final

Ruipeng Wei

4/27/2019

library package

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.4.4
## Loading required package: lattice
## Warning: package 'lattice' was built under R version 3.4.4
## Loading required package: ggplot2
```

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.4.4
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.4.4
## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.4.4
## Loaded gbm 2.1.5
```

```
library(ape)
```

```
## Warning: package 'ape' was built under R version 3.4.4
```

read in data

```
test <- read.table("testset",header=T)
train <- read.table("trainset",header=T)
```

```
sum(complete.cases(test))==nrow(test)
```

```
## [1] TRUE
```

```
sum(complete.cases(train))==nrow(train)
```

```
## [1] TRUE
```

No missing data in the two dataset.

```
## 'data.frame':    1919 obs. of  55 variables:
##  $ V1 : num  0.1681 0.0966 0.0232 0.054 0.0512 ...
##  $ V2 : num  0.4365 0.0828 0.39 0.4767 0.5784 ...
##  $ V3 : num  0.438 0.62 0.612 0.294 0.166 ...
##  $ V4 : num  2.24 8.49 2.58 2.22 1.33 ...
##  $ V5 : num  18.345 59.536 12.67 4.889 0.553 ...
##  $ V6 : num  0 0.21414 0.000228 0 0 ...
##  $ V7 : num  0.2105 0.118 0.0296 0.0687 0.0717 ...
##  $ V8 : num  1.291 11.066 1.564 1.098 0.729 ...
##  $ V9 : num  2.58 1.04 1.79 1.47 1.95 ...
##  $ V10: num  0.564 0.917 0.61 0.523 0.422 ...
##  $ V11: num  0.2445 0.118 0.0895 0.1076 0.1086 ...
##  $ V12: num  0.5939 1.4244 0.0764 0.285 0.1443 ...
##  $ V13: num  0.1041 0.045 0.0168 0.1015 0.0678 ...
##  $ V14: num  0.2105 0.118 0.0296 0.0687 0.0717 ...
##  $ V15: num  593 200 4734 1164 1600 ...
##  $ V16: num  0.6152 1.8256 0.0771 0.3137 0.2281 ...
##  $ V17: num  2.29 12.07 2.56 2.1 1.73 ...
##  $ V18: num  0.2105 0.118 0.0296 0.0687 0.0717 ...
##  $ V19: num  0.0816 0.0351 0.0166 0.0466 0.0368 ...
##  $ V20: num  44.3 25.8 112.4 67.8 26.8 ...
##  $ V22: num  0.2439 0.1199 0.0559 0.1068 0.1014 ...
##  $ V23: num  0.0652 0.0287 0.013 0.0367 0.0263 ...
##  $ V25: num  0.519 0.917 0.589 0.137 0.395 ...
##  $ V26: num  0.5182 1.5677 0.0607 0.283 0.1927 ...
##  $ V29: num  3.69 4.16 3.39 3.71 4.05 ...
##  $ V30: num  0.1691 -0.0298 0.1287 0.2829 0.2946 ...
##  $ V31: num  0.0856 0.0351 0.0166 0.0616 0.0431 ...
##  $ V32: num  54.23 9.39 81.08 64.16 98.74 ...
##  $ V33: num  6.73 38.87 4.5 5.69 3.7 ...
##  $ V34: num  5.46 1.45 4.47 2.88 3.17 ...
##  $ V35: num  0.1949 0.1199 0.0424 0.1026 0.1105 ...
##  $ V36: num  2.58 3.37 1.79 1.47 1.95 ...
##  $ V38: num  0.61 0.917 0.61 0.759 0.488 ...
##  $ V39: num  0.0755 0.0356 0.0237 0.0696 0.0567 ...
##  $ V40: num  0.0187 0.4745 0.4132 0.2586 0.0512 ...
##  $ V41: num  0.0482 0.0178 0.2304 0.0847 0.1193 ...
##  $ V42: num  0.0945 0.0356 0.0313 0.0725 0.0521 ...
##  $ V43: num  111.2 72.1 171.5 117.1 119.5 ...
##  $ V44: num  66.9 46.3 59 49.2 92.7 ...
##  $ V46: num  1.35 5.62 1.16 1.08 1.05 ...
##  $ V47: num  48 26.9 115.2 72.9 28.4 ...
##  $ V48: num  0.1859 0.0866 0.0555 0.0259 0.0411 ...
##  $ V49: num  0.072 0.0258 0.031 0.0176 0.0211 ...
##  $ V50: num  1.82 8.49 2.56 1.12 1.15 ...
##  $ V51: num  0.3544 0.0828 0.3877 0.2409 0.4968 ...
##  $ V52: num  0.1486 0.0257 0.2221 0.1758 0.2705 ...
##  $ V55: num  2162 9069 1495 1520 1865 ...
##  $ V56: num  0.0755 0.0428 0.0237 0.0696 0.0567 ...
##  $ V57: num  0.298 0.105 0.038 0.103 0.121 ...
##  $ V58: num  0.921 0.957 0.984 0.954 0.964 ...
```

