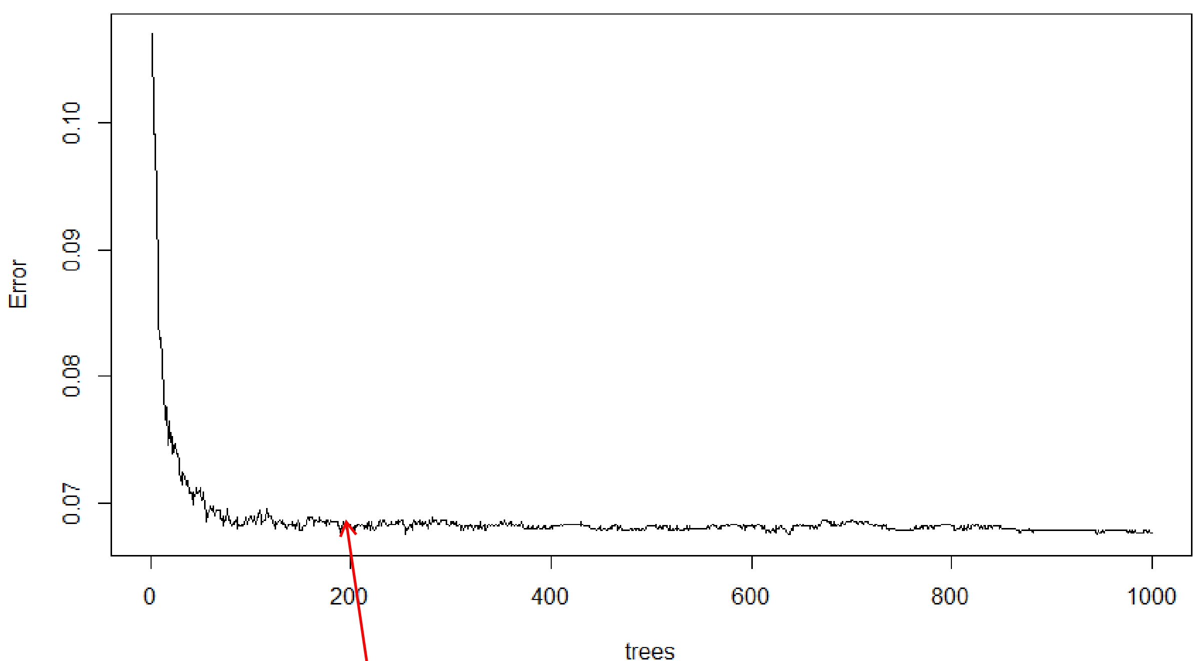
Final Assignment

Jing Tang

1 Supervised learning

1.1 Random forest

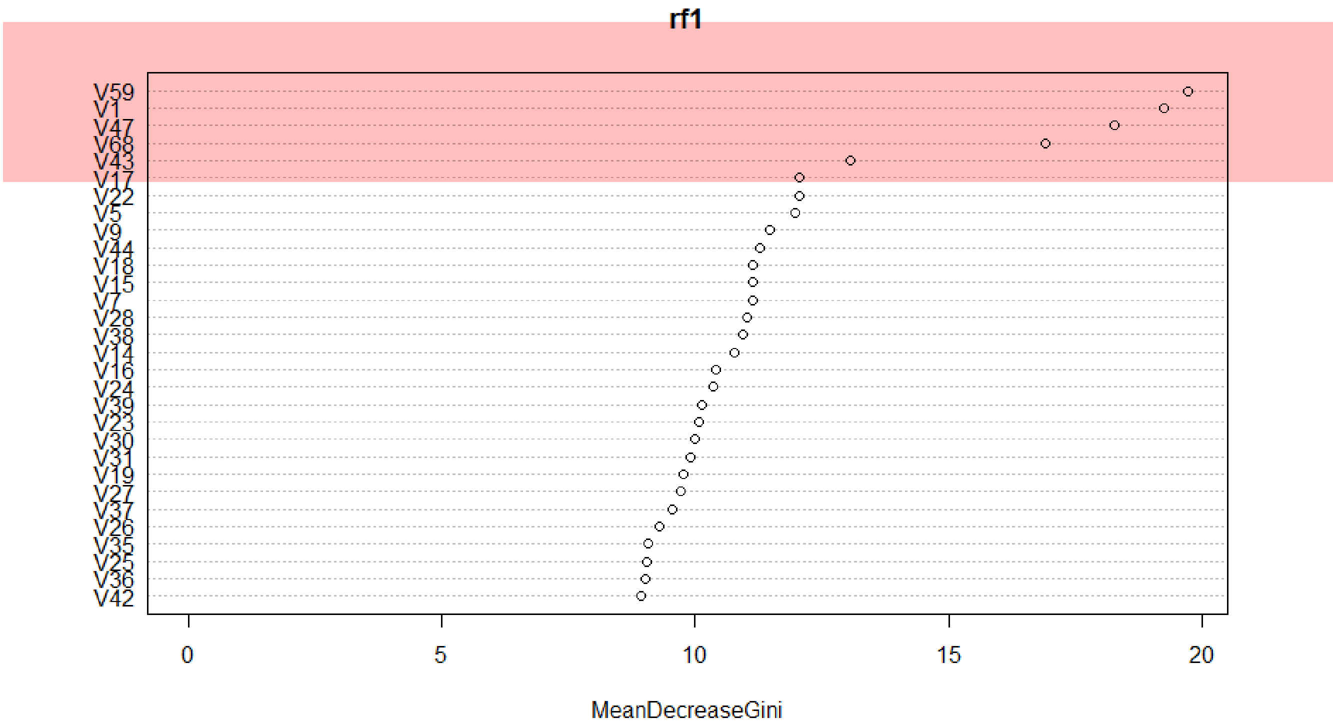
```
Call:
 randomForest(formula = ytrain ~ ., data = Xtrain, ntree = 1000)
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 9

        OOB estimate of  error rate: 6.77%
Confusion matrix:
     0  1 class.error
0 5416 58  0.01059554
1  336 12  0.96551724
```



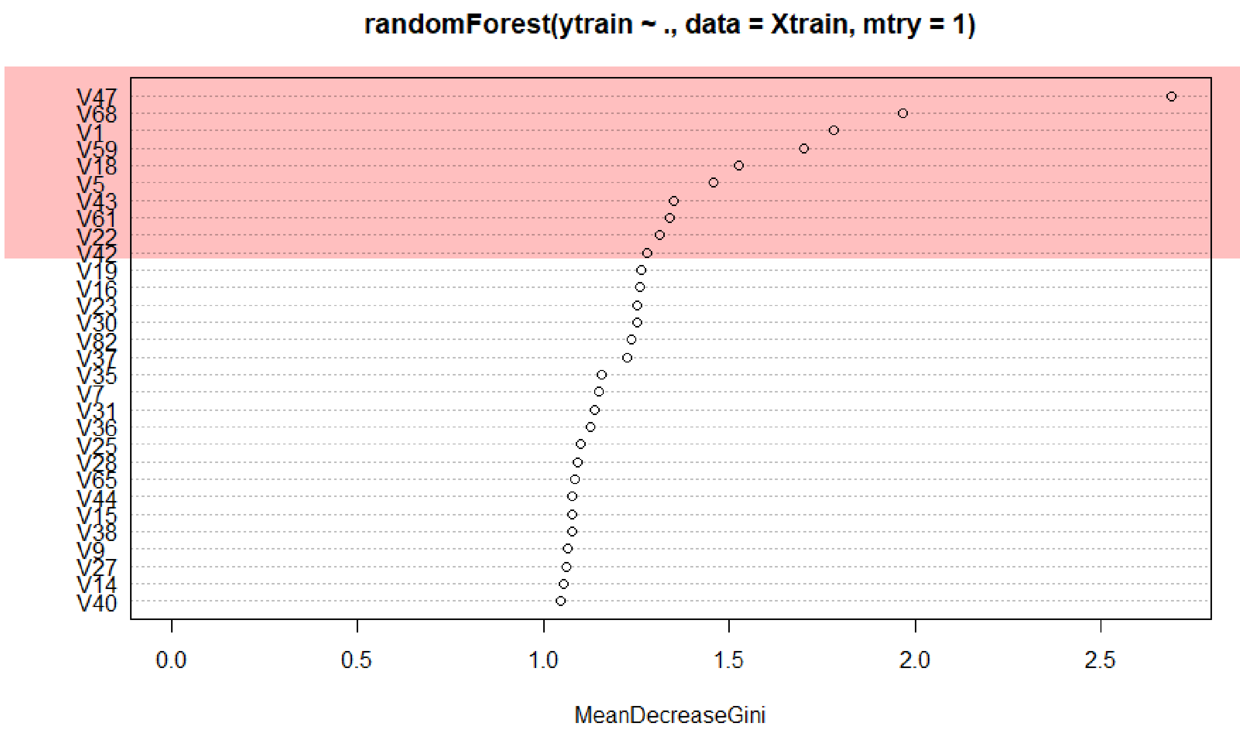

```
> rf1$confusion ## same as table(Heart2.train$AHD, rf$predicted)
     0  1 class.error
0 5416 58  0.01059554
1  336 12  0.96551724
```

