In [1]:
```
#We will be fitting some classification models on a cardio vascular dataset from K
#Thise dataset has 12 variables including the target variable "cardio" with levels
#absence of the disease respectively
```

In [ ]:
```
https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset
```

In [ ]:
```
setwd("C:\\Users\\Khumo Sibeko\\Documents\\datasets")
```

In [2]:
```
install.packages("e1071")
library(e1071)
```

```
  There is a binary version available but the source version is later:
      binary source needs_compilation
e1071  1.7-6 1.7-12              TRUE

  Binaries will be installed
package 'e1071' successfully unpacked and MD5 sums checked
```

```
Warning message:
"cannot remove prior installation of package 'e1071'"Warning message in file.copy
(savedcopy, lib, recursive = TRUE):
"problem copying C:\Anaconda\envs\r-tutorial\Lib\R\library\00LOCK\e1071\libs\x64\e
1071.dll to C:\Anaconda\envs\r-tutorial\Lib\R\library\e1071\libs\x64\e1071.dll: Pe
rmission denied"Warning message:
"restored 'e1071'"
```

```
The downloaded binary packages are in
        C:\Users\Khumo Sibeko\AppData\Local\Temp\RtmpQhLmWu\downloaded_packages
```

```
Warning message:
"package 'e1071' was built under R version 3.6.3"
```

In [3]:
```
install.packages("randomForest")
library(randomForest)
```

```
Warning message:
"package 'randomForest' is not available (for R version 3.6.1)"randomForest 4.6-14
Type rfNews() to see new features/changes/bug fixes.
```

In [4]:
```
install.packages("gbm")
library(gbm)
```

```
  There is a binary version available but the source version is later:
    binary  source needs_compilation
gbm  2.1.8 2.1.8.1              TRUE

  Binaries will be installed
package 'gbm' successfully unpacked and MD5 sums checked
```

```
Warning message:
"cannot remove prior installation of package 'gbm'"Warning message in file.copy(sa
vedcopy, lib, recursive = TRUE):
"problem copying C:\Anaconda\envs\r-tutorial\Lib\R\library\00LOCK\gbm\libs\x64\gb
m.dll to C:\Anaconda\envs\r-tutorial\Lib\R\library\gbm\libs\x64\gbm.dll: Permissio
n denied"Warning message:
"restored 'gbm'"
```

```
The downloaded binary packages are in
        C:\Users\Khumo Sibeko\AppData\Local\Temp\RtmpQhLmWu\downloaded_packages
```

```
Warning message:
"package 'gbm' was built under R version 3.6.3"Loaded gbm 2.1.8
```

In [5]:
```
install.packages("dplyr",type = "binary")
```

```
  There is a binary version available (and will be installed) but the
  source version is later:
      binary source
dplyr  1.0.6 1.0.10
```

```
Warning message in download.file(url, destfile, method, mode = "wb", ...):
"downloaded length 49152 != reported length 1559698"Warning message in unzip(zipna
me, exdir = dest):
"error 1 in extracting from zip file"Warning message in read.dcf(file.path(pkgnam
e, "DESCRIPTION"), c("Package", "Type")):
"cannot open compressed file 'dplyr/DESCRIPTION', probable reason 'No such file or
directory'"
Error in read.dcf(file.path(pkgname, "DESCRIPTION"), c("Package", "Type")): cannot
open the connection
Traceback:

1. install.packages("dplyr", type = "binary")
2. .install.winbinary(pkgs = pkgs, lib = lib, contriburl = contriburl,
 .     method = method, available = available, destdir = destdir,
 .     dependencies = dependencies, libs_only = libs_only, quiet = quiet,
 .     ...)
3. unpackPkgZip(foundpkgs[okp, 2L], foundpkgs[okp, 1L], lib, libs_only,
 .     lock)
4. read.dcf(file.path(pkgname, "DESCRIPTION"), c("Package", "Type"))
```