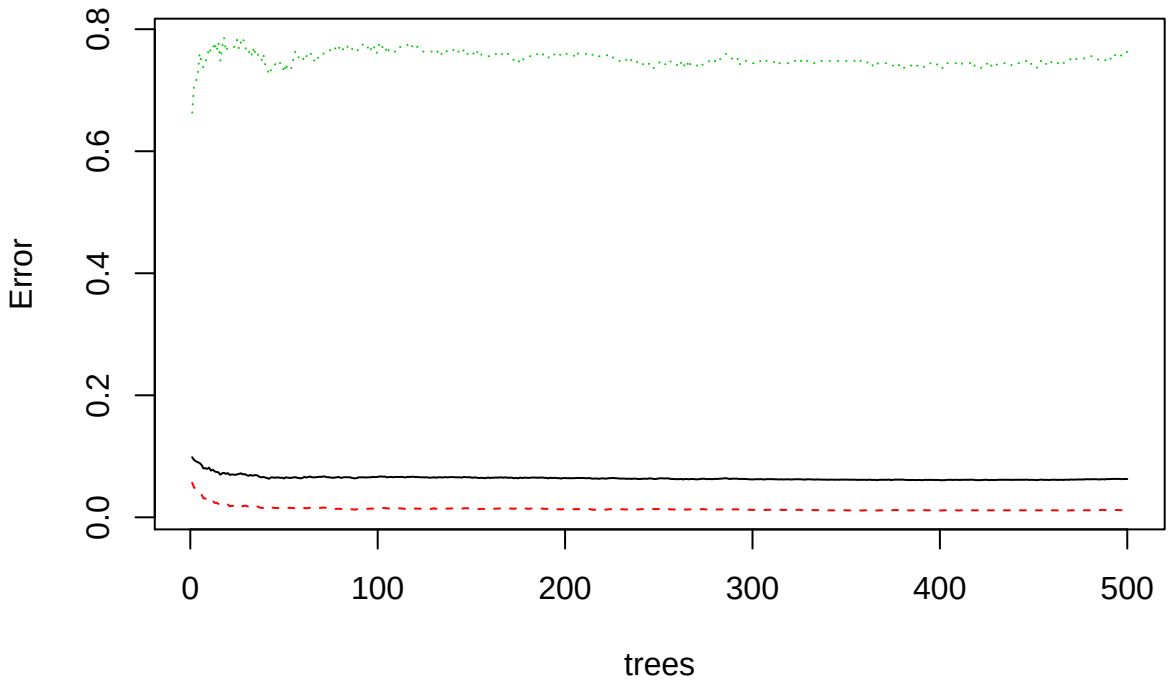
```
##  $ V59: num  0.083 0 0 0.45 0.158 ...
##  $ V61: num  5.46 7.89 6.18 7.42 3.94 ...
##  $ V62: num  50.13 8.99 79.16 59.69 93.14 ...
##  $ V63: num  7.28 40.61 4.61 6.11 3.92 ...
##  $ V65: int  0 0 0 0 0 0 1 0 0 0 ...
```

```
## 'data.frame':    3838 obs. of  55 variables:
##  $ V1 : num  0.0882 -0.0062 -0.09 0.0482 0.2316 ...
##  $ V2 : num  0.555 0.485 0.887 0.55 0.51 ...
##  $ V3 : num  0.0113 0.233 0.2693 0.1076 0.4729 ...
##  $ V4 : num  1.02 1.6 1.52 1.24 1.94 ...
##  $ V5 : num  -66.52 6.18 -55.99 -22.96 15.1 ...
##  $ V6 : num  0.342 0 -0.074 0 0 ...
##  $ V7 : num  0.1095 -0.0062 -0.09 0.0593 0.2877 ...
##  $ V8 : num  0.578 1.063 0.127 0.817 0.959 ...
##  $ V9 : num  1.09 1.28 1.28 1.51 1.79 ...
##  $ V10: num  0.32 0.515 0.113 0.45 0.49 ...
##  $ V11: num  0.10949 0.00133 -0.08079 0.09199 0.33362 ...
##  $ V12: num  0.198 -0.016 -0.174 0.134 0.571 ...
##  $ V13: num  0.0969 0.0375 0.084 0.0556 0.1669 ...
##  $ V14: num  0.1095 -0.0062 -0.09 0.0593 0.2877 ...
##  $ V15: num  1475 3693 3020 2386 623 ...
##  $ V16: num  0.2474 0.0988 0.1208 0.153 0.5857 ...
##  $ V17: num  1.8 2.06 1.13 1.82 1.96 ...
##  $ V18: num  0.1095 -0.0062 -0.09 0.0593 0.2877 ...
##  $ V19: num  0.07729 -0.00486 -0.07053 0.03913 0.16059 ...
##  $ V20: num  50.2 59.9 47.7 36.1 79.2 ...
##  $ V22: num  0.13523 0.00129 0 0.06762 0.32983 ...
##  $ V23: num  0.06229 -0.00486 -0.07053 0.0318 0.12926 ...
##  $ V25: num  0.3204 0.0803 0.0647 0.4292 0.4862 ...
##  $ V26: num  0.2091 0.0988 0.1208 0.1328 0.4758 ...
##  $ V29: num  6.13 4 4.02 5.88 4.17 ...
##  $ V30: num  0.378 0.364 0.658 0.363 0.173 ...
##  $ V31: num  0.077287 0.000778 -0.13865 0.039129 0.16679 ...
##  $ V32: num  155 108 254 140 126 ...
##  $ V33: num  2.35 3.38 1.44 2.66 2.9 ...
##  $ V34: num  0.244 2.708 0.836 2.134 2.864 ...
##  $ V35: num  0.1352 -0.0365 0.014 0.3642 0.3297 ...
##  $ V36: num  1.45 1.28 1.28 1.51 1.79 ...
##  $ V38: num  0.321 0.594 0.438 0.512 0.496 ...
##  $ V39: num  0.0955 -0.0286 0.011 0.2404 0.184 ...
##  $ V40: num  0.1288 0.0578 0.4562 0.0673 0.444 ...
##  $ V41: num  0.1119 0.2917 0.15 0.1983 0.0499 ...
##  $ V42: num  0.09546 0.00101 0 0.04463 0.18411 ...
##  $ V43: num  127 171 157 125 153 ...
##  $ V44: num  77.1 111.5 109.6 89.1 74.2 ...
##  $ V46: num  0.669 1.061 1.199 0.905 1.167 ...
##  $ V47: num  54.6 58.3 82.1 46.5 97.1 ...
##  $ V48: num  0.1075 -0.0528 -0.1971 0.0427 0.3185 ...
##  $ V49: num  0.0759 -0.0414 -0.1546 0.0282 0.1778 ...
##  $ V50: num  1.019 1.282 0.885 0.998 1.913 ...
##  $ V51: num  0.554 0.388 0.516 0.442 0.503 ...
##  $ V52: num  0.426 0.296 0.696 0.376 0.344 ...
##  $ V55: num  15182 2341.8 2789.6 1.24 7008.8 ...
```

```
##  $ V56: num  0.081 -0.0286 0.4188 0.2404 0.184 ...
##  $ V57: num  0.275 -0.012 -0.796 0.107 0.473 ...
##  $ V58: num  0.919 1.005 0.591 0.77 0.84 ...
##  $ V59: num  0.00202 0.15222 2.8787 0.13938 0.01424 ...
##  $ V61: num  4.73 3.27 3.33 4.09 4.92 ...
##  $ V62: num  143 111 148 106 103 ...
##  $ V63: num  2.56 3.28 2.47 3.43 3.56 ...
##  $ V65: int  0 0 0 0 0 0 0 0 0 0 ...
```

The 55th column, which colname is V65 is the outcome - bankrupt or not. #supervised learning ##random forest

```
train$V65 <- as.factor(train$V65)
rf1 <- randomForest(V65 ~ ., data = train)

matplot(rf1$err.rate, type='l', xlab='trees', ylab='Error')
```




```
table(train$V65,predict(rf1))
```

```
##
##        0    1
##   0 3535   41
##   1  200   62
```

