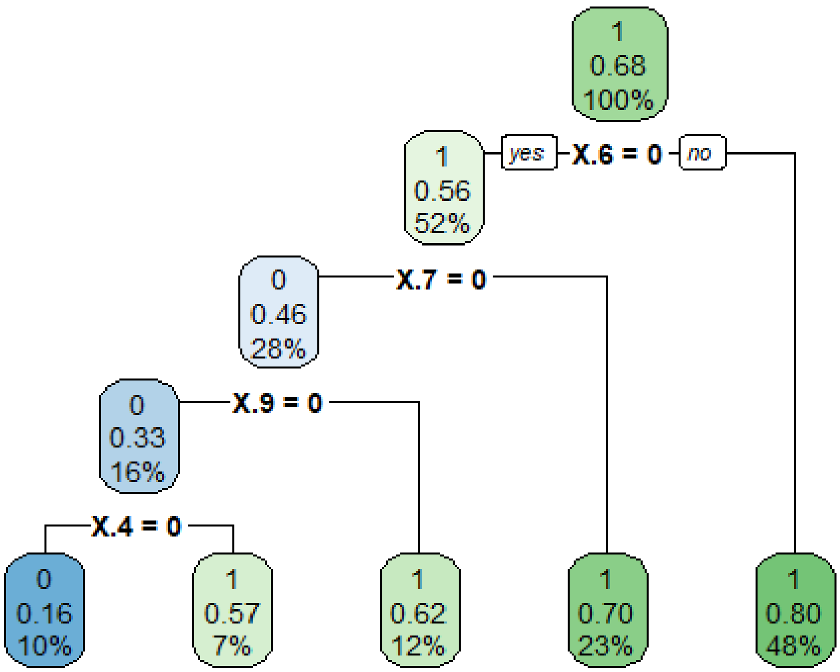

```r
error_train1 = 20.93*0.75
error_gen1 = 20.93*0.76
error_train1

## [1] 15.6975

error_gen1

## [1] 15.9068

# Since the trees has been pruned in order to avoid overfiiting. For pruning,
the split on non-terminal nodes has been penalised with cp = 0.023 and cross
validated in order to get the minimum error rate on generalised dataset. The
error rate on training and generalisation error would not be exact since we
have trained on a different dataset and validated on different dataset but
the error rate on validated dataset would be minimum given the optimal split
penalty we have taken.

var_imp1 = modelp1$variable.importance
var_imp1
```

```
##   failures   absences        age       Mjob   guardian     higher
## 1042.74339  944.91294  213.57395  193.21332  125.63173   75.37904
## traveltime      goout
##   72.68561   36.34281

# 'failures' is the most important variable
```

4

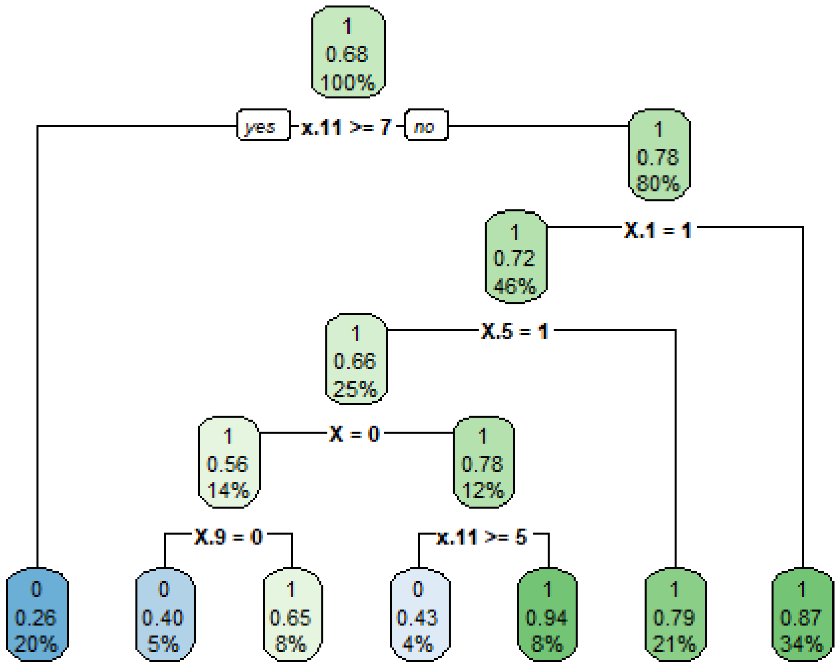
```
# (a)

# Reading the dataset and factorizing the binary variables
d1 =  read.csv("D:/Data Mining/hw4/strange_binary.csv",colClasses =
c('factor','factor','factor','factor','factor','factor','factor','factor','fa
ctor','factor','factor'))

# Creating target variable as binary and factorizing it
d1$y = ifelse(d1$c=='good',1,0)
d1$y = as.factor(d1$y)

# subset columns
df = subset(d1,select=-c(c))

# creating decision tree classifier using train dataset and printing cp of
the model