In [6]:
```
library(dplyr)
```

```
Warning message:
"package 'dplyr' was built under R version 3.6.3"
Attaching package: 'dplyr'

The following object is masked from 'package:randomForest':

    combine

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

In [7]:
```
cvascular.data = read.csv("cardio.csv",header = TRUE,sep = ";")
head(cvascular.data)
```

id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	18393	2	168	62	110	80	1	1	0	0	1	0
1	20228	1	156	85	140	90	3	1	0	0	1	1
2	18857	1	165	64	130	70	3	1	0	0	0	1
3	17623	2	169	82	150	100	1	1	0	0	1	1
4	17474	1	156	56	100	60	1	1	0	0	0	0
8	21914	1	151	67	120	80	2	2	0	0	0	0

In [ ]:
```
#SOME DATA PREPARATION
```

In [ ]:
```
#We see that age is in days.we convert it to years.
```

In [8]:
```
cvascular.data$age = round(cvascular.data$age/365)
head(cvascular.data)
```

id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	50	2	168	62	110	80	1	1	0	0	1	0
1	55	1	156	85	140	90	3	1	0	0	1	1
2	52	1	165	64	130	70	3	1	0	0	0	1
3	48	2	169	82	150	100	1	1	0	0	1	1
4	48	1	156	56	100	60	1	1	0	0	0	0
8	60	1	151	67	120	80	2	2	0	0	0	0

In [9]:
```
#Lets rename the variables in a meaningful way
cvascular.data = cvascular.data %>% rename("sys_pressure"="ap_hi",
                          "dias_pressure"="ap_lo", "glucose"="gluc",
                          "alcohol"="alco","disease"="cardio")
```

In [10]:
```
#Remove missing(NaN) values
na.omit(cvascular.data)
sum(is.na(cvascular.data))
```

id	age	gender	height	weight	sys_pressure	dias_pressure	cholesterol	glucose	smoke	alc
0	50	2	168	62	110	80	1	1	0	
1	55	1	156	85	140	90	3	1	0	
2	52	1	165	64	130	70	3	1	0	
3	48	2	169	82	150	100	1	1	0	
4	48	1	156	56	100	60	1	1	0	
8	60	1	151	67	120	80	2	2	0	
9	61	1	157	93	130	80	3	1	0	
12	62	2	178	95	130	90	3	3	0	
13	48	1	158	71	110	70	1	1	0	
14	54	1	164	68	110	60	1	1	0	
15	62	1	169	80	120	80	1	1	0	
16	52	2	173	60	120	80	1	1	0	
18	41	2	165	60	120	80	1	1	0	
21	54	1	158	78	110	70	1	1	0	
23	40	2	181	95	130	90	1	1	1	
24	46	2	172	112	120	80	1	1	0	
25	58	1	170	75	130	70	1	1	0	
27	46	1	158	52	110	70	1	3	0	
28	48	1	154	68	100	70	1	1	0	
29	60	2	162	56	120	70	1	1	1	
30	54	2	163	83	120	80	1	1	0	
31	59	1	157	69	130	80	1	1	0	
32	63	1	158	90	145	85	2	2	0	
33	64	2	156	45	110	60	1	1	0	
35	46	1	170	68	150	90	3	1	0	
36	40	1	153	65	130	100	2	1	0	
37	54	1	156	59	130	90	1	1	0	
38	50	1	159	78	120	80	1	1	0	
39	40	2	166	66	120	80	1	1	0	
40	58	2	169	74	130	70	1	3	0	
...	...	...	...	...	...	...	...	...	...	
99958	62	2	173	103	140	80	3	1	1	
99959	55	2	177	80	130	80	1	1	0	
99960	47	1	165	76	140	90	1	1	0	
99961	61	2	175	72	130	80	1	1	0	
99962	50	1	168	75	120	80	1	1	0	

id	age	gender	height	weight	sys_pressure	dias_pressure	cholesterol	glucose	smoke	alc
99963	58	2	182	100	120	80	1	1	0	
99964	59	1	163	65	120	80	2	2	0	
99965	46	1	168	75	120	79	1	1	0	
99967	52	1	163	78	90	60	1	1	0	
99969	61	1	163	74	160	100	2	2	0	
99971	49	2	167	69	110	80	1	1	0	
99972	48	2	182	110	130	90	2	2	0	
99973	52	1	153	86	130	90	1	2	0	
99974	54	1	165	72	120	80	1	1	0	
99975	49	2	168	80	120	80	1	1	0	
99977	50	1	156	102	130	80	1	1	0	
99978	50	2	180	78	120	80	1	1	0	
99979	52	1	151	49	120	80	1	1	0	
99981	60	1	160	59	110	70	1	1	0	
99985	58	1	157	83	120	70	1	1	0	
99986	41	1	168	72	110	70	1	1	0	
99988	56	1	159	72	130	90	2	2	0	
99990	51	1	161	56	170	90	1	1	0	
99991	54	1	172	70	130	90	1	1	0	
99992	58	1	165	80	150	80	1	1	0	
99993	53	2	168	76	120	80	1	1	1	
99995	62	1	158	126	140	90	2	2	0	
99996	52	2	183	105	180	90	3	1	0	
99998	61	1	163	72	135	80	1	2	0	
99999	56	1	170	72	120	80	2	1	0	

In [11]:
```
#removing the id variable (not useful for analysis)
cvascular.data = cvascular.data[,-1]
head(cvascular.data)
```

age	gender	height	weight	sys_pressure	dias_pressure	cholesterol	glucose	smoke	alcohol	a
50	2	168	62	110	80	1	1	0	0	
55	1	156	85	140	90	3	1	0	0	
52	1	165	64	130	70	3	1	0	0	
48	2	169	82	150	100	1	1	0	0	
48	1	156	56	100	60	1	1	0	0	
60	1	151	67	120	80	2	2	0	0	

In [12]:
```
#change target  variable to a factor for classification
cvascular.data$disease = as.factor(cvascular.data$disease)
```

In [13]:
```
#remove missing values from the target variable (#we've already done so but this re
head(na.omit(cvascular.data$disease))
```

1. 0
2. 1
3. 1
4. 1
5. 0
6. 0

▶ **Levels:**

In [ ]:
```
#We start with bagging and Random Forest models for classification
```

In [14]:
```
#declaring no. of variables and observations
p = ncol(cvascular.data)
n = nrow(cvascular.data)/5 ##decrease size of data for computational efficiency
```

In [15]:
```
#split the data for training and testing
train = sample(1:n,n/2)
test = (1:n)[-train]
```

In [16]:
```
#build owun function(mrate) that computes the performance measure(missclassificati

mrate = function(predicted,actual){

  cmatrix = table(actual,predicted)