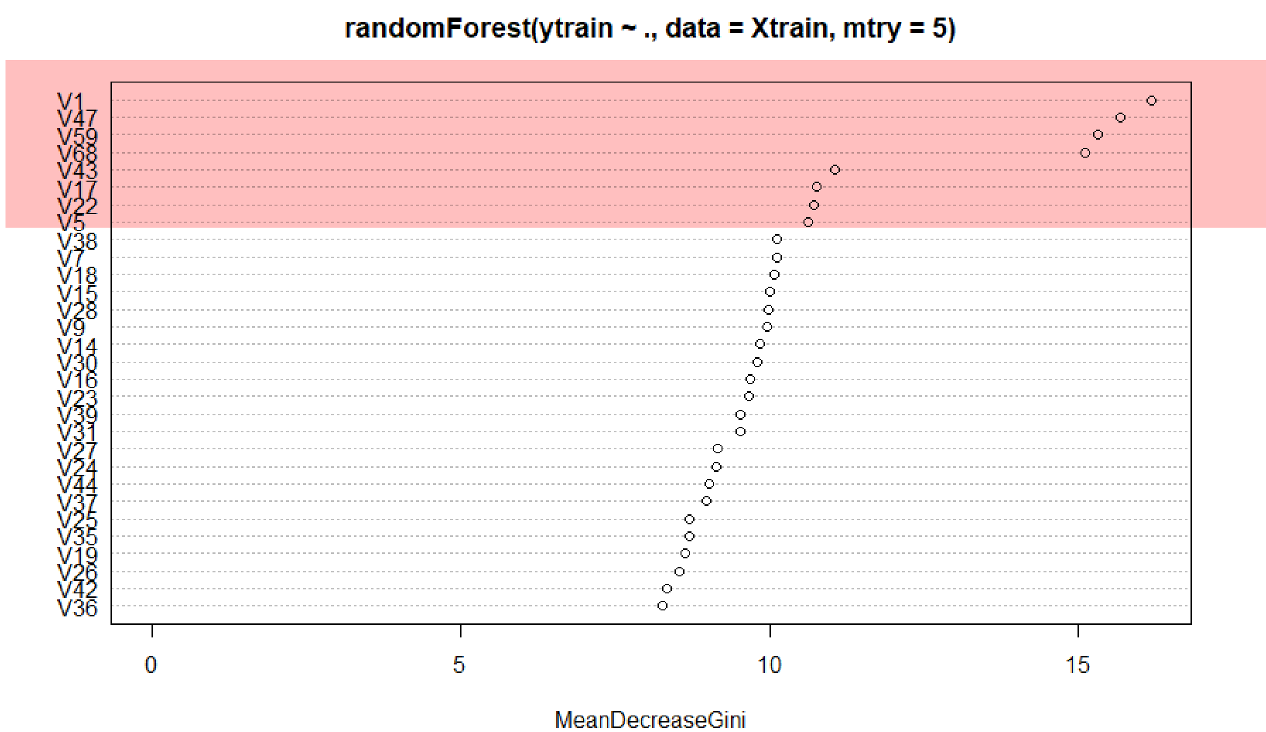
The results should be stable when n.trees>200