model = rpart(formula = y~.,data = df,method="class",control=rpart.control(cp
= 0))
printcp(model)

##
## Classification tree:
## rpart(formula = y ~ ., data = df, method = "class", control =
rpart.control(cp = 0))
##
## Variables actually used in tree construction:
## [1] X   X.1 X.2 X.3 X.4 X.6 X.7 X.9
##
## Root node error: 64/200 = 0.32
##
## n= 200
##
##          CP nsplit rel error xerror    xstd
## 1 0.0572917      0   1.00000 1.0000 0.10308
## 2 0.0312500      3   0.82812 1.0938 0.10540
## 3 0.0156250      4   0.79688 1.0469 0.10430
## 4 0.0078125      5   0.78125 1.0469 0.10430
## 5 0.0000000      9   0.75000 1.0469 0.10430
```

```r
# pruning the model at 3 split
modelp = prune(model,cp=0.03)
rpart.plot(modelp)
```

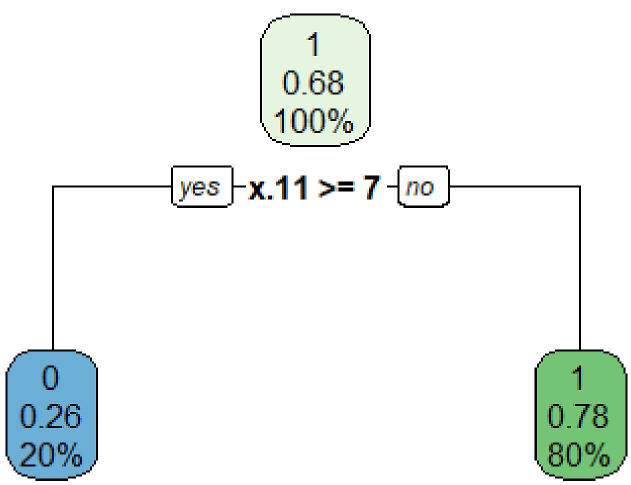



```r
# calculating the error rate on training and generalised dataset
train_error = 0.32*0.82
gen_error = 0.32*1.0938
train_error

## [1] 0.2624

gen_error

## [1] 0.350016

# Calculating the important variables for prediction
var_imp = modelp$variable.importance
var_imp

##        X.6        X.7        X.4        X.9        X.2        X.1
## 5.80368732 2.92064903 2.75689223 2.42752746 1.14778636 0.91490281
##          X        X.3        X.5
## 0.41882280 0.15833329 0.05983183

# It is not reasonable to assume error rate on generalised data set similar
because at split n = 3, the x error is high. So for generalised dataset, the
error rate will be higher.