  TP = cmatrix[2,2]
  TN = cmatrix[2,1]
  FN = cmatrix[1,1]
  FP = cmatrix[1,2]

  merror = (TN+FP)/(sum(cmatrix))

  return(merror)

}
```

In [17]:
```
#grid of inputs to use to find the best model
ptry = c(p,round(sqrt(p)),round(p/2)) #grid of p(# of variables) to try at each sp
ntree = seq(from = 0,to = 100,by = 25)
ntree = ntree[2:length(ntree)]
```

In [18]:
```
#matrix to hold our performance measures(pm)
pm = matrix(nrow = length(ntree),ncol = length(ptry))
```

In [19]:
```
for(i in 1:length(ptry)){
  for(j in 1:length(ntree)){

    RF = randomForest(cvascular.data[train,]$disease ~ .,
                      data = cvascular.data[train,],mtry = ptry[i],
                      ntree = ntree[j],importance = TRUE)



    yhat = predict(RF,newdata = cvascular.data[test,],type = "class")

    pm[j,i] = mrate(yhat,cvascular.data[test,]$disease)


  }
}
```

```
Warning message in randomForest.default(m, y, ...):
"invalid mtry: reset to within valid range"Warning message in randomForest.default
(m, y, ...):
"invalid mtry: reset to within valid range"Warning message in randomForest.default
(m, y, ...):
"invalid mtry: reset to within valid range"Warning message in randomForest.default
(m, y, ...):
"invalid mtry: reset to within valid range"
```

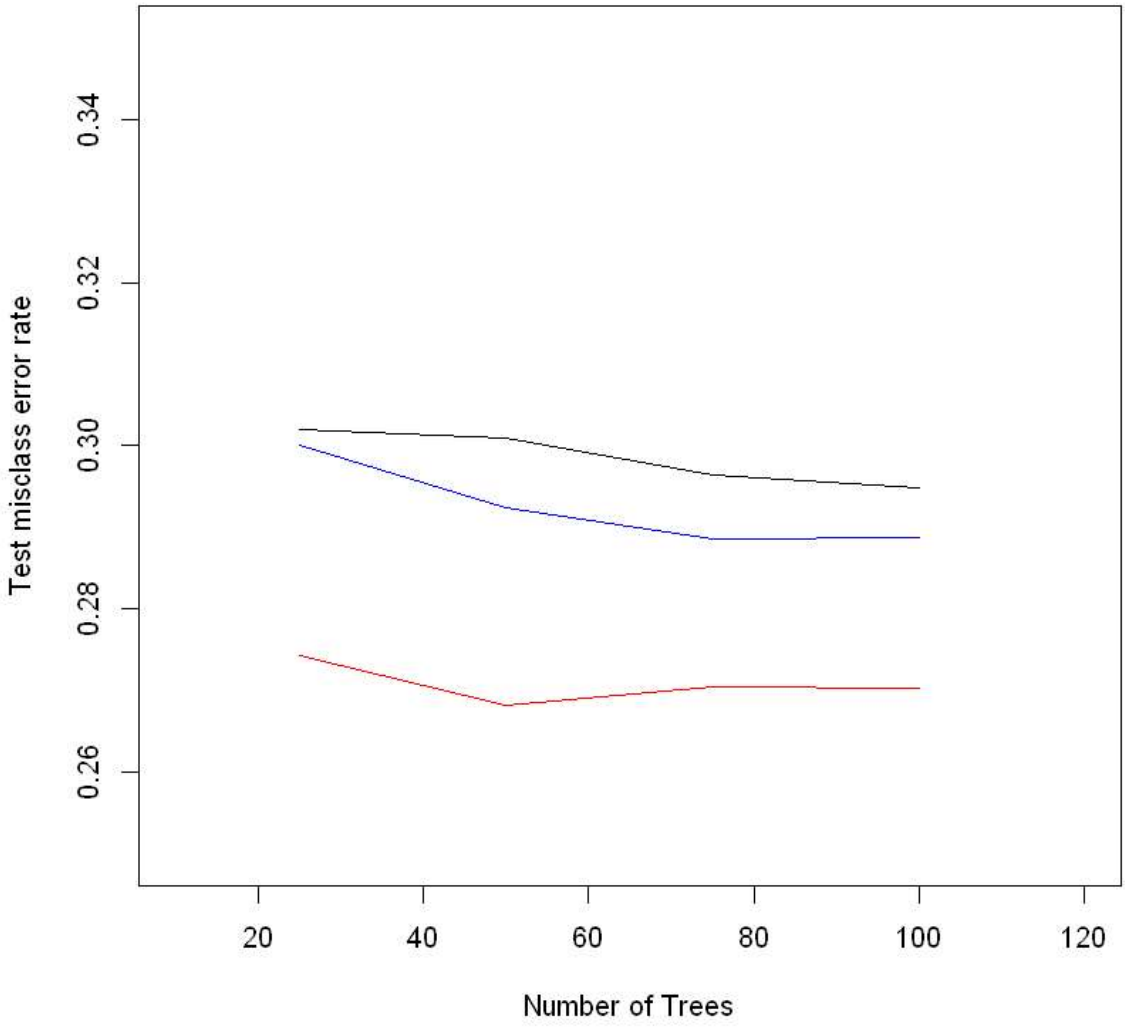
In [20]:
```
#matrix of misclassification error rates
pm
```

0.3020000	0.2742857	0.3000000
0.3010000	0.2681429	0.2924286
0.2964286	0.2704286	0.2885714
0.2948571	0.2702857	0.2887143

In [21]:
```
#Extract the best model along with its inputs
bestmod = which(pm == min(pm),arr.ind = TRUE)
print(paste("The best model has the following: misclass error",min(pm),
            ",number of trees :",ntree[bestmod[1]],
            ",number of features used at each split:",ptry[bestmod[2]]))
```