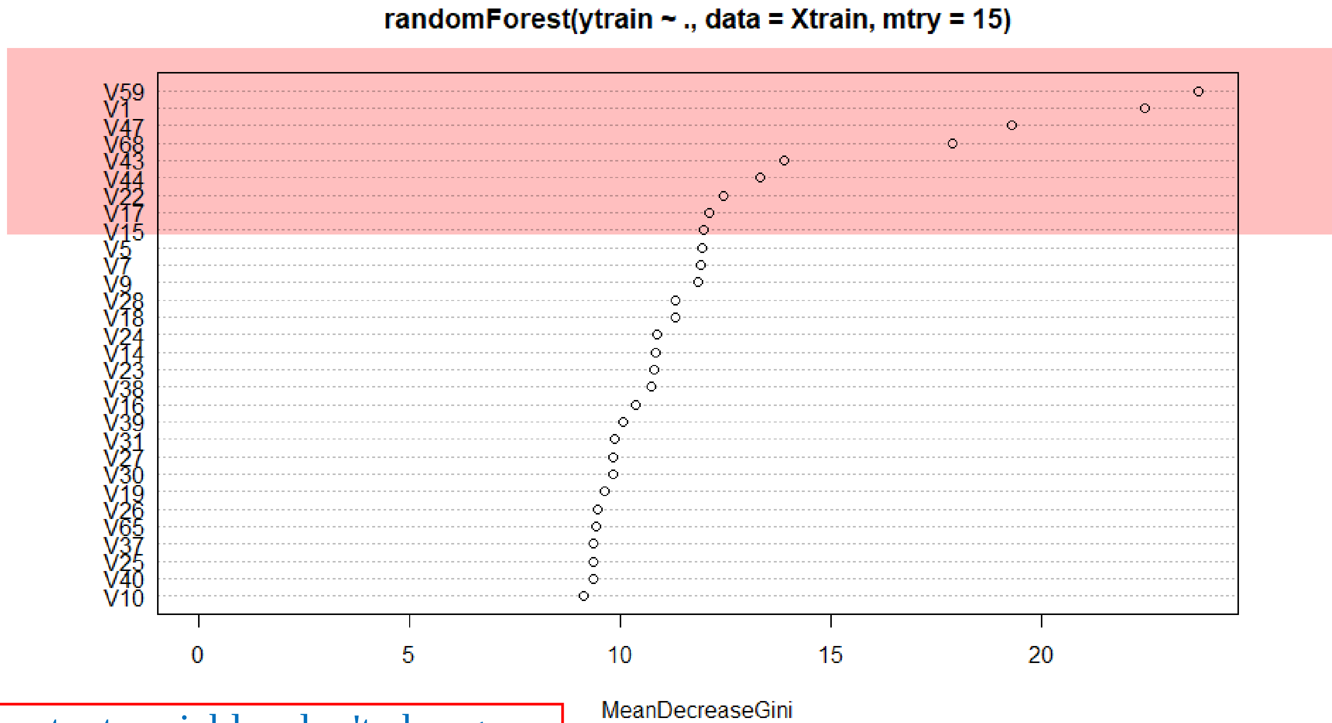
important variables



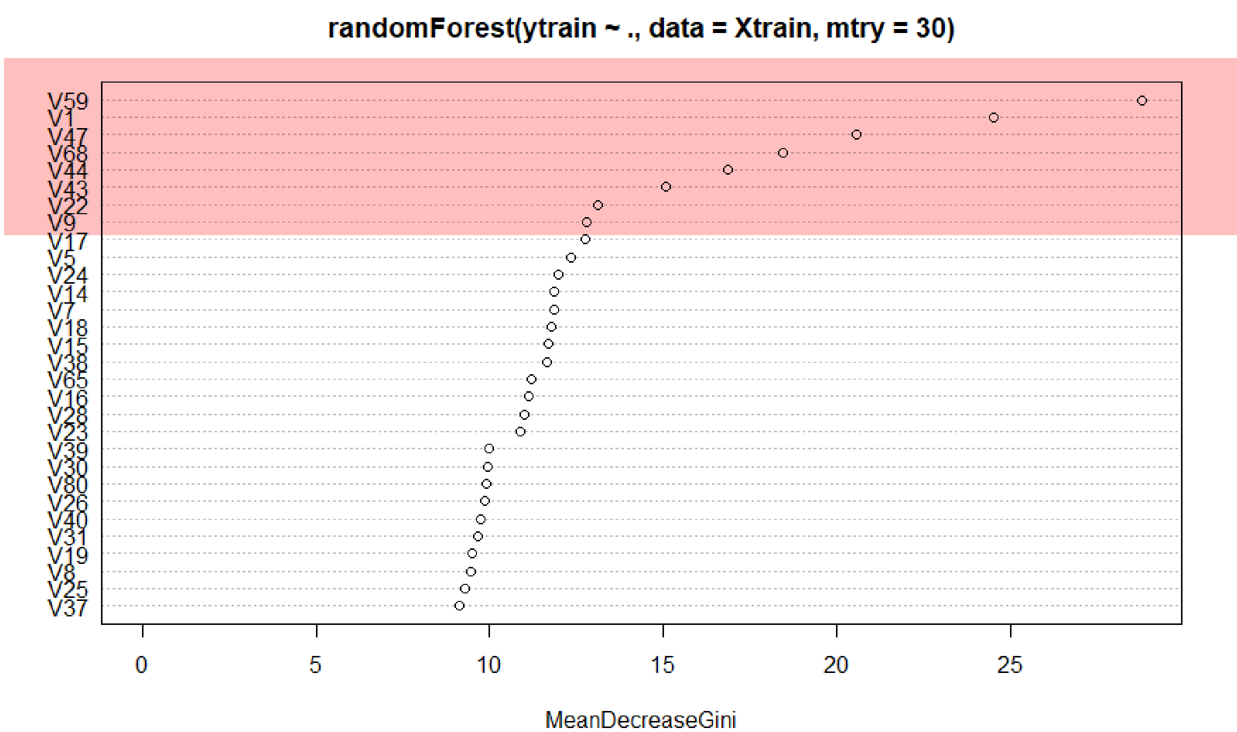
Default: $m = \sqrt{p}$

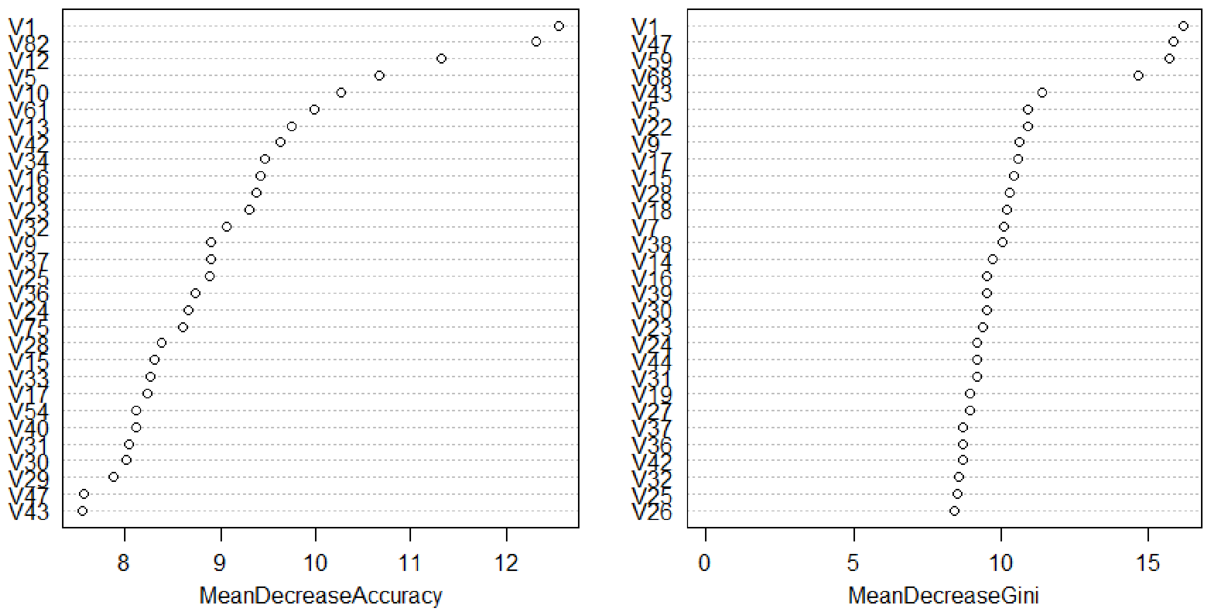

randomForest(ytrain ~ ., data = Xtrain, mtry = 1)
V47
V68
V1
V59
V18
V5
V43
V61
V22
V42
V19
V16
V23
V30
V35
V7
V31
V36
V25
V28
V65
V44
V15
V38
V9
V27
V14
V40
0.0
0.5
1.0
1.5
2.0
2.5
MeanDecreaseGini


randomForest(ytrain ~ ., data = Xtrain, mtry = 5)
V1
V47
V59
V68
V43
V17
V22
V5
V38
V7
V18
V15
V28
V9
V14
V30
V16
V23
V39
V31
V27
V24
V44
V37
V25
V35
V19
V26
V42
V36
0
5
10
15
MeanDecreaseGini

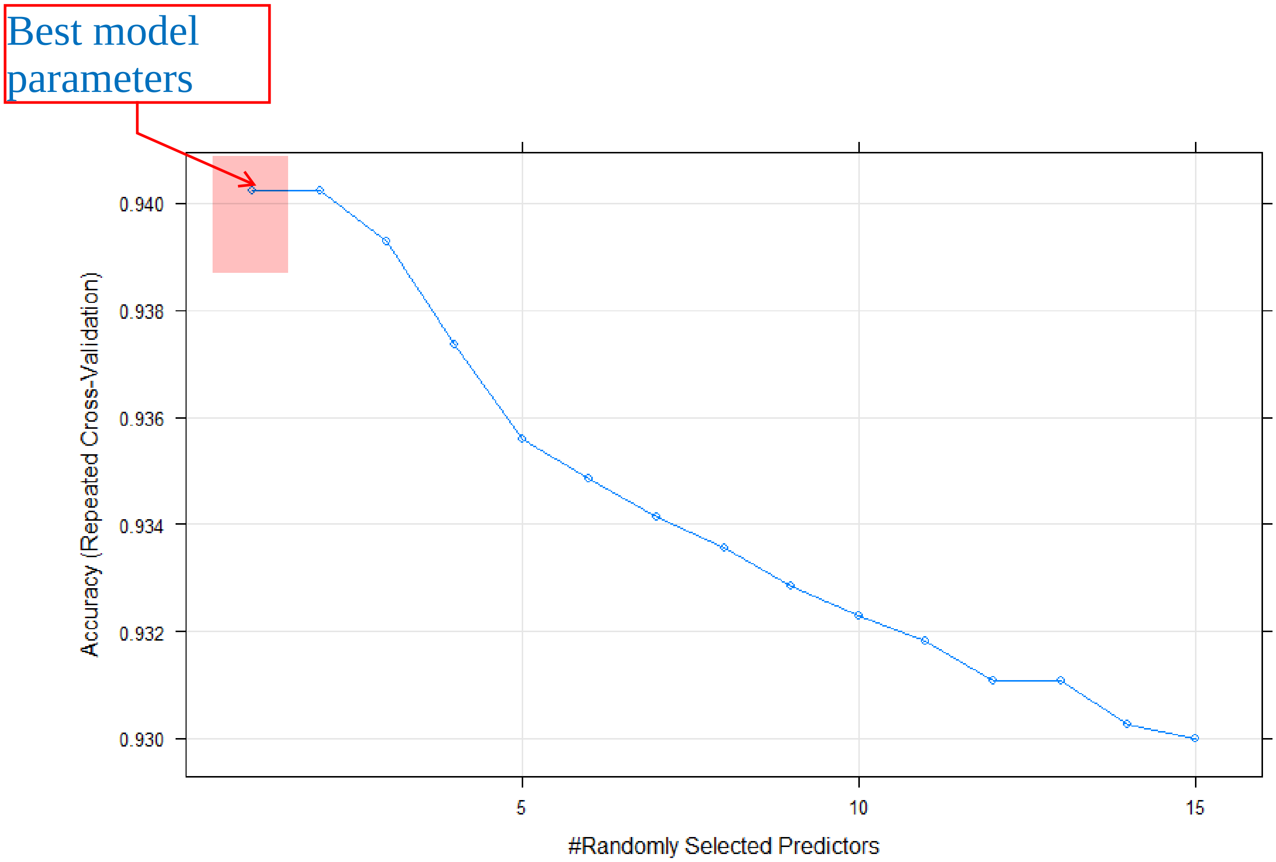


important variables don't change too much when mtry>15






```
> rf3

Call:
 randomForest(formula = ytrain ~ ., data = Xtrain, mtry = 5, importance = T,
     ntree = 500)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 5

        OOB estimate of  error rate: 6.42%
Confusion matrix:
     0  1 class.error
0 5441 33 0.006028498
1  341  7 0.979885057
```




```
> fitRFcaret
Random Forest

5822 samples
  85 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 4 times)
Summary of sample sizes: 4657, 4657, 4657, 4658, 4659, 4658, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   1    0.9402269  0.0000000000
   2    0.9402269  0.0000000000
   3    0.9392822  0.0005900409
   4    0.9373500  0.0063828841
   5    0.9355893  0.0075910832
   6    0.9348595  0.0197728115
   7    0.9341296  0.0267764935
   8    0.9335713  0.0338881696
   9    0.9328411  0.0283151607
  10    0.9322829  0.0365391783
  11    0.9318106  0.0363864358
  12    0.9310804  0.0329258373
  13    0.9310803  0.0350096328
  14    0.9302648  0.0428454060
  15    0.9300068  0.0369804461
```