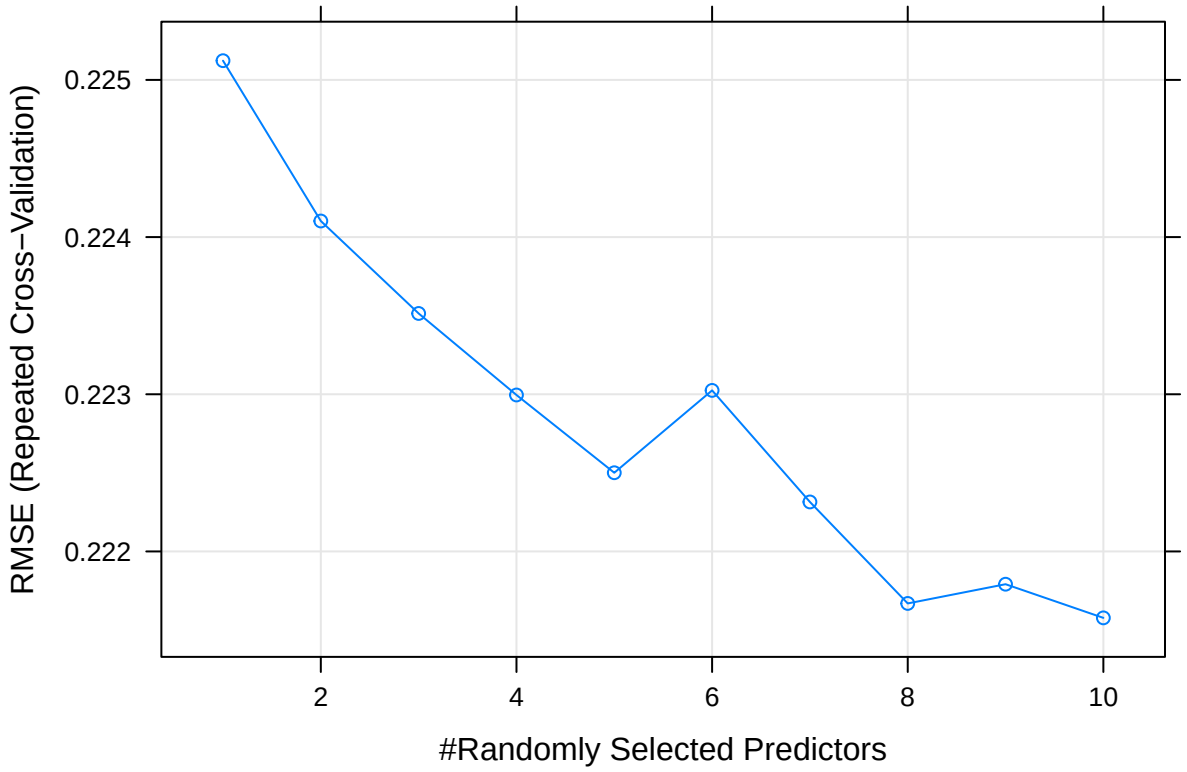
```
pre_train_rf1 <- table(train$V65,predict(rf1))

#accuracy in train data
sum(diag(pre_train_rf1))/sum(pre_train_rf1)
```

```
## [1] 0.9372069
```

The accuracy in train dataset in random forest model is about 93.56%. And according to the error plot, the error rate becomes stable when the number of trees be about 100. Thus, I will use ntree=100 is the following analysis.

In order to improve the performance of random forest, I will try cross validation to tune the hyperparameter.

```
cv <- trainControl(method="repeatedcv", number=10, repeats=5, classProbs=TRUE)
train$V65 <- as.numeric(as.character(train$V65))
rf2 <- train(x=train[,c(1:54)], y=train$V65, trControl=cv,tuneGrid=data.frame(mtry=1:10), method="rf", n
plot(rf2)
```



From the plot, we could know that then mtry=10, the RMSE has the lowest value.

```
rf2
```

```
## Random Forest
##
## 3838 samples
##   54 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 3454, 3454, 3454, 3454, 3455, 3454, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE       Rsquared   MAE
##    1    0.2251223  0.2047674  0.1030025
##    2    0.2241024  0.2133946  0.1026242
##    3    0.2235142  0.2177987  0.1023252
##    4    0.2229954  0.2217404  0.1019534
##    5    0.2225008  0.2260857  0.1017376
##    6    0.2230246  0.2230543  0.1020890
##    7    0.2223149  0.2277030  0.1016227
##    8    0.2216692  0.2312187  0.1011568
##    9    0.2217916  0.2308567  0.1010911
##   10    0.2215774  0.2322766  0.1008075
```

```
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 10.
```

```
rf2$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = 100, mtry = param$mtry)
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 0.04900222
##                     % Var explained: 22.96
```

```
table(train$V65,round(predict(rf2)))
```

```
##
##        0    1
##   0 3576    0
##   1    6  256
```

```
pre_train_rf2 <- table(train$V65,round(predict(rf2)))

#accuracy in train data
sum(diag(pre_train_rf2))/sum(pre_train_rf2)
```

```
## [1] 0.9984367
```

The accuracy of the trainin dataset is arriving about 99.90% after improving by cross validation. The accuracy is too good to be true, maybe it is overfitted. Let's see the accuracy in test data.

```
table(round(predict(rf2$finalModel, test[,c(1:54)])),test$V65)
```

```
##
##        0    1
##   0 1754  112
##   1   23   30
```

```
pre_test_rf2 <- table(round(predict(rf2$finalModel,test[,c(1:54)])),test$V65)
#accuracy in test data
sum(diag(pre_test_rf2))/sum(pre_test_rf2)
```

```
## [1] 0.9296509
```

The accuracy of this model in test dataset is about 92.97%.

boosting

```
bt1 = gbm(V65 ~ ., data=train, distribution="gaussian", n.trees=500)
bt1
```

```
## gbm(formula = V65 ~ ., distribution = "gaussian", data = train,
##     n.trees = 500)
## A gradient boosted model with gaussian loss function.
## 500 iterations were performed.
## There were 54 predictors of which 51 had non-zero influence.
```

```r
#condusion matirx in train
table(train$V65,round(predict(bt1,train,n.trees=500)))
```

```
##
##        0    1
##   0 3560   16
##   1  159  103
```

```r
pre_train_bt1 <- table(train$V65,round(predict(bt1,train,n.trees=500)))
#accuracy in train
sum(diag(pre_train_bt1))/sum(pre_train_bt1)
```

```
## [1] 0.9544033
```

The accuracy of the boosting tree in training dataset is about 95.54%. Let's use cross validation to tune the hyperparameter in boosting.