```

```r
# (b)
# creating features which is sum of 0's and 1's respectively

library("reshape2")
d1$x.11 = rowSums(d1[,1:10]==0)
d1$x.12 = rowSums(d1[,1:10]==1)

# subsetting the dataset
df = subset(d1,select=-c(c))

# Training the model using train dataset and plotting the cp and tree
model = rpart(formula = y~.,data = df,method="class",control=rpart.control(cp
= 0))
printcp(model)

##
## Classification tree:
## rpart(formula = y ~ ., data = df, method = "class", control =
rpart.control(cp = 0))
##
## Variables actually used in tree construction:
## [1] X    X.1  x.11 X.5  X.9
##
## Root node error: 64/200 = 0.32
##
## n= 200
##
##          CP nsplit rel error  xerror     xstd
## 1 0.296875      0   1.00000 1.00000 0.103078
## 2 0.009375      1   0.70312 0.70312 0.092274
## 3 0.000000      6   0.65625 0.82812 0.097522

rpart.plot(model)
```




```
# pruning the model using no more than 3 splits
modelp = prune(model,cp=0.009375)
rpart.plot(modelp)
```



```
# calculating the error rate for train and generalised dataset which has
reduced around 20%
train_error1 = 0.32*0.7
gen_error1 = 0.32*0.7

# printing variable score
var_imp = modelp$variable.importance
var_imp

##    x.11    x.12
## 17.3856 17.3856
```

6

```
library(stringr)
library(data.table)

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:reshape2':
##
##     dcast, melt

df =  read.csv("D:/Data Mining/hw4/classification_accuracy.csv")
df$error_dt = 100 - df$decision_tree
df$error_nb = 100 - df$naive_bayes
df$error_svm = 100 - df$svm

df1 = subset(df,select=-c(decision_tree,svm,naive_bayes))
df1$error_dt = df1$error_dt/100
df1$error_nb = df1$error_nb/100
df1$error_svm = df1$error_svm/100

df1$dt_ci_l = df1$error_dt - qnorm(0.995)*((df1$error_dt*(1-
df1$error_dt))/(df1$size))
df1$dt_ci_h = df1$error_dt + qnorm(0.995)*((df1$error_dt*(1-
df1$error_dt))/(df1$size))

df1$nb_ci_l = df1$error_nb - qnorm(0.995)*((df1$error_nb*(1-
df1$error_nb))/(df1$size))
df1$nb_ci_h = df1$error_nb + qnorm(0.995)*((df1$error_nb*(1-
df1$error_nb))/(df1$size))

df1$svm_ci_l = df1$error_svm - qnorm(0.995)*((df1$error_svm*(1-
df1$error_svm))/(df1$size))
df1$svm_ci_h = df1$error_svm + qnorm(0.995)*((df1$error_svm*(1-
df1$error_svm))/(df1$size))
```

```
status = function(l1,h1,l2,h2,l3,h3) {
  if ((h1<l2) & (h1<l3)) {
    s = 'WW'
  } else if (((l2<=h1 & h1<=h2)|(l2<=l1 & l1<=h2)) & (h1<l3)) {
    s = 'DW'
  } else if ((h1<l2) & ((l3<=h1 & h1<=h3)|(l3<=l1 & l1<=h3))) {
    s = 'WD'
  } else if ((h1<l2) & (l1>h3)) {
    s = 'WL'
  } else if ((l1>h2) & (h1<l3)) {
    s = 'LW'
  } else if (((l2<=h1 & h1<=h2)|(l2<=l1 & l1<=h2)) & (l1>h3)) {
    s = 'DL'
  } else if ((l1>h2) & ((l3<=h1 & h1<=h3)|(l3<=l1 & l1<=h3))) {
    s = 'LD'
  } else if (((l2<=h1 & h1<=h2)|(l2<=l1 & l1<=h2)) & ((l3<=h1 &
h1<=h3)|(l3<=l1 & l1<=h3))) {
    s = 'DD'
  } else {
    s = 'LL'
  }
  s
}

df1$dt_s = NA
for (i in 1:nrow(df))
  df1[i,12] = status(df1[i,6],df1[i,7],df1[i,8],df1[i,9],df1[i,10],df1[i,11])

df1$nb_s = NA
for (i in 1:nrow(df))
  df1[i,13] = status(df1[i,8],df1[i,9],df1[i,6],df1[i,7],df1[i,10],df1[i,11])

df1$svm_s = NA
for (i in 1:nrow(df))
  df1[i,14] = status(df1[i,10],df1[i,11],df1[i,6],df1[i,7],df1[i,8],df1[i,9])

df1$win_dt = str_count(df1$dt_s, "W")
df1$draw_dt = str_count(df1$dt_s, "D")
df1$loss_dt = str_count(df1$dt_s, "L")

df1$win_nb = str_count(df1$nb_s, "W")
df1$draw_nb = str_count(df1$nb_s, "D")
df1$loss_nb = str_count(df1$nb_s, "L")

df1$win_svm = str_count(df1$svm_s, "W")
df1$draw_svm = str_count(df1$svm_s, "D")
df1$loss_svm = str_count(df1$svm_s, "L")

win = c(decisiontree = sum(df1$win_dt),naivebayes = sum(df1$win_nb),svm =
```

```r
sum(df1$win_svm))
draw = c(decisiontree = sum(df1$draw_dt),naivebayes = sum(df1$draw_nb),svm =
sum(df1$draw_svm))
loss = c(decisiontree = sum(df1$loss_dt),naivebayes = sum(df1$loss_nb),svm =
sum(df1$loss_svm))

t = cbind(win,draw,loss)
t
```

```
##              win draw loss
## decisiontree  19    1   26
## naivebayes    14    1   31
## svm           35    0   11
```