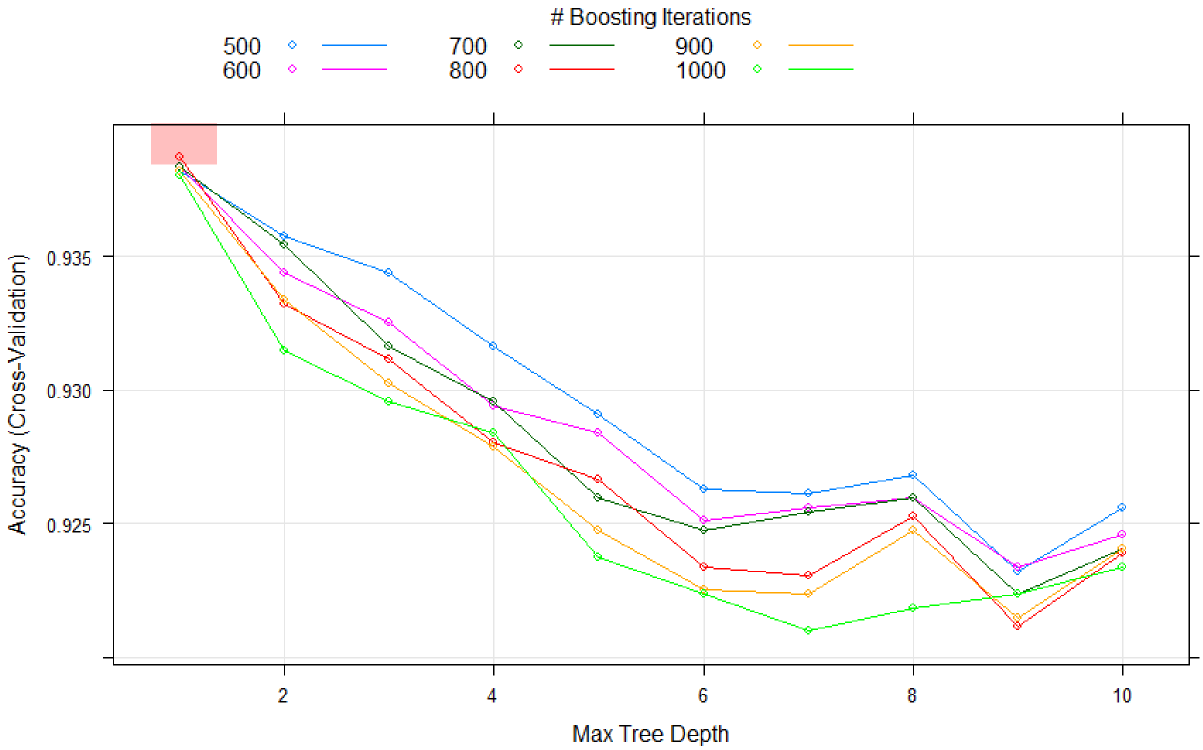
```
Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 1.
> fitRFcaret$results
   mtry  Accuracy        Kappa   AccuracySD     KappaSD
1     1 0.9402269 0.0000000000 0.0004043252 0.000000000
2     2 0.9402269 0.0000000000 0.0004043252 0.000000000
3     3 0.9392822 0.0005900409 0.0009564775 0.007450466
4     4 0.9373500 0.0063828841 0.0013938047 0.016942251
5     5 0.9355893 0.0075910832 0.0018432104 0.017279734
6     6 0.9348595 0.0197728115 0.0023748721 0.025277640
7     7 0.9341296 0.0267764935 0.0023867457 0.029056280
8     8 0.9335713 0.0338881696 0.0022060082 0.035610929
9     9 0.9328411 0.0283151607 0.0026492134 0.032897900
10   10 0.9322829 0.0365391783 0.0026853505 0.033119201
11   11 0.9318106 0.0363864358 0.0028549762 0.034983898
12   12 0.9310804 0.0329258373 0.0030386774 0.033058614
13   13 0.9310803 0.0350096328 0.0033786478 0.035697648
14   14 0.9302648 0.0428454060 0.0039815281 0.031770676
15   15 0.9300068 0.0369804461 0.0035415607 0.034421472
> fitRFcaret$finalModel

Call:
 randomForest(x = x, y = y, ntree = 500, mtry = param$mtry)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 1

        OOB estimate of  error rate: 5.98%
Confusion matrix:
     0 1 class.error
0 5474 0           0
1  348 0           1
> fitRFcaret$finalModel$confusion ## OOB confusion matrix
     0 1 class.error
0 5474 0           0
1  348 0           1
```

Test_set show similar performance as the Training set

```
> rff

Call:
 randomForest(formula = ytest ~ ., data = Xtest, mtry = 1, importance = T,
   ntree = 500)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 1

        OOB estimate of  error rate: 5.95%
Confusion matrix:
     0 1 class.error
0 3762 0           0
1  238 0           1
```

1.2 Boosting

```
> mse(ytrain, predict(bt4b, Xtrain, n.trees=500))
[1] 0.04863351
> bt4c = gbm(ytrain ~ ., data=Xtrain, distribution="gaussian", n.trees=1000,
interaction.depth=4,shrinkage =0.01)
> mse(ytrain, predict(bt4c, Xtrain, n.trees=1000))
[1] 0.04657675
> bt4d = gbm(ytrain ~ ., data=Xtrain, distribution="gaussian", n.trees=1000,
interaction.depth=4,shrinkage =0.1)
> mse(ytrain, predict(bt4d, Xtrain, n.trees=1000))
[1] 0.03518477
> bt4e = gbm(ytrain ~ ., data=Xtrain, distribution="gaussian", n.trees=1000,
interaction.depth=8,shrinkage =0.1)
> mse(ytrain, predict(bt4e, Xtrain, n.trees=1000))
[1] 0.02490013
```




```
> boost.caretk
Stochastic Gradient Boosting

5822 samples
  85 predictor
   2 classes: '0', '1'

Pre-processing: centered (85), scaled (85)
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 4658, 4657, 4658, 4657, 4658
Resampling results across tuning parameters:

  interaction.depth  n.trees  Accuracy   Kappa
```

1	500	0.9381653	0.01054431
1	600	0.9383371	0.01538167
1	700	0.9383372	0.02419928
1	800	0.9386809	0.02539704
1	900	0.9381657	0.02411130
1	1000	0.9379936	0.01928926
2	500	0.9357604	0.04518158
2	600	0.9343864	0.04219736
2	700	0.9354173	0.04917754
2	800	0.9331836	0.04808394
2	900	0.9333559	0.04400326
2	1000	0.9314669	0.04408119
3	500	0.9343867	0.05826943
3	600	0.9324977	0.05796594
3	700	0.9316389	0.06709426
3	800	0.9311235	0.06996922
3	900	0.9302641	0.06063468
3	1000	0.9295779	0.07488432
4	500	0.9316384	0.07954427
4	600	0.9294058	0.06030805
4	700	0.9295775	0.06390862
4	800	0.9280318	0.06002950
4	900	0.9278597	0.05965105
4	1000	0.9283756	0.07589225
5	500	0.9290624	0.06299767
5	600	0.9283750	0.06144044
5	700	0.9259704	0.05580681
5	800	0.9266574	0.06094539
5	900	0.9247676	0.05320606
5	1000	0.9237373	0.06482440
6	500	0.9263130	0.05300641
6	600	0.9251104	0.05812387
6	700	0.9247670	0.06734556
6	800	0.9233932	0.05738089
6	900	0.9225350	0.05197630
6	1000	0.9223629	0.05834840
7	500	0.9261424	0.07346706
7	600	0.9256264	0.06896808
7	700	0.9254549	0.07567146
7	800	0.9230496	0.07039719
7	900	0.9223629	0.07151350
7	1000	0.9209885	0.06590763
8	500	0.9268282	0.07948756
8	600	0.9259689	0.08073917
8	700	0.9259692	0.07650827
8	800	0.9252824	0.07520018
8	900	0.9247672	0.07782488
8	1000	0.9218468	0.05050853
9	500	0.9232214	0.05734985
9	600	0.9233931	0.05689134
9	700	0.9223626	0.05526615
9	800	0.9211607	0.05281097
9	900	0.9215039	0.05686916
9	1000	0.9223627	0.07227016
10	500	0.9256270	0.06500132
10	600	0.9245958	0.06624805
10	700	0.9240802	0.06200659

Best Model

```
  10                    800      0.9239093  0.06144751
  10                    900      0.9240814  0.07224810
  10                   1000      0.9233942  0.07358039

Tuning parameter 'shrinkage' was held constant at a value of 0.1
Tuning parameter 'n.minobsinnode' was held constant
 at a value of 5
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were n.trees = 800, interaction.depth =
1, shrinkage = 0.1 and n.minobsinnode = 5.

> boost.caretk$results
```