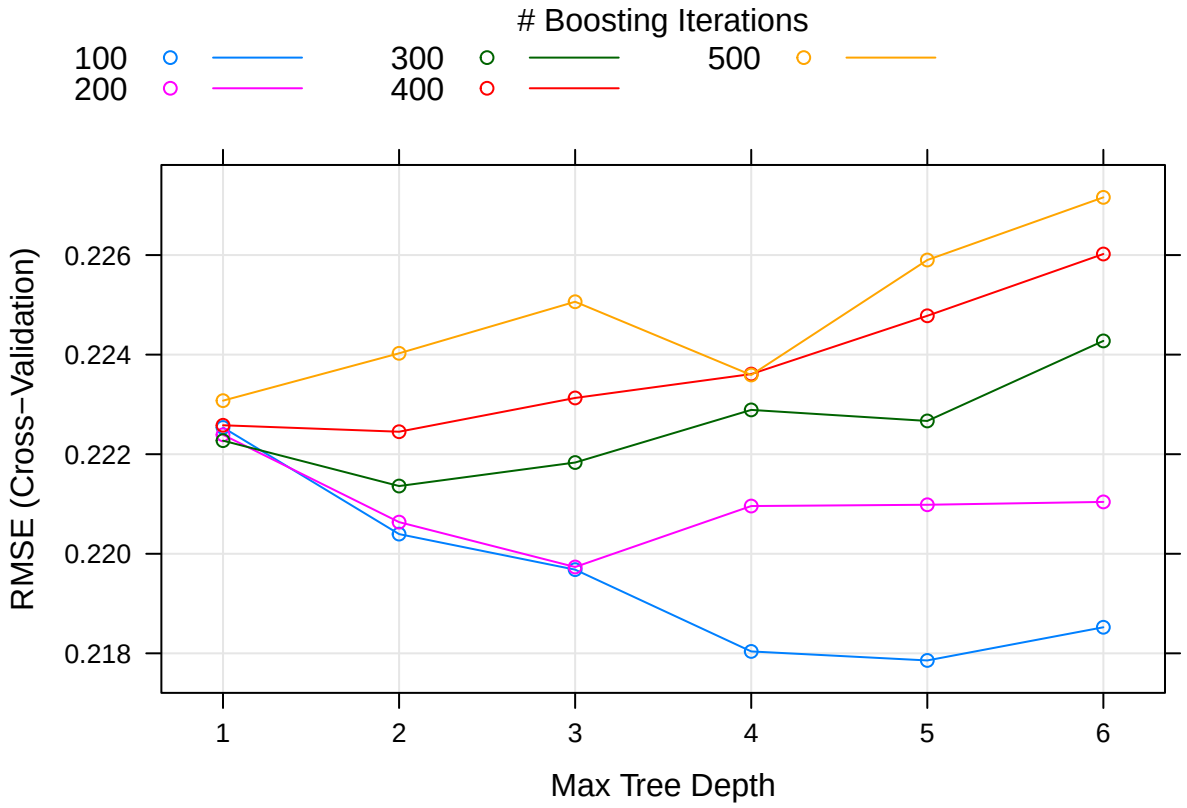
```r
ctr = trainControl(method="cv", number=10)
mygrid = expand.grid(n.trees=seq(100, 500, 100), interaction.depth=1:6,
shrinkage=0.1, n.minobsinnode=10)
bt2 <- train(V65 ~ ., train, method='gbm',
trControl=ctr, tuneGrid=mygrid,
preProc=c('center','scale'), verbose=F)
```

```
## Warning in train.default(x, y, weights = w, ...): You are trying to do
## regression and your outcome only has two possible values Are you trying to
## do classification? If so, use a 2 level factor as your outcome column.
```

```r
bt2$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 21     100                 5       0.1             10
```

```r
plot(bt2)
```



Accroding to the cross validation result, the model has the best performance when the interaction.depth = 3, n.trees = 200, and n.minobsinode = 10.

```
bt3 = gbm(V65 ~ ., data=train, distribution="gaussian", n.trees=200, interaction.depth = 3, n.minobsinn
bt3
```

```
## gbm(formula = V65 ~ ., distribution = "gaussian", data = train,
##     n.trees = 200, interaction.depth = 3, n.minobsinnode = 10)
## A gradient boosted model with gaussian loss function.
## 200 iterations were performed.
## There were 54 predictors of which 51 had non-zero influence.
```

```
#confusion matirx in train
table(train$V65,round(predict(bt3,train,n.trees=200)))
```

```
##
##        0    1
##   0 3572    4
##   1  129  133
```

```
pre_train_bt3 <- table(train$V65,round(predict(bt3,train,n.trees=200)))
#accuracy in train
sum(diag(pre_train_bt3))/sum(pre_train_bt3)
```

```
## [1] 0.9653465
```

The accuracy in traning dataset improves to about 96.51% about tuning the hypermeters by cross validation.

```
#confusion matirx in test
table(test$V65,round(predict(bt3,test,n.trees=200)))
```

```
##
```

```
##        0    1
##   0 1761   16
##   1  106   36
```

```
pre_test_bt3 <- table(test$V65,round(predict(bt3,test,n.trees=200)))
#accuracy in test
sum(diag(pre_test_bt3))/sum(pre_test_bt3)
```

```
## [1] 0.9364252
```

Using the tuned boosting model, the accuracy in test dataset is about 93.59%.

SVM

```
train$V65 <- as.factor(train$V65)
sf1 = svm(V65 ~ ., data=train, kernel='linear')
sf1
```

```
##
## Call:
## svm(formula = V65 ~ ., data = train, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.01851852
##
## Number of Support Vectors:  529
```

```
#confusion matirx in train
table(train$V65, sf1$fitted)
```

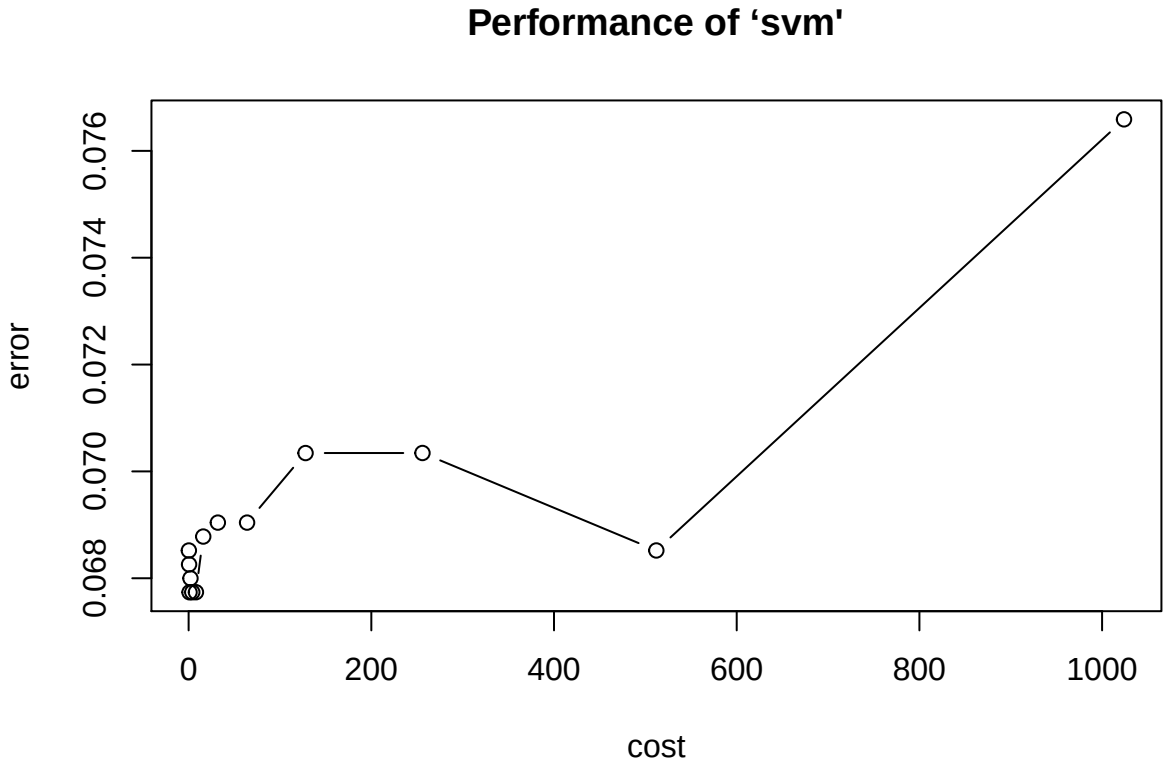
```
##
##        0    1
##   0 3575    1
##   1  249   13
```

```
pre_train_sf1 <- table(train$V65, sf1$fitted)
#accuracy in train
sum(diag(pre_train_sf1)/sum(pre_train_sf1))
```