```
[1] "The best model has the following: misclass error 0.268142857142857 ,number of
trees : 50 ,number of features used at each split: 3"
```

In [22]:
```
#plots
plot(ntree,pm[,1],type = "l",ylim = c(0.25,0.35),xlab = "Number of Trees",
     ylab = "Test misclass error rate",xlim = c(10,120))
lines(ntree,pm[,2],type = "l",col="red")
lines(ntree,pm[,3],type = "l",col="blue")
```



In [ ]:
```
#We see that the models with mtry = ptry = sqrt(p)(-indicated by red line) have lo
#Specifically the model with 75 trees
```

In [ ]:
```
############################################################BOOSTING
```

In [23]:
```
#change target variable to character because gbm() fails to accept the target vari
cvascular.data$disease = as.character(cvascular.data$disease )
```

In [24]:
```
#grid of shrikage inputs to be used and a matrix(pm4) to hold misclassification er
shrinker = c(0.001,0.02,0.1)
pm4 = matrix(nrow = length(ntree),ncol = length(shrinker))
```

In [25]:
```
for(i in 1:length(shrinker)){
  for(j in 1:length(ntree)){

    Booster = gbm(disease~.,data = cvascular.data[train,],
                  distribution = "bernoulli",shrinkage = shrinker[i],
                  interaction.depth = 4,n.trees = ntree[j],verbose = TRUE)

    yhat4 = predict(Booster,newdata = cvascular.data[test,],n.trees = ntree[j],
                    type = "response")

    pm4[j,i] = mrate(yhat4,cvascular.data[test,]$disease)
```

```
  }
}
pm4
```

```
      Iter   TrainDeviance   ValidDeviance   StepSize   Improve
           1        1.3857             nan     0.0010    0.0002
           2        1.3852             nan     0.0010    0.0002
           3        1.3848             nan     0.0010    0.0002
           4        1.3843             nan     0.0010    0.0002
           5        1.3839             nan     0.0010    0.0002
           6        1.3834             nan     0.0010    0.0002
           7        1.3829             nan     0.0010    0.0002
           8        1.3825             nan     0.0010    0.0002
           9        1.3820             nan     0.0010    0.0002
          10        1.3816             nan     0.0010    0.0002
          20        1.3771             nan     0.0010    0.0002
          25        1.3749             nan     0.0010    0.0002

      Iter   TrainDeviance   ValidDeviance   StepSize   Improve
           1        1.3857             nan     0.0010    0.0002
           2        1.3852             nan     0.0010    0.0002
           3        1.3848             nan     0.0010    0.0002
           4        1.3843             nan     0.0010    0.0002
           5        1.3838             nan     0.0010    0.0002
           6        1.3834             nan     0.0010    0.0002
           7        1.3829             nan     0.0010    0.0002
           8        1.3825             nan     0.0010    0.0002
           9        1.3820             nan     0.0010    0.0002
          10        1.3816             nan     0.0010    0.0002
          20        1.3771             nan     0.0010    0.0002
          40        1.3684             nan     0.0010    0.0002
          50        1.3642             nan     0.0010    0.0002