	shrinkage	interaction.depth	n.minobsinnode	n.trees	Accuracy	Kappa	AccuracySD	KappaSD
1	0.1	1	5	500	0.9381653	0.01054431	0.0010734365	0.01369015
7	0.1	2	5	500	0.9357604	0.04518158	0.0016775932	0.03763587
13	0.1	3	5	500	0.9343867	0.05826943	0.0018868959	0.02886149
19	0.1	4	5	500	0.9316384	0.07954427	0.0039217657	0.04060863
25	0.1	5	5	500	0.9290624	0.06299767	0.0045630673	0.04006288
31	0.1	6	5	500	0.9263130	0.05300641	0.0047762955	0.03214117
37	0.1	7	5	500	0.9261424	0.07346706	0.0046222131	0.01781837
43	0.1	8	5	500	0.9268282	0.07948756	0.0049302802	0.03001483
49	0.1	9	5	500	0.9232214	0.05734985	0.0054660193	0.02251782
55	0.1	10	5	500	0.9256270	0.06500132	0.0036209154	0.01488375
2	0.1	1	5	600	0.9383371	0.01538167	0.0015485408	0.03259579
8	0.1	2	5	600	0.9343864	0.04219736	0.0027014687	0.04616569
14	0.1	3	5	600	0.9324977	0.05796594	0.0035653031	0.02466463
20	0.1	4	5	600	0.9294058	0.06030805	0.0054783038	0.03873499
26	0.1	5	5	600	0.9283750	0.06144044	0.0047289216	0.03989789
32	0.1	6	5	600	0.9251104	0.05812387	0.0055335412	0.02552678
38	0.1	7	5	600	0.9256264	0.06896808	0.0053228182	0.02762979
44	0.1	8	5	600	0.9259689	0.08073917	0.0050501489	0.02758914
50	0.1	9	5	600	0.9233931	0.05689134	0.0050038165	0.02792410
56	0.1	10	5	600	0.9245958	0.06624805	0.0042794424	0.01666580
3	0.1	1	5	700	0.9383372	0.02419928	0.0007286356	0.03759580
9	0.1	2	5	700	0.9354173	0.04917754	0.0018651242	0.02711429
15	0.1	3	5	700	0.9316389	0.06709426	0.0035659372	0.02920548
21	0.1	4	5	700	0.9295775	0.06390862	0.0039823789	0.02970442
27	0.1	5	5	700	0.9259704	0.05580681	0.0041819122	0.03303151
33	0.1	6	5	700	0.9247670	0.06734556	0.0045491845	0.01722973
39	0.1	7	5	700	0.9254549	0.07567146	0.0044416557	0.02493348
45	0.1	8	5	700	0.9259692	0.07650827	0.0046255836	0.02619863
51	0.1	9	5	700	0.9223626	0.05526615	0.0050456365	0.03637053
57	0.1	10	5	700	0.9240802	0.06200659	0.0038417601	0.02113308
4	0.1	1	5	800	0.9386809	0.02539704	0.0017803992	0.04098618
10	0.1	2	5	800	0.9331836	0.04808394	0.0027001199	0.03425380
16	0.1	3	5	800	0.9311235	0.06996922	0.0041763624	0.02252601
22	0.1	4	5	800	0.9280318	0.06002950	0.0032871683	0.03114995
28	0.1	5	5	800	0.9266574	0.06094539	0.0041030168	0.03823665
34	0.1	6	5	800	0.9233932	0.05738089	0.0038794739	0.01339706
40	0.1	7	5	800	0.9230496	0.07039719	0.0058572900	0.02950995
46	0.1	8	5	800	0.9252824	0.07520018	0.0040521276	0.02621601
52	0.1	9	5	800	0.9211607	0.05281097	0.0039627859	0.02913087
58	0.1	10	5	800	0.9239093	0.06144751	0.0038691638	0.02693910
5	0.1	1	5	900	0.9381657	0.02411130	0.0014805154	0.03993995
11	0.1	2	5	900	0.9333559	0.04400326	0.0022520384	0.03238760
17	0.1	3	5	900	0.9302641	0.06063468	0.0026176425	0.01903974
23	0.1	4	5	900	0.9278597	0.05965105	0.0036978392	0.03059895

```
29        0.1               5               5      900 0.9247676 0.05320606 0.0041117038 0.02863171
35        0.1               6               5      900 0.9225350 0.05197630 0.0041849682 0.02220264
41        0.1               7               5      900 0.9223629 0.07151350 0.0042454218 0.02248000
47        0.1               8               5      900 0.9247672 0.07782488 0.0039849874 0.02715224
53        0.1               9               5      900 0.9215039 0.05686916 0.0052246280 0.02252700
59        0.1              10               5      900 0.9240814 0.07224810 0.0048735663 0.02751137
6         0.1               1               5     1000 0.9379936 0.01928926 0.0011345011 0.02922307
12        0.1               2               5     1000 0.9314669 0.04408119 0.0018585234 0.01387209
18        0.1               3               5     1000 0.9295779 0.07488432 0.0059137189 0.03901509
24        0.1               4               5     1000 0.9283756 0.07589225 0.0041803401 0.03573436
30        0.1               5               5     1000 0.9237373 0.06482440 0.0041428553 0.03739020
36        0.1               6               5     1000 0.9223629 0.05834840 0.0035872770 0.02004546
42        0.1               7               5     1000 0.9209885 0.06590763 0.0053788609 0.01234493
48        0.1               8               5     1000 0.9218468 0.05050853 0.0046131576 0.02634323
54        0.1               9               5     1000 0.9223627 0.07227016 0.0053272995 0.03713970
60        0.1              10               5     1000 0.9233942 0.07358039 0.0034491322 0.03346573
```

```
> ytest <- as.numeric(ytest)
> mse(ytest, test.pred)
[1] 0.05047135
```

MSE for Test_set

1.3 Support vector machine

```
> tune.out <- tune(svm, ytrain ~ ., data=cbind(Xtrain,ytrain), kernel = "radi
al" ,ranges=list(cost=2^(-5:5), gamma=2^(-5:0)))
> tune.out$best.model

Call:
best.tune(method = svm, train.x = ytrain ~ ., data = cbind(Xtrain, ytrain), r
anges = list(cost = 2^(-5:5),
    gamma = 2^(-5:0)), kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.03125
      gamma:  0.03125

Number of Support Vectors:  2010

> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
    cost   gamma
 0.03125 0.03125


- best performance: 0.05977318
```

- Detailed performance results:

	cost	gamma	error	dispersion
1	0.03125	0.03125	0.05977318	0.008864128
2	0.06250	0.03125	0.05977318	0.008864128
3	0.12500	0.03125	0.05977318	0.008864128
4	0.25000	0.03125	0.05977318	0.008864128
5	0.50000	0.03125	0.05977318	0.008864128
6	1.00000	0.03125	0.05977318	0.008864128
7	2.00000	0.03125	0.06200480	0.009024414
8	4.00000	0.03125	0.07042051	0.009357542
9	8.00000	0.03125	0.07660401	0.008747793
10	16.00000	0.03125	0.08244358	0.009695321
11	32.00000	0.03125	0.08519095	0.009133592
12	0.03125	0.06250	0.05977318	0.008864128
13	0.06250	0.06250	0.05977318	0.008864128
14	0.12500	0.06250	0.05977318	0.008864128
15	0.25000	0.06250	0.05977318	0.008864128
16	0.50000	0.06250	0.05977318	0.008864128
17	1.00000	0.06250	0.06080293	0.008761079
18	2.00000	0.06250	0.06767225	0.008597157
19	4.00000	0.06250	0.07540155	0.009265892
20	8.00000	0.06250	0.07918074	0.008646942
21	16.00000	0.06250	0.07918044	0.008643504
22	32.00000	0.06250	0.08192870	0.008199369
23	0.03125	0.12500	0.05977318	0.008864128
24	0.06250	0.12500	0.05977318	0.008864128
25	0.12500	0.12500	0.05977318	0.008864128
26	0.25000	0.12500	0.05977318	0.008864128
27	0.50000	0.12500	0.05977318	0.008864128
28	1.00000	0.12500	0.06234874	0.008213617
29	2.00000	0.12500	0.06973440	0.009373304
30	4.00000	0.12500	0.07231113	0.008955164
31	8.00000	0.12500	0.07299812	0.008724610
32	16.00000	0.12500	0.07420028	0.009146360
33	32.00000	0.12500	0.07523121	0.008595104
34	0.03125	0.25000	0.05977318	0.008864128
35	0.06250	0.25000	0.05977318	0.008864128
36	0.12500	0.25000	0.05977318	0.008864128
37	0.25000	0.25000	0.05977318	0.008864128
38	0.50000	0.25000	0.05994501	0.008888592
39	1.00000	0.25000	0.06337966	0.007568377
40	2.00000	0.25000	0.07093774	0.008799485
41	4.00000	0.25000	0.07128138	0.009279282
42	8.00000	0.25000	0.07196808	0.009596251
43	16.00000	0.25000	0.07265566	0.008876413
44	32.00000	0.25000	0.07351447	0.009122442
45	0.03125	0.50000	0.05977318	0.008864128
46	0.06250	0.50000	0.05977318	0.008864128
47	0.12500	0.50000	0.05977318	0.008864128
48	0.25000	0.50000	0.05977318	0.008864128
49	0.50000	0.50000	0.06011653	0.008833330
50	1.00000	0.50000	0.06544181	0.007085746
51	2.00000	0.50000	0.07007922	0.008754614
52	4.00000	0.50000	0.07059439	0.008815485
53	8.00000	0.50000	0.07128167	0.008921513
54	16.00000	0.50000	0.07214049	0.009201937
55	32.00000	0.50000	0.07231231	0.008929264

```
56  0.03125 1.00000 0.05977318 0.008864128
57  0.06250 1.00000 0.05977318 0.008864128
58  0.12500 1.00000 0.05977318 0.008864128
59  0.25000 1.00000 0.05977318 0.008864128
60  0.50000 1.00000 0.06011653 0.008833330
61  1.00000 1.00000 0.06492605 0.007885387
62  2.00000 1.00000 0.06887706 0.008281053
63  4.00000 1.00000 0.06939282 0.008275513
64  8.00000 1.00000 0.07007981 0.008259033
65 16.00000 1.00000 0.07025163 0.008003409
66 32.00000 1.00000 0.07025163 0.008003409
> svm.radial <- svm(ytrain ~ ., data=cbind(Xtrain,ytrain), kernel = "radial",
  cost = tune.out$best.parameter$cost)
> summary(svm.radial)

Call:
svm(formula = ytrain ~ ., data = cbind(Xtrain, ytrain), kernel = "radial", co
st = tune.out$best.parameter$cost)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.03125
      gamma:  0.01176471

Number of Support Vectors:  1549

 ( 1201 348 )


Number of Classes:  2

Levels:
 0 1



> train.pred <- predict(svm.radial, Xtrain)
> table(ytrain, train.pred)
      train.pred
ytrain    0    1
     0 5474    0
     1  348    0
> test.pred <- predict(svm.radial, Xtest)
> table(ytest, test.pred)
     test.pred
ytest    0    1
    0 3762    0
    1  238    0
```

Parameter tuning of ‘svm’:

```
- sampling method: 10-fold cross validation

- best parameters:
       cost
 0.03162278

- best performance: 0.05960136

- Detailed performance results:
         cost      error  dispersion
1   0.01000000 0.05977318 0.008864128
2   0.01778279 0.05977318 0.008864128
3   0.03162278 0.05960136 0.009128050
4   0.05623413 0.05977318 0.009155393
5   0.10000000 0.06011683 0.008946052
6   0.17782794 0.06063229 0.009094641
7   0.31622777 0.06063229 0.008613000
8   0.56234133 0.06097534 0.008227416
9   1.00000000 0.06217750 0.007628309
10  1.77827941 0.06269267 0.006013307
11  3.16227766 0.06355207 0.006377726
12  5.62341325 0.06595698 0.006128858
13 10.00000000 0.06784672 0.006746541
```

For Training_set, model_Kernal=Polynomial show better performance than model_Kernal=radical, while for test_set, they show the same performance

```
>
> set.seed(1)
> svm.poly <- svm(ytrain ~ ., data=cbind(Xtrain,ytrain), kernel = "polynomial
", degree = 2, cost = tune.outp$best.parameter$cost)
> summary(svm.poly)

Call:
svm(formula = ytrain ~ ., data = cbind(Xtrain, ytrain), kernel = "polynomial
", degree = 2, cost = tune.outp$best.parameter$cost)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  0.03162278
     degree:  2
      gamma:  0.01176471
     coef.0:  0

Number of Support Vectors:  1124

 ( 776 348 )


Number of Classes:  2

Levels:
 0 1



> train.pred <- predict(svm.poly, Xtrain)
> table(ytrain, train.pred)
```

```
       train.pred
ytrain    0    1
     0 5474    0
     1  345    3
> test.pred <- predict(svm.poly, Xtest)
> table(ytest, test.pred)
     test.pred
ytest    0    1
    0 3761    1
    1  238    0
```

1.4 Compare the performance of the 3 best models.

For this particular dataset, Boosting tree show the best predict performance on Test data.