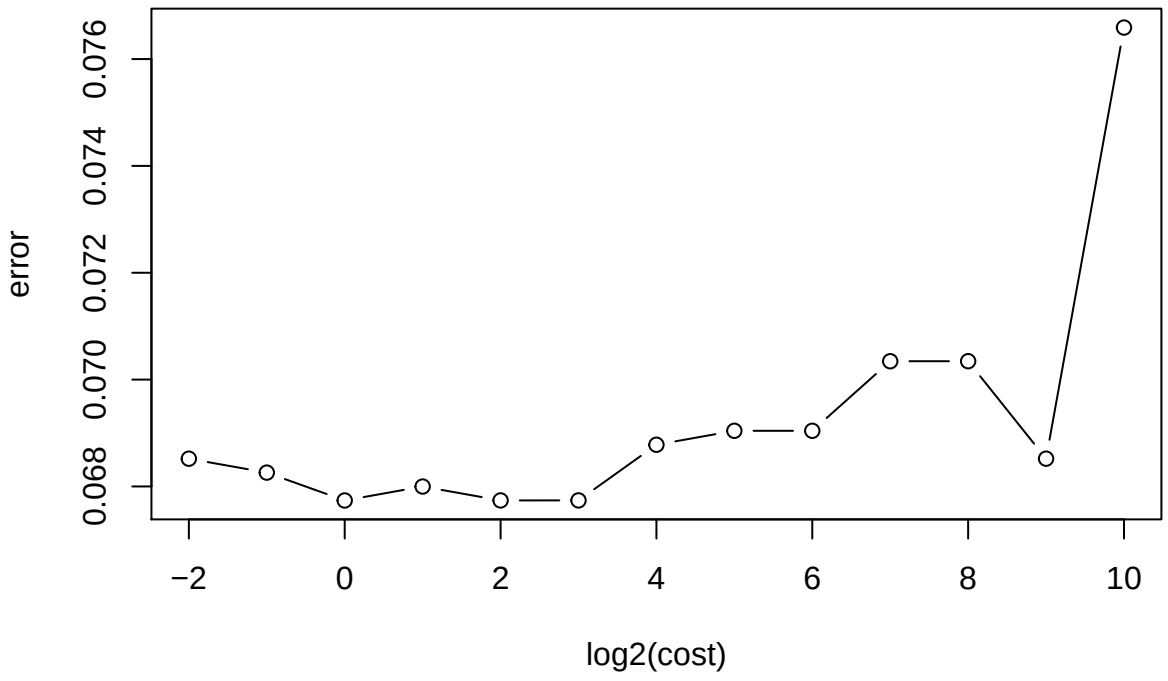
```
## [1] 0.9348619
```

The accuracy of SVM in test dataset is about 93.49% using default setting.

```
sf2 = tune(svm, V65 ~ ., data=train, kernel='linear',
ranges = list(cost = 2^(-2:10)),
tunecontrol = tune.control(nrepeat=5, cross=10))
plot(sf2, transform.x=log2)
```




```
with(sf2$performances, plot(log2(cost), error, type='b'))
```




```
sf2$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = V65 ~ ., data = train, ranges = list(cost = 2^(-2:10)),
##     tunecontrol = tune.control(nrepeat = 5, cross = 10), kernel = "linear")
##
```

```
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.01851852
##
## Number of Support Vectors:  529
```

From the plot and the cross validation result, we could know that when cost = 64, the model has the smallest error rate.

```
#confusion matirx in train
table(train$V65, sf2$best.model$fitted)
```

```
##
##        0    1
##   0 3575    1
##   1  249   13
```

```
pre_train_sf2 <- table(train$V65, sf2$best.model$fitted)

#accuracy in train
sum(diag(pre_train_sf2)/sum(pre_train_sf2))
```

```
## [1] 0.9348619
```

The accuracy in training data after cross validation is about 93.67%, only having a little improvement compared with the default setting.

```
table(test$V65, predict(sf2$best.model, test))
```

```
##
##        0    1
##   0 1775    2
##   1  138    4
```

```
pre_test_sf2 <- table(test$V65, predict(sf2$best.model, test))
sum(diag(pre_test_sf2)/sum(pre_test_sf2))
```

```
## [1] 0.9270453
```

The accuracy in test dataset applied with tuned SVM is about 92.50%.

summary

I tried three supervised models and tuned these three based cross validation. Tuned randomForest mode has the best performance in training dataset with an accuracy over 99%. But tuned boosting has the best performance with the accuracy about 93.6%. Thus, random forest may be overfitted in the training dataset. In general, the three tuned models have very similar accuracy in test dataset.