      Iter   TrainDeviance   ValidDeviance   StepSize   Improve
           1        1.3857             nan     0.0010    0.0002
           2        1.3852             nan     0.0010    0.0002
           3        1.3847             nan     0.0010    0.0002
           4        1.3843             nan     0.0010    0.0002
           5        1.3838             nan     0.0010    0.0002
           6        1.3834             nan     0.0010    0.0002
           7        1.3829             nan     0.0010    0.0002
           8        1.3825             nan     0.0010    0.0002
           9        1.3820             nan     0.0010    0.0002
          10        1.3816             nan     0.0010    0.0002
          20        1.3772             nan     0.0010    0.0002
          40        1.3685             nan     0.0010    0.0002
          60        1.3602             nan     0.0010    0.0002
          75        1.3541             nan     0.0010    0.0002

      Iter   TrainDeviance   ValidDeviance   StepSize   Improve
           1        1.3857             nan     0.0010    0.0002
           2        1.3852             nan     0.0010    0.0002
           3        1.3848             nan     0.0010    0.0002
           4        1.3843             nan     0.0010    0.0002
           5        1.3838             nan     0.0010    0.0002
           6        1.3834             nan     0.0010    0.0002
           7        1.3829             nan     0.0010    0.0002
           8        1.3824             nan     0.0010    0.0002
           9        1.3820             nan     0.0010    0.0002
          10        1.3815             nan     0.0010    0.0002
          20        1.3771             nan     0.0010    0.0002
          40        1.3684             nan     0.0010    0.0002
          60        1.3602             nan     0.0010    0.0002
          80        1.3522             nan     0.0010    0.0002
         100        1.3445             nan     0.0010    0.0002

      Iter   TrainDeviance   ValidDeviance   StepSize   Improve
           1        1.3769             nan     0.0200    0.0045
```

```
         2        1.3681             nan     0.0200    0.0044
         3        1.3593             nan     0.0200    0.0042
         4        1.3514             nan     0.0200    0.0039
         5        1.3441             nan     0.0200    0.0037
         6        1.3367             nan     0.0200    0.0037
         7        1.3296             nan     0.0200    0.0035
         8        1.3224             nan     0.0200    0.0035
         9        1.3159             nan     0.0200    0.0033
        10        1.3094             nan     0.0200    0.0032
        20        1.2561             nan     0.0200    0.0022
        25        1.2352             nan     0.0200    0.0019

      Iter   TrainDeviance   ValidDeviance   StepSize   Improve
         1        1.3768             nan     0.0200    0.0045
         2        1.3682             nan     0.0200    0.0043
         3        1.3600             nan     0.0200    0.0040
         4        1.3519             nan     0.0200    0.0039
         5        1.3441             nan     0.0200    0.0039
         6        1.3367             nan     0.0200    0.0037
         7        1.3292             nan     0.0200    0.0036
         8        1.3223             nan     0.0200    0.0034
         9        1.3157             nan     0.0200    0.0033
        10        1.3093             nan     0.0200    0.0031
        20        1.2559             nan     0.0200    0.0022
        40        1.1921             nan     0.0200    0.0010
        50        1.1726             nan     0.0200    0.0009

      Iter   TrainDeviance   ValidDeviance   StepSize   Improve
         1        1.3768             nan     0.0200    0.0044
         2        1.3680             nan     0.0200    0.0042
         3        1.3597             nan     0.0200    0.0042
         4        1.3520             nan     0.0200    0.0038
         5        1.3445             nan     0.0200    0.0038
         6        1.3371             nan     0.0200    0.0037
         7        1.3300             nan     0.0200    0.0036
         8        1.3232             nan     0.0200    0.0034
         9        1.3165             nan     0.0200    0.0034
        10        1.3102             nan     0.0200    0.0032
        20        1.2566             nan     0.0200    0.0022
        40        1.1925             nan     0.0200    0.0011
        60        1.1574             nan     0.0200    0.0005
        75        1.1412             nan     0.0200    0.0003

      Iter   TrainDeviance   ValidDeviance   StepSize   Improve
         1        1.3773             nan     0.0200    0.0043
         2        1.3683             nan     0.0200    0.0043
         3        1.3596             nan     0.0200    0.0042
         4        1.3514             nan     0.0200    0.0041
         5        1.3437             nan     0.0200    0.0038
         6        1.3367             nan     0.0200    0.0036
         7        1.3292             nan     0.0200    0.0036
         8        1.3226             nan     0.0200    0.0034
         9        1.3159             nan     0.0200    0.0033
        10        1.3095             nan     0.0200    0.0032
        20        1.2563             nan     0.0200    0.0022
        40        1.1922             nan     0.0200    0.0012
        60        1.1578             nan     0.0200    0.0005
        80        1.1376             nan     0.0200    0.0004
       100        1.1256             nan     0.0200    0.0002

      Iter   TrainDeviance   ValidDeviance   StepSize   Improve
         1        1.3436             nan     0.1000    0.0217
         2        1.3071             nan     0.1000    0.0173
         3        1.2761             nan     0.1000    0.0146
```