2 Unsupervised learning:

٫.1 K-means

Separating observations to two categories

Data not standardized

it's the usual decomposition of deviance in deviance " Between" and deviance "Within". Ideally, a clustering is good if it has the properties of internal cohesion and external separation, i.e. the BSS/TSS ratio should approach 1.

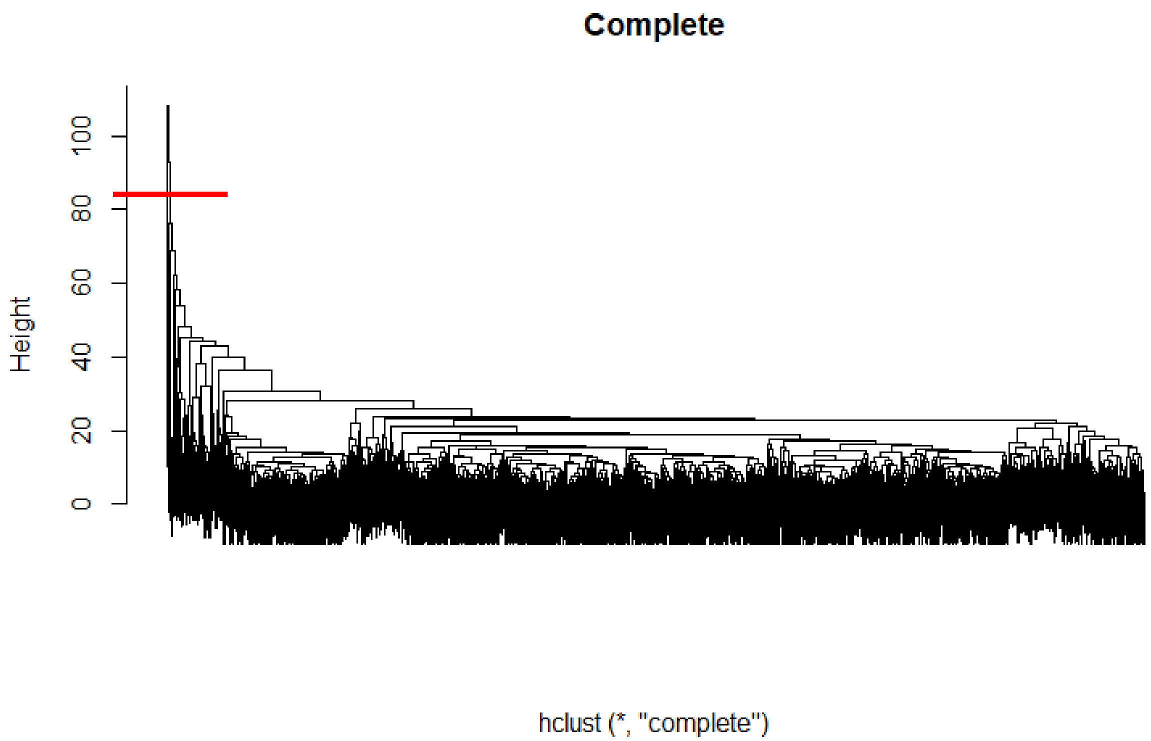
```
> table(aa$cluster, bb$cluster)

     1    2
 1 1579  416
 2  623 3204
> aa
K-means clustering with 2 clusters of sizes 1995, 3827
Within cluster sum of squares by cluster:
[1] 291404.7 580337.7
 (between_SS / total_SS =  51.4 %)
```

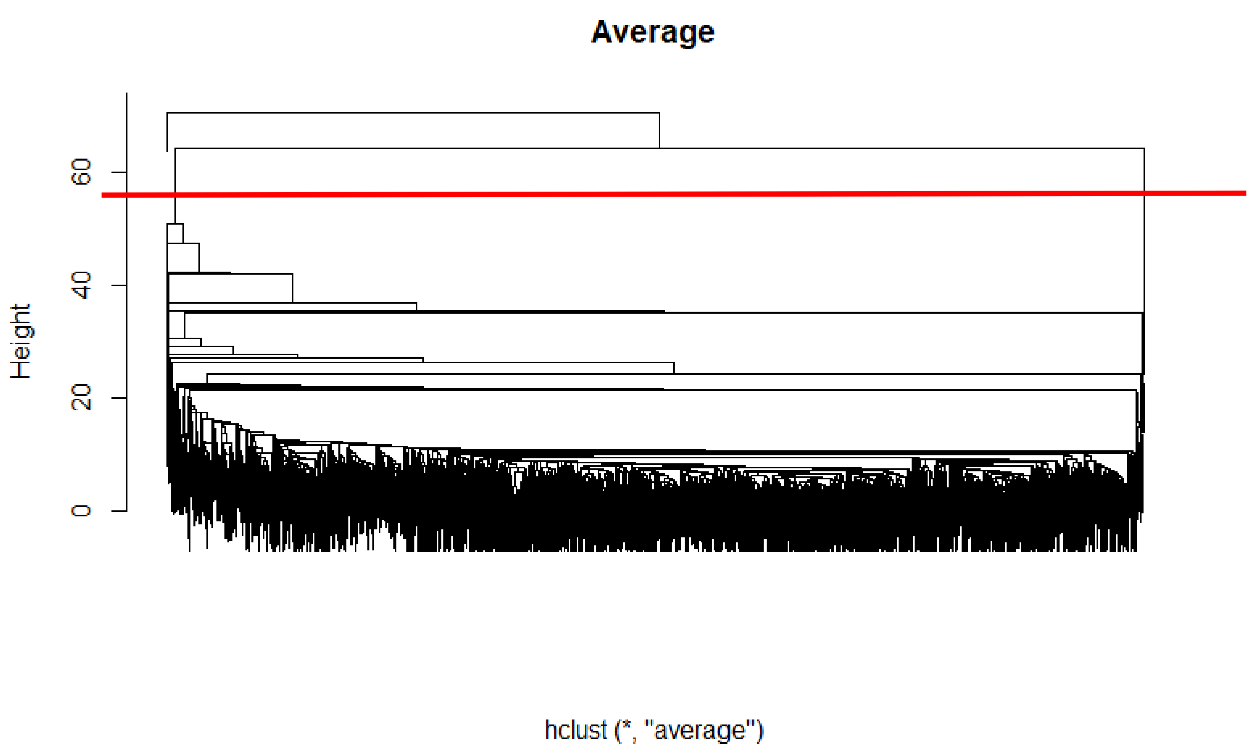
Standardized results

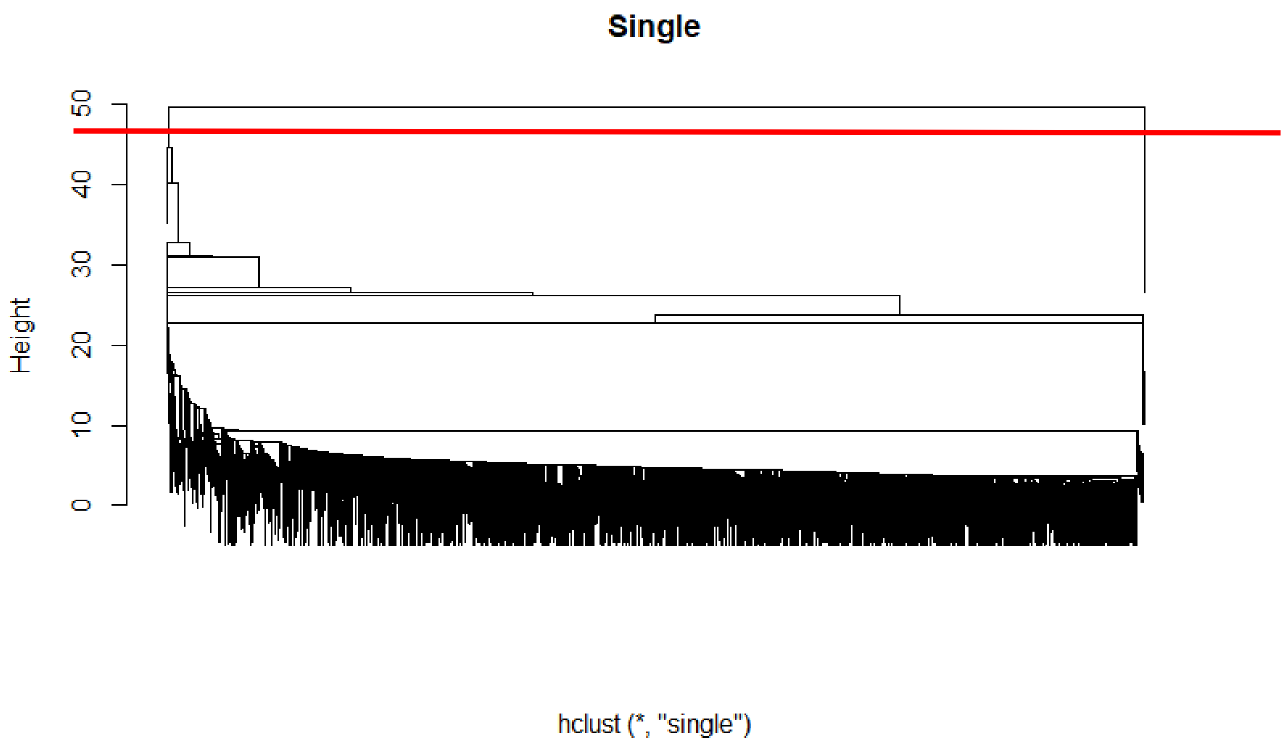
In this case, standardized scales data show relatively bad results

```
K-means clustering with 2 clusters of sizes 2202, 3620

Within cluster sum of squares by cluster:
[1] 204015.1 254321.5
 (between_SS / total_SS =   7.4 %)
```