unsupervised learning

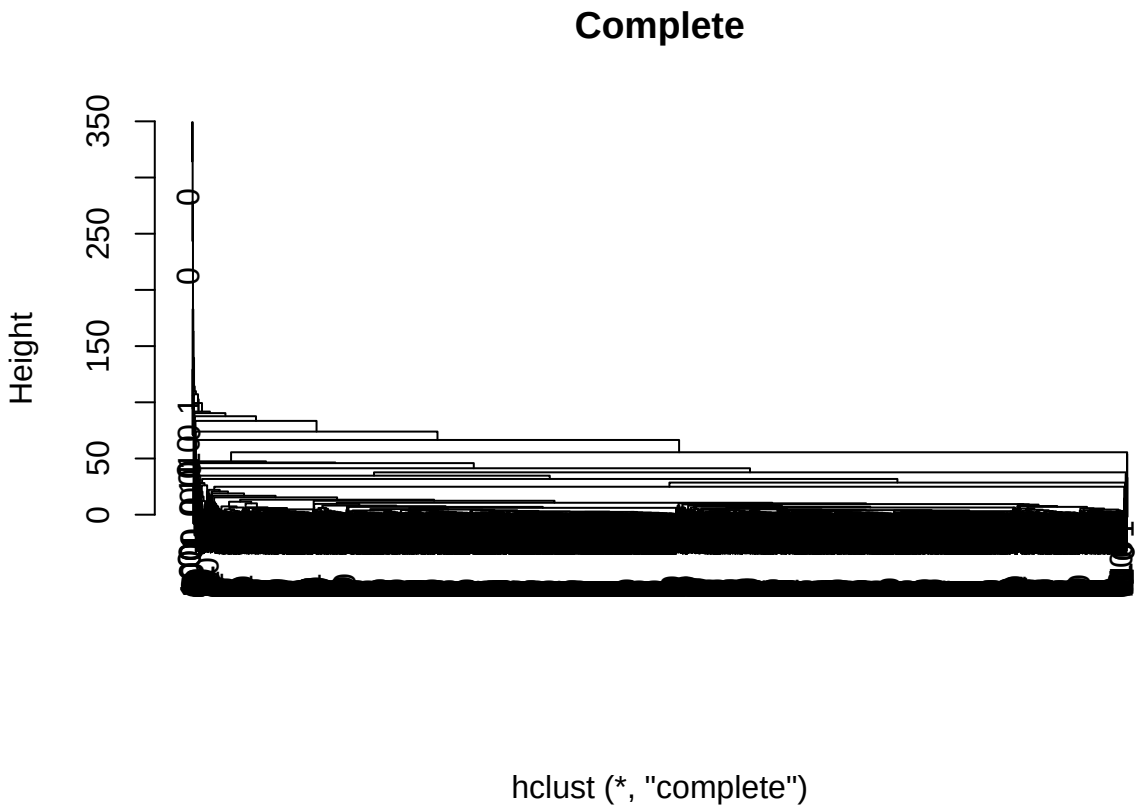
cluster

```
all <- rbind(test,train)
all_std <- scale(all[,1:54])
all_dist <- dist(all_std)
```

Combine the two dataset, train and test, into a big datset.

```
clust1 = hclust(all_dist)
plot(clust1, xlab='', main="Complete", horiz=T, labels = all$V65)
```

```
## Warning in graphics:::plotHclust(n1, merge, height, order(x$order), hang, :
## "horiz" is not a graphical parameter

## Warning in graphics:::plotHclust(n1, merge, height, order(x$order), hang, :
## "horiz" is not a graphical parameter

## Warning in axis(2, at = pretty(range(height)), ...): "horiz" is not a
## graphical parameter

## Warning in title(main = main, sub = sub, xlab = xlab, ylab = ylab, ...):
## "horiz" is not a graphical parameter
```




```
pdf("rplot.pdf",width = 20,height = 100)
plot(as.phylo(clust1), cex = 0.6, label.offset = 0.5,label=all$V65)
```

```
## Warning in plot.window(...): "label" is not a graphical parameter
```

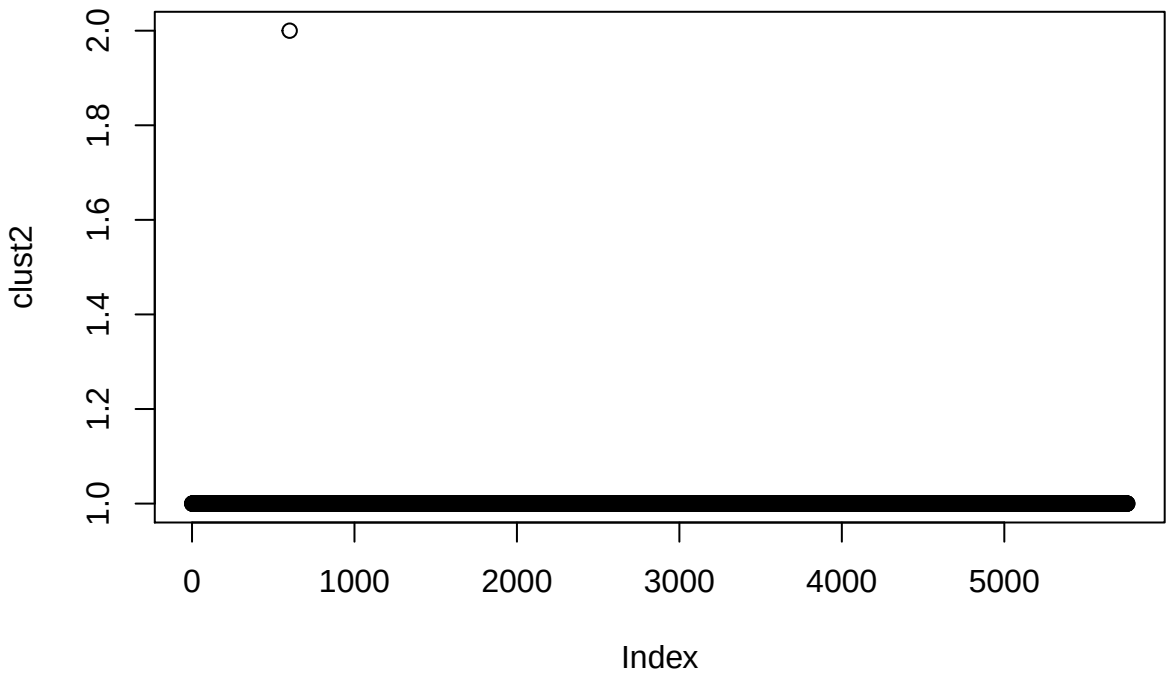
```
## Warning in plot.xy(xy, type, ...): "label" is not a graphical parameter
## Warning in title(...): "label" is not a graphical parameter
```

```
dev.off()
```

```
## pdf
##   2
```

There is no way to see the label clearly, and we cannot tell the cluster result from the plot.

```
clust2 <-cutree(hclust(all_dist),k=2)
plot(clust2)
```




```
table(all$V65)
```

```
##
##    0    1
## 5353  404
```

```
table(clust2)
```