```
         4        1.2531             nan     0.1000    0.0112
         5        1.2341             nan     0.1000    0.0092
         6        1.2162             nan     0.1000    0.0087
         7        1.2011             nan     0.1000    0.0072
         8        1.1880             nan     0.1000    0.0062
         9        1.1776             nan     0.1000    0.0046
        10        1.1695             nan     0.1000    0.0039
        20        1.1239             nan     0.1000    0.0010
        25        1.1141             nan     0.1000    0.0004

      Iter   TrainDeviance   ValidDeviance   StepSize   Improve
         1        1.3421             nan     0.1000    0.0217
         2        1.3066             nan     0.1000    0.0173
         3        1.2777             nan     0.1000    0.0141
         4        1.2533             nan     0.1000    0.0121
         5        1.2341             nan     0.1000    0.0095
         6        1.2173             nan     0.1000    0.0079
         7        1.2015             nan     0.1000    0.0076
         8        1.1884             nan     0.1000    0.0061
         9        1.1779             nan     0.1000    0.0049
        10        1.1705             nan     0.1000    0.0036
        20        1.1239             nan     0.1000    0.0009
        40        1.1017             nan     0.1000   -0.0000
        50        1.0963             nan     0.1000   -0.0001

      Iter   TrainDeviance   ValidDeviance   StepSize   Improve
         1        1.3438             nan     0.1000    0.0218
         2        1.3092             nan     0.1000    0.0177
         3        1.2802             nan     0.1000    0.0148
         4        1.2546             nan     0.1000    0.0124
         5        1.2344             nan     0.1000    0.0098
         6        1.2175             nan     0.1000    0.0082
         7        1.2029             nan     0.1000    0.0069
         8        1.1916             nan     0.1000    0.0057
         9        1.1821             nan     0.1000    0.0044
        10        1.1719             nan     0.1000    0.0050
        20        1.1250             nan     0.1000    0.0004
        40        1.1013             nan     0.1000   -0.0001
        60        1.0918             nan     0.1000   -0.0001
        75        1.0863             nan     0.1000   -0.0002

      Iter   TrainDeviance   ValidDeviance   StepSize   Improve
         1        1.3413             nan     0.1000    0.0220
         2        1.3066             nan     0.1000    0.0167
         3        1.2775             nan     0.1000    0.0149
         4        1.2532             nan     0.1000    0.0120
         5        1.2341             nan     0.1000    0.0094
         6        1.2176             nan     0.1000    0.0077
         7        1.2031             nan     0.1000    0.0067
         8        1.1895             nan     0.1000    0.0065
         9        1.1798             nan     0.1000    0.0045
        10        1.1712             nan     0.1000    0.0038
        20        1.1254             nan     0.1000    0.0008
        40        1.1026             nan     0.1000   -0.0001
        60        1.0928             nan     0.1000   -0.0002
        80        1.0859             nan     0.1000   -0.0002
       100        1.0795             nan     0.1000   -0.0003
```

0.02371429	0.0108571429	0.0001428571
0.02171429	0.0058571429	0.0001428571
0.02328571	0.0001428571	0.0001428571
0.03757143	0.0001428571	0.0001428571

In [26]:
```
#find the best model and its inputs
best.gbm = which(pm4 == min(pm4),arr.ind = TRUE)
best.gbm
```

row	col
3	2
4	2
1	3
2	3
3	3
4	3

In [27]:
```
paste("we see that some of the best models locations are row",best.gbm[1,1],
      "column",best.gbm[1,2]," and row",best.gbm[2,1],"column",best.gbm[2,2])
paste("Both models have the same misclassification error of",min(pm4))
paste("The first of these models has the following inputs: tress=",
      ntree[best.gbm[1,1]],"shrinkage =",shrinker[best.gbm[1,2]])
```

'we see that some of the best models locations are row 3 column 2 and row 4 column 2'

'Both models have the same misclassification error of 0.000142857142857143'

'The first of these models has the following inputs: tress= 75 shrinkage = 0.02'

In [ ]:
```
#In terms of number of trees we see that the best models have the number
#of trees which are at the end of our grid of number of trees used so,
#we will move the grid to see if any improvements can be made
```

In [28]:
```
#new.ntree = seq(from = 100,to = 200,by = 25)
#new.ntree = ntree[2:length(ntree)]
```

In [31]:
```
shrinker = c(0.1,0.2,0.3)
pm5 = matrix(nrow = length(new.ntree),ncol = length(shrinker))
```

In [32]:
```
for(i in 1:length(shrinker)){
  for(j in 1:length(new.ntree)){

    Booster2 = gbm(disease~.,data = cvascular.data[train,],
                   distribution = "bernoulli",shrinkage = shrinker[i],
                   interaction.depth = 4,n.trees = ntree[j],verbose = TRUE)

    yhat5 = predict(Booster2,newdata = cvascular.data[test,],n.trees = ntree[j],
                    type = "response")

    pm5[j,i] = mrate(yhat4,cvascular.data[test,]$disease)