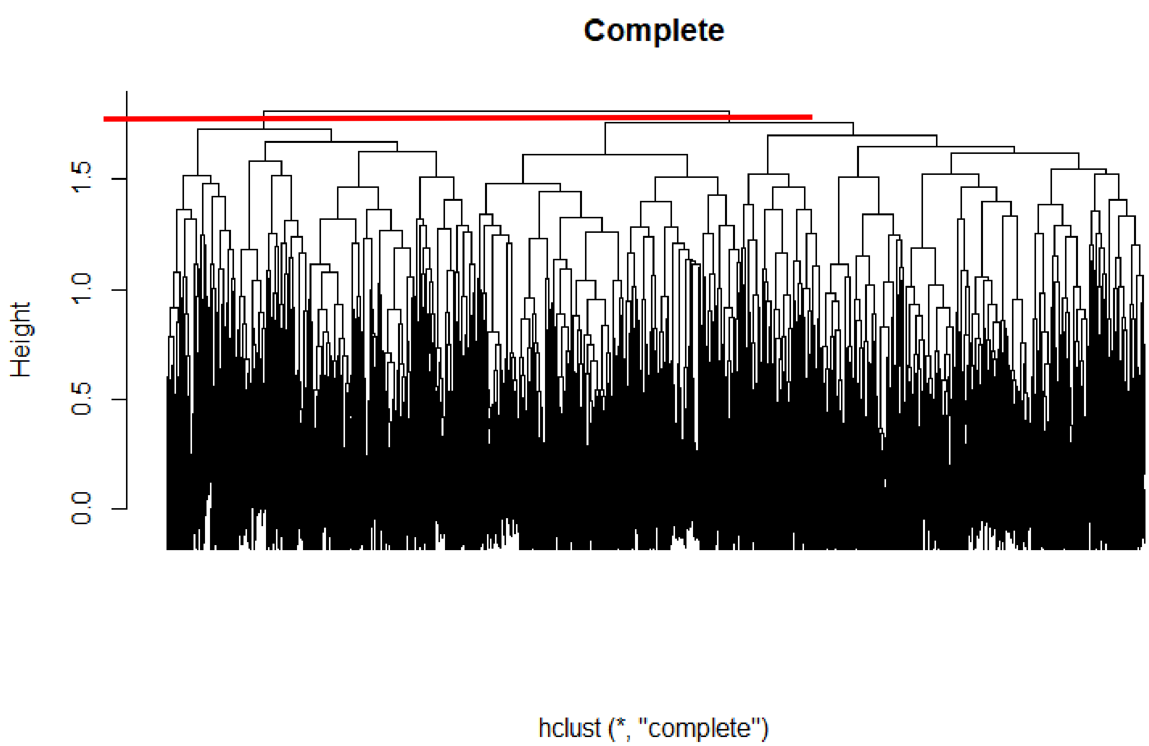
2 Hierarchical clustering

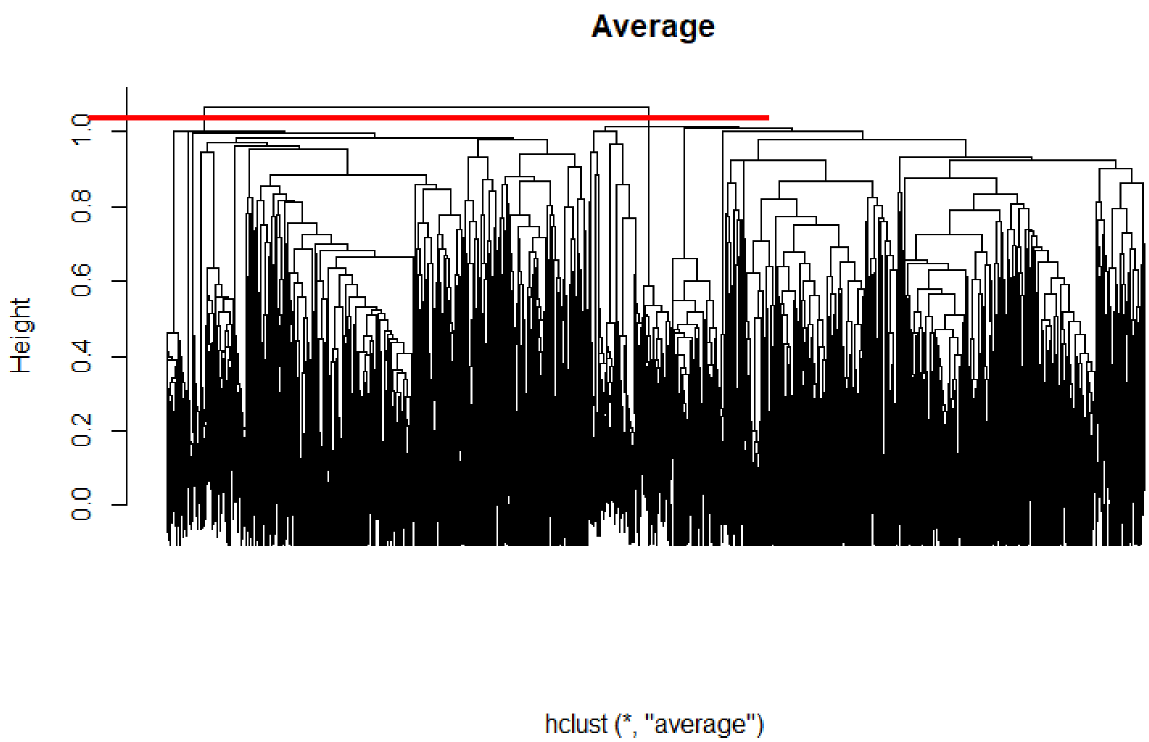


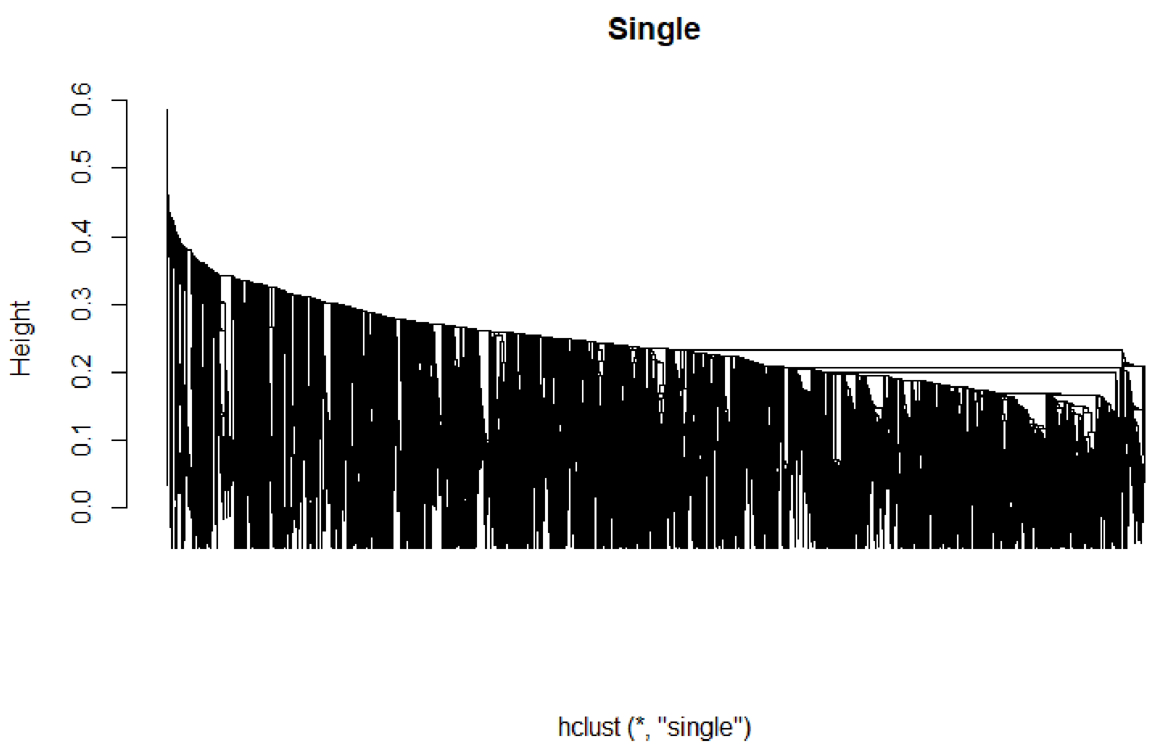




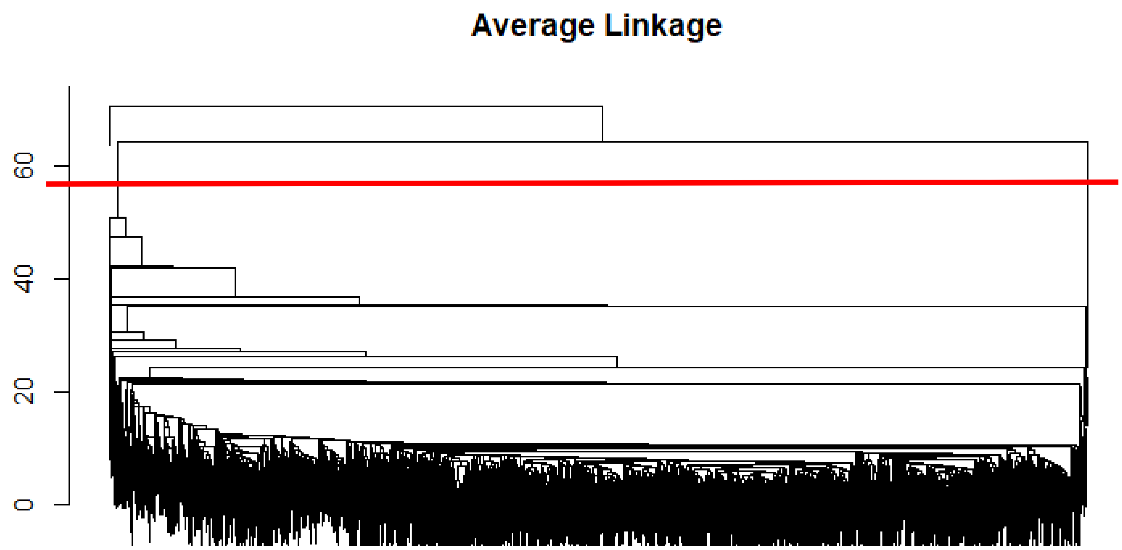
#try the correlation approach.



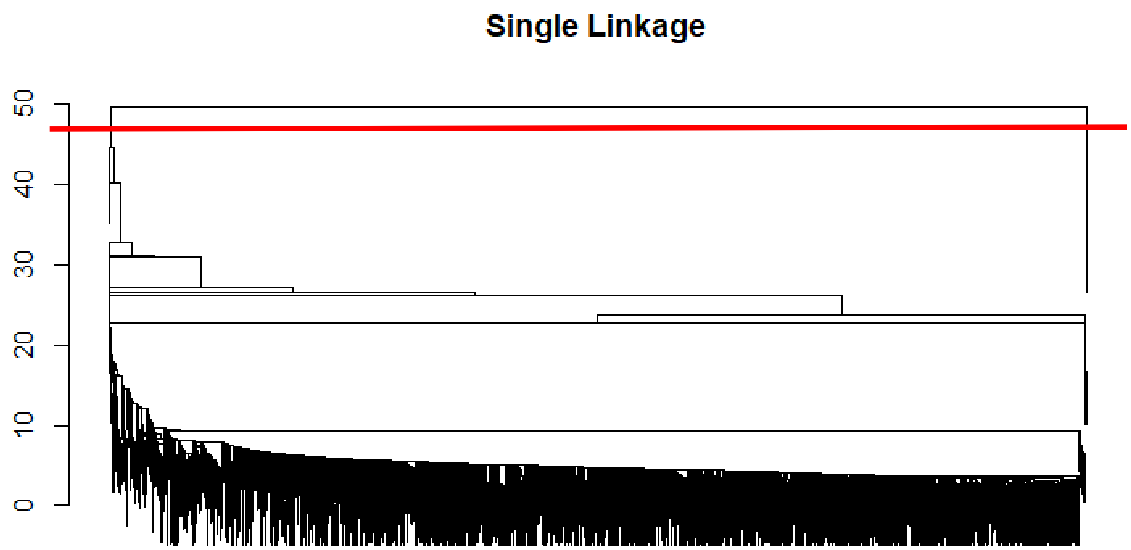

Average
Height
0.0
0.2
0.4
0.6
0.8
1.0
hclust (*, "average")


Single
Height
0.0
0.1
0.2
0.3
0.4
0.5
0.6
hclust (*, "single")

#Change the distance measurement: data.dist = dist(sd.data) ## default is Euclidean distance

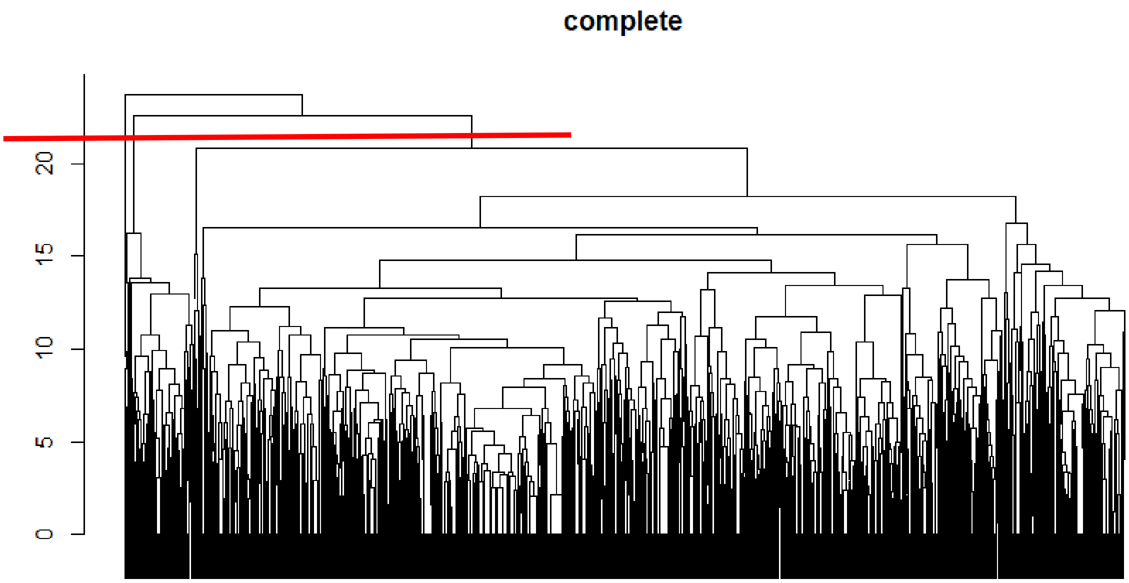