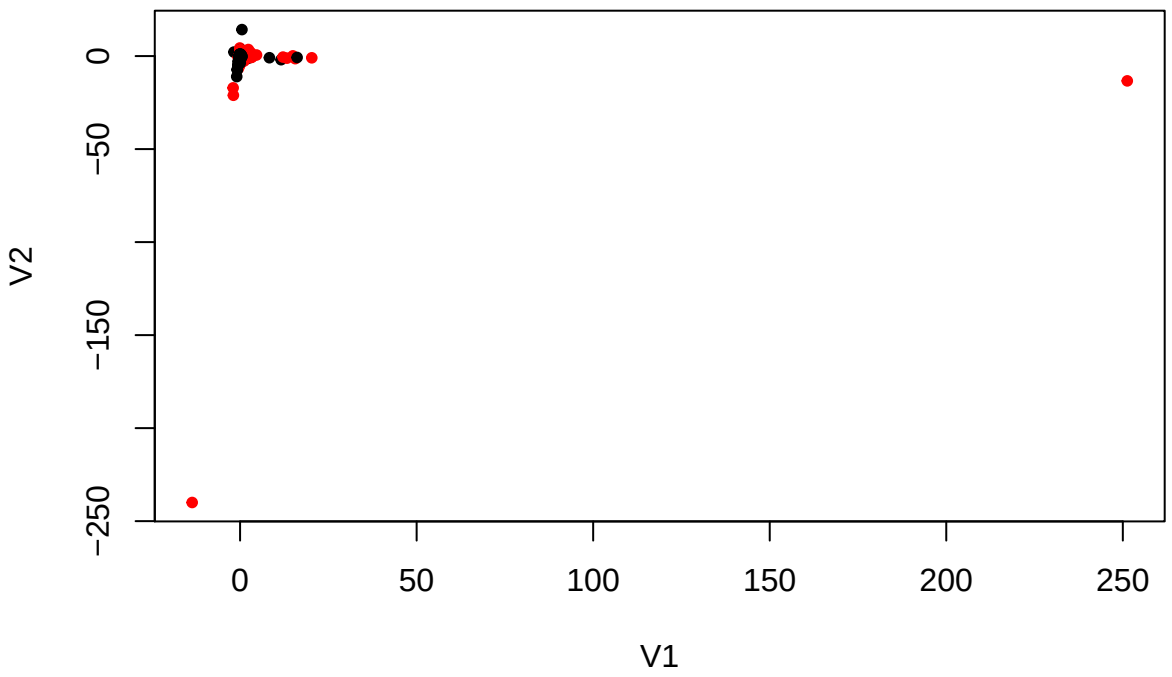
```
## clust2
##    1    2
## 5756    1
```

In all the 5757 samples, 404 of them are in one group and the other 5352 are in the other group. But according to the cluster result, using complete linkage, 5756 are in one group and the left one in the other group. It is obvious that most would-bankrupt companies are not clustered together. So I think it is not a good way to seperate these samples using clustering.

MDS

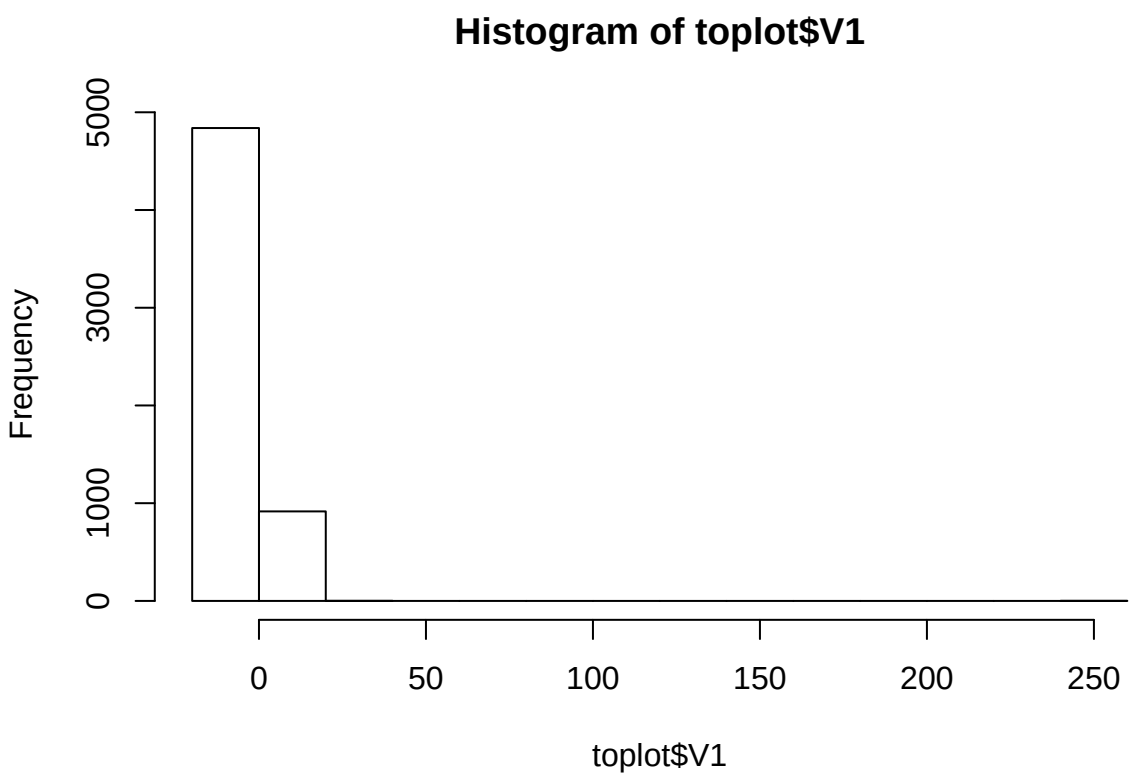
```
mds = cmdscale(all_dist, k=2, add=T, list.=T) ## 2-dim, returning a list because list.=T
```

```
toplot <- as.data.frame(mds$points)
col <- all$V65
col <- as.character(col)
col[which(col=="1")] <- "black"
col[which(col=="0")] <- "red"
plot(toplot, pch=20,col=col)
```



From this plot it is hard to tell it seperate the two groups sucessfully or not. I will remove the out lier and than redraw the plot.

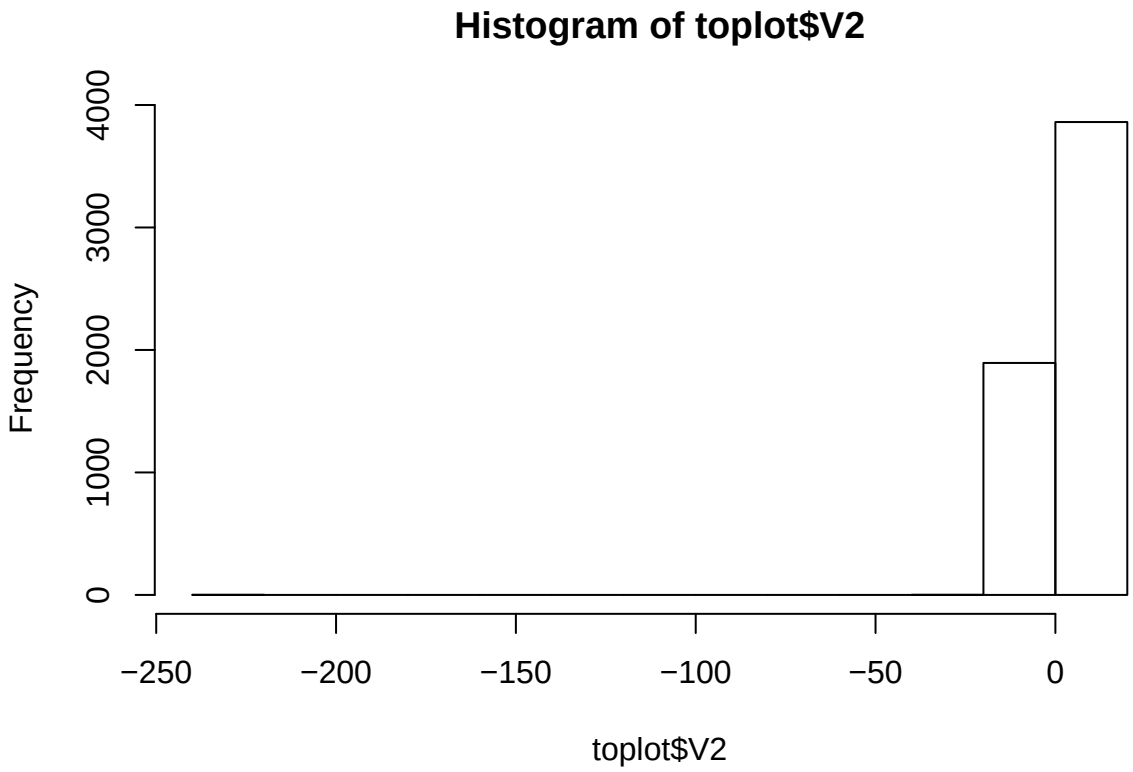
```
hist(toplot$V1)
```




```
summary(toplot$V1)
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -13.56637  -0.13516  -0.10446   0.00000  -0.04929 251.26669
```