  }
}
pm5
```

```
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3443             nan     0.1000    0.0210
     2        1.3081             nan     0.1000    0.0178
     3        1.2774             nan     0.1000    0.0150
     4        1.2531             nan     0.1000    0.0119
     5        1.2328             nan     0.1000    0.0099
     6        1.2166             nan     0.1000    0.0076
     7        1.2026             nan     0.1000    0.0069
     8        1.1900             nan     0.1000    0.0059
     9        1.1789             nan     0.1000    0.0051
    10        1.1702             nan     0.1000    0.0039
    20        1.1246             nan     0.1000    0.0006
    25        1.1158             nan     0.1000    0.0006

Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3417             nan     0.1000    0.0216
     2        1.3073             nan     0.1000    0.0175
     3        1.2779             nan     0.1000    0.0146
     4        1.2546             nan     0.1000    0.0119
     5        1.2340             nan     0.1000    0.0099
     6        1.2174             nan     0.1000    0.0083
     7        1.2021             nan     0.1000    0.0068
     8        1.1905             nan     0.1000    0.0056
     9        1.1801             nan     0.1000    0.0049
    10        1.1713             nan     0.1000    0.0040
    20        1.1251             nan     0.1000    0.0008
    40        1.1020             nan     0.1000   -0.0004
    50        1.0965             nan     0.1000   -0.0001

Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3422             nan     0.1000    0.0215
     2        1.3073             nan     0.1000    0.0166
     3        1.2764             nan     0.1000    0.0150
     4        1.2509             nan     0.1000    0.0120
     5        1.2311             nan     0.1000    0.0101
     6        1.2144             nan     0.1000    0.0080
     7        1.2008             nan     0.1000    0.0067
     8        1.1874             nan     0.1000    0.0062
     9        1.1776             nan     0.1000    0.0047
    10        1.1684             nan     0.1000    0.0043
    20        1.1245             nan     0.1000    0.0008
    40        1.1014             nan     0.1000    0.0002
    60        1.0920             nan     0.1000   -0.0000
    75        1.0867             nan     0.1000   -0.0002

Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3028             nan     0.2000    0.0408
     2        1.2484             nan     0.2000    0.0265
     3        1.2121             nan     0.2000    0.0177
     4        1.1872             nan     0.2000    0.0131
     5        1.1698             nan     0.2000    0.0078
     6        1.1540             nan     0.2000    0.0073
     7        1.1437             nan     0.2000    0.0046
     8        1.1357             nan     0.2000    0.0039
     9        1.1295             nan     0.2000    0.0026
    10        1.1236             nan     0.2000    0.0027
    20        1.1033             nan     0.2000    0.0001
    25        1.0989             nan     0.2000   -0.0008

Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3066             nan     0.2000    0.0404
     2        1.2516             nan     0.2000    0.0268
     3        1.2120             nan     0.2000    0.0187
     4        1.1863             nan     0.2000    0.0122
```

```
     5        1.1679             nan     0.2000    0.0088
     6        1.1554             nan     0.2000    0.0054
     7        1.1437             nan     0.2000    0.0043
     8        1.1366             nan     0.2000    0.0031
     9        1.1286             nan     0.2000    0.0032
    10        1.1240             nan     0.2000    0.0015
    20        1.1021             nan     0.2000   -0.0004
    40        1.0865             nan     0.2000   -0.0003
    50        1.0805             nan     0.2000   -0.0002

Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3027             nan     0.2000    0.0421
     2        1.2468             nan     0.2000    0.0258
     3        1.2087             nan     0.2000    0.0185
     4        1.1837             nan     0.2000    0.0117
     5        1.1640             nan     0.2000    0.0092
     6        1.1500             nan     0.2000    0.0065
     7        1.1403             nan     0.2000    0.0039
     8        1.1324             nan     0.2000    0.0027
     9        1.1269             nan     0.2000    0.0022
    10        1.1219             nan     0.2000    0.0019
    20        1.1000             nan     0.2000   -0.0005
    40        1.0854             nan     0.2000   -0.0002
    60        1.0748             nan     0.2000   -0.0008
    75        1.0676             nan     0.2000   -0.0005

Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.2708             nan     0.3000    0.0578
     2        1.2092             nan     0.3000    0.0321
     3        1.1727             nan     0.3000    0.0169
     4        1.1529             nan     0.3000    0.0082
     5        1.1376             nan     0.3000    0.0065
     6        1.1295             nan     0.3000    0.0036
     7        1.1227             nan     0.3000    0.0022
     8        1.1179             nan     0.3000    0.0012
     9        1.1150             nan     0.3000    0.0002
    10        1.1113             nan     0.3000    0.0007
    20        1.0945             nan     0.3000   -0.0007
    25        1.0905             nan     0.3000   -0.0007

Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.2694             nan     0.3000    0.0568
     2        1.2032             nan     0.3000    0.0306
     3        1.1698             nan     0.3000    0.0159
     4        1.1479             nan     0.3000    0.0102
     5        1.1348             nan     0.3000    0.0048
     6        1.1244             nan     0.3000    0.0041
     7        1.1187             nan     0.3000    0.0021
     8        1.1151             nan     0.3000    0.0004
     9        1.1109             nan     0.3000    0.0005
    10        1.1094             nan     0.3000   -0.0005
    20        1.0945             nan     0.3000   -0.0015
    40        1.0779             nan     0.3000   -0.0009
    50        1.0710             nan     0.3000   -0.0007

Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.2699             nan     0.3000    0.0580
     2        1.2109             nan     0.3000    0.0293
     3        1.1785             nan     0.3000    0.0144
     4        1.1544             nan     0.3000    0.0108
     5        1.1384             nan     0.3000    0.0066
     6        1.1281             nan     0.3000    0.0040
     7        1.1214             nan     0.3000    0.0021
     8        1.1173             nan     0.3000    0.0016
```

```
  9        1.1141            nan     0.3000   -0.0002
 10        1.1124            nan     0.3000   -0.0004
 20        1.0983            nan     0.3000   -0.0001
 40        1.0808            nan     0.3000   -0.0005
 60        1.0672            nan     0.3000   -0.0018
 75        1.0619            nan     0.3000   -0.0009
```

0.0001428571	0.0001428571	0.0001428571
0.0001428571	0.0001428571	0.0001428571
0.0001428571	0.0001428571	0.0001428571

In [42]:
```
best.gbm1 = which(pm5 == min(pm5),arr.ind = TRUE)
best.gbm1
```

row	col
1	1
2	1
3	1
1	2
2	2
3	2
1	3
2	3
3	3

In [ ]:
```
#we see no further improvements in the models
```

In [33]:
```
#####SUPPORT VECTOR MACHINES#########
cvascular.data$disease = as.factor(cvascular.data$disease)
```

In [34]:
```
#vector of different inputs to try to find the best model
kern = c("linear","polynomial","radial")
costs = c(0.1,1,10)
garmer = c(1,50)
```

In [35]:
```
#vector to hold miscalssification error rates
pv1 = rep(0,length(costs))
```

In [36]:
```
for(l in 1:length(costs)){

  svm1 = svm(disease~.,data = cvascular.data[train,],kernel = kern[1],cost = costs

  yhat1 = predict(svm1,newdata = cvascular.data[test,])

  pv1[l] =  mrate(yhat1,cvascular.data[test,]$disease)

}
```

In [37]:
```
######polynomial kernel
pv2 = rep(0,length(costs))
```

In [38]:
```
for(k in 1:length(costs)){

  svm2 = svm(disease~.,data = cvascular.data[train,],kernel = kern[2],cost = costs

  yhat2 = predict(svm2,newdata = cvascular.data[test,])

  pv2[k] = mrate(yhat2,cvascular.data[test,]$disease)


}
```

In [39]:
```
#######radial kernel
pv3 = matrix(nrow = length(costs), ncol = length(garmer))
```

In [45]:
```
for(n in 1:length(garmer)){
  for(m in 1:length(costs)){

    svm3 = svm(disease~.,data = cvascular.data[train,],kernel = kern[3],cost = cos

    yhat3 = predict(svm1,newdata = cvascular.data[test,])

    pv3[m,n] =  mrate(yhat1,cvascular.data[test,]$disease)


  }
}
```

In [ ]:
```
#compare performance results among the svm models with different kernels
```

In [41]:
```
pv1
best.mod1 = which(pv1 == min(pv1))
best.mod1
```

1. 0.285285714285714
2. 0.274571428571429
3. 0.274

3

In [42]:
```
pv2
best.mod2 = which(pv2 == min(pv2))
best.mod2
```

1. 0.396571428571429
2. 0.359714285714286
3. 0.312857142857143

3

In [46]:
```
pv3
best.mod3 = which(pv3 == min(pv3))
best.mod3
```

0.274	0.274
0.274	0.274
0.274	0.274

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6

In [ ]:
```
#we se that the model with the radial kernel performs best across all inputs
```

In [47]:
```
#we see that the model with the linear(and cost = 0.1 & 1) kernel performs best as
#let us use cross validation to compare performance
best.mod1 = which(pv1 == min(pv1))
best.mod1
```

3

In [48]:
```
svm.cv = svm(disease~.,data = cvascular.data[train,],kernel = kern[1],
             cost = costs)
```

In [49]:
```
tune.out = tune(svm,disease~.,data = cvascular.data[train,],kernel = kern[1],
                ranges = list(cost = costs))
```

In [50]:
```
summary(tune.out)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
   10

- best performance: 0.2821429

- Detailed performance results:
  cost     error dispersion
1  0.1 0.2917143 0.02037962
2  1.0 0.2844286 0.01509531
3 10.0 0.2821429 0.01501700
```

In [51]:
```
pred = predict(tune.out$best.model, newdata = cvascular.data[test, ])
```

In [52]:
```
mrate(pred,cvascular.data[test,]$disease)
```

0.274

In [ ]:
```
#10 fold cross validation performs just as well.
```

In [ ]:
```
#seems that Boosting performed best.
```