It's reasonable to separate the data as two categories



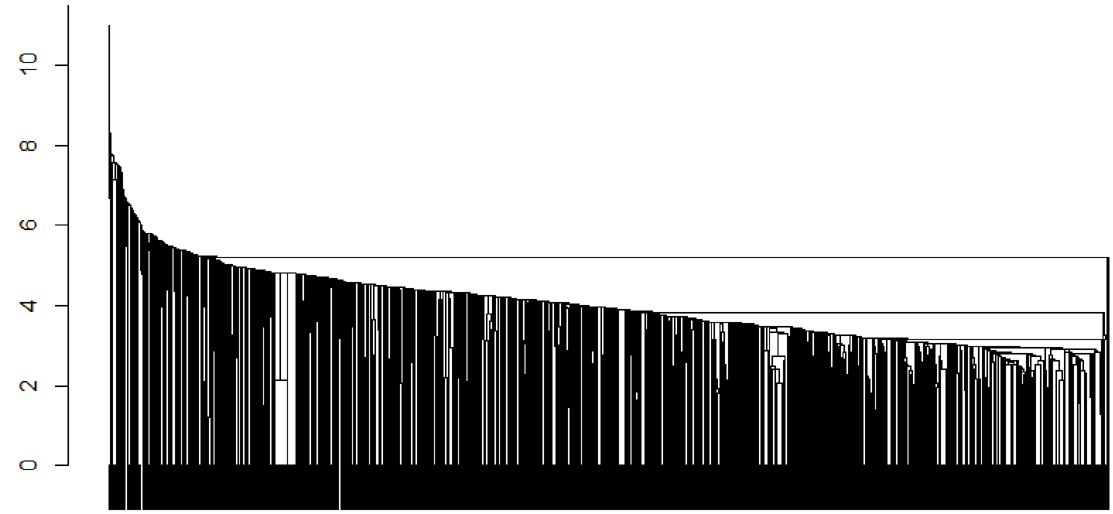
- Use only the variables 6-41. Can you cluster your training data into subsets? Which method performs better for the data?

```
>Xtrain3=C.train[,6:41]
Xtrain4=scale(Xtrain3)
cc = kmeans(Xtrain4, 2, nstart=10)

cc
K-means clustering with 2 clusters of sizes 3342, 2480


Within cluster sum of squares by cluster:
[1] 96414.31 83419.46
 (between_SS / total_SS =  14.2 %)
```



The complete show clearer clusters on the branch

Single Linkage



Average Linkage