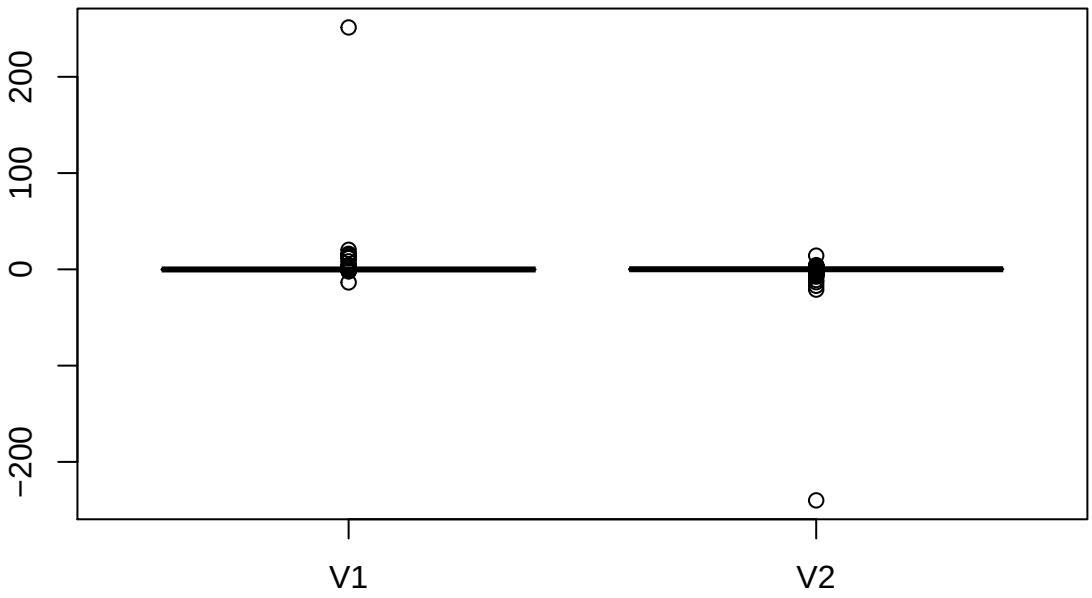
```
hist(toplot$V2)
```




```
summary(toplot$V2)
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -239.97273   -0.07319    0.11578    0.00000    0.26697   14.22921
```

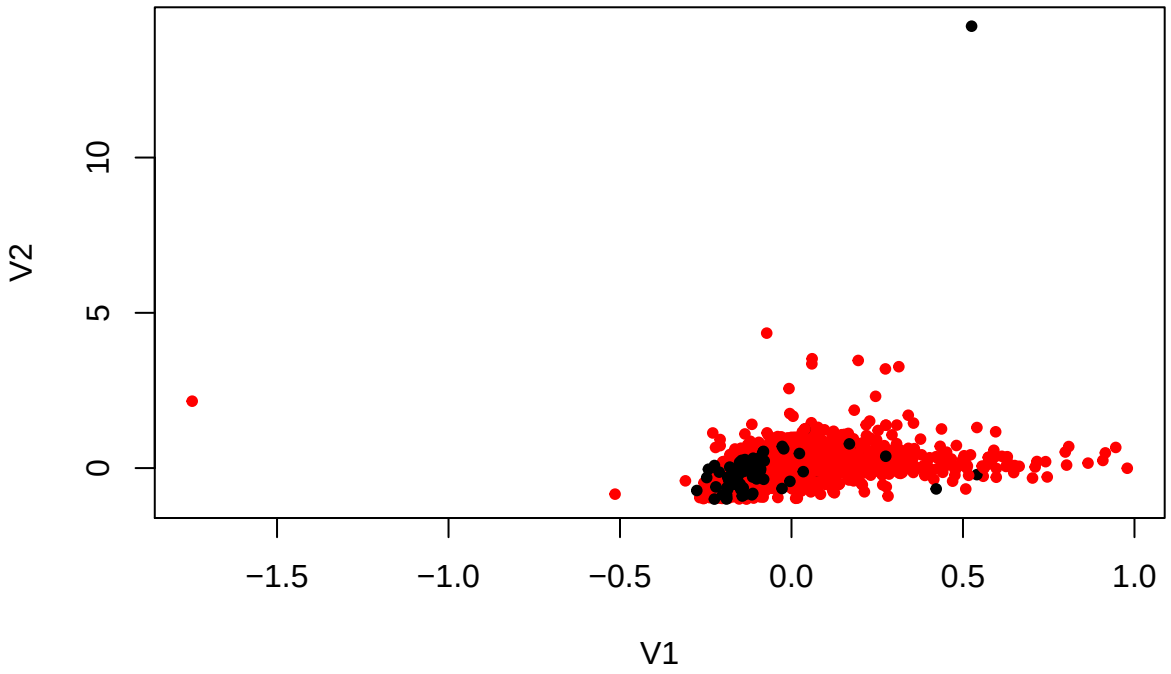
```
boxplot(toplot)
```



Most V1 are clusterd from -15 to 1, V2 are clustered from -1 to 15. Thus, I will ignore

```
toplot1 <- cbind.data.frame(toplot,col)
toplot1 <- toplot1[which(toplot1$V1<1),]
toplot1 <- toplot1[which(toplot1$V2>-1),]
```

```
plot(toplot1[,1:2], pch=20,col=col)
```



After removing the outliers, the MDS plot could seperate the two group, bankrupt or not, in general.

In general, MDS method is much better than cluster, but both of the unsupervised methods are not so good as supervised method. However, MDS gives a great visulization result.