Zahra Jalil Pour

Email: h19zahja@du.se

Home Exercise 2

Statistical Learning (AMI22T)

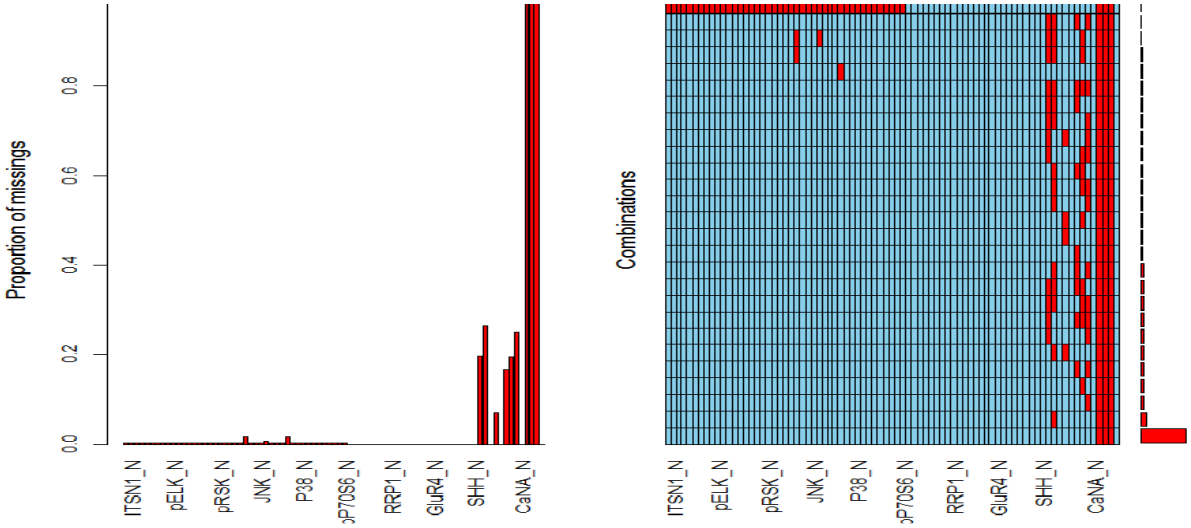
The data ('Mice Protein Expression Data Set') consist of 1080 observations over 82 variables.
These data consisted of numerical protein expression data, and categorical data including mouse identification ,class, genotype, treatment and behavior. The data set consists of the expression levels of 77 proteins. There are 38 control mice and 34 trisomic mice (Down syndrome), for a total of 72 mice. In the experiments, 15 measurements were registered of each protein per mouse.
The aim of this project is to identify a subset of proteins that is discriminant between the classes.
In the present study, mice protein expression data set from UCI repository are classified using Decision Tree, Random Forest ,bagging, Support Vector Machine and clustering which are some of classification methods.
The categorical variables are presented as table below:

Genotype	Treatment	Behavior	class
<chr>	*<chr>*	*<chr>*	*<chr>*
Control	Memantine	C/S	c-CS-m
Control	Memantine	S/C	c-SC-m
Control	Saline	C/S	c-CS-s
Control	Saline	S/C	c-SC-s
Ts65Dn	Memantine	C/S	t-CS-m
Ts65Dn	Memantine	S/C	t-SC-m
Ts65Dn	Saline	C/S	t-CS-s
Ts65Dn	Saline	S/C	t-SC-s

The categorical variable counts are summarized in the below:

Categorical Variable	Count	Total
Genotype	Control = 570, Ts65Dn = 510	1080
Behavior	Memantine = 570, Saline = 510	1080
Treatment	C/S = 525, S/C = 555	1080
class	c-CS-m=150, c-CS-s=135, c-SC-m=150, c-SC-s=135, t-CS-m=135, t-CS-s=105, t-SC-m=135, t-SC-s=135	1080

At first I remove the ID-variable from the data set and transform the three binary variables Genotype, Treatment and Behavior to integer. Then, I look at the missing values. In the plot the color red indicates the NAs.



There are three observations with missing in more than half of the variables. During the program, we see that 528 rows which account for 48.9% of all rows has NAs. So it is better not to exclude all row s with NA, because we are losing too much information. The distribution of NAs among rows is presented the below table:

0	1	2	3	4	5	43
552	165	120	117	104	19	3

Most of the rows contains less than 6 NAs out of 77 variables. For then we can fill the NA value in by replacing with the column mean. In this step we have cleaned the data set. For classification we need to split our dataset. The classification algorithms are tested by splitting the data by 75% as train and 25% as test data to classify on the mice protein data set. Now as the data set is prepared, we can build the model. we can start by Decision tree model.

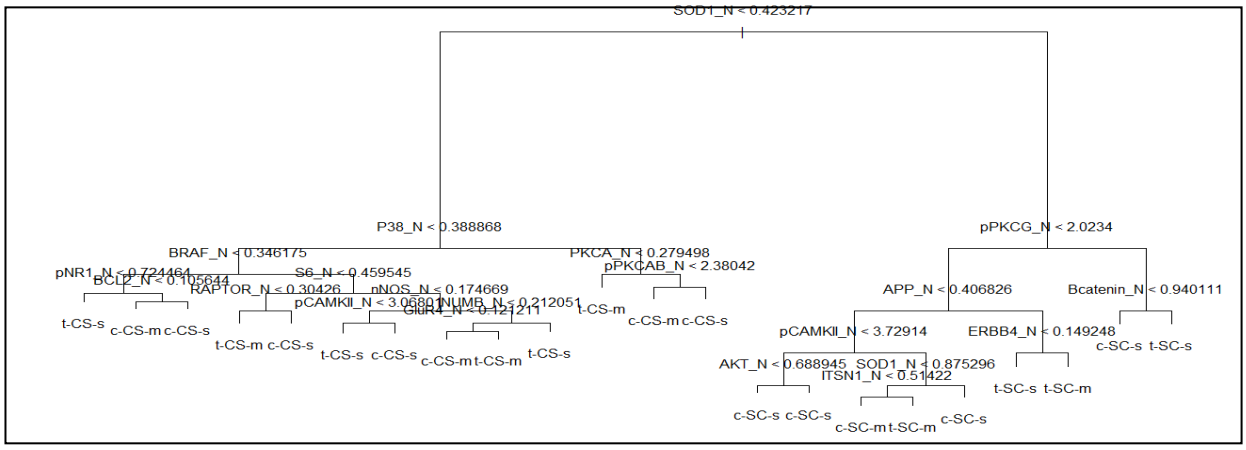
Decision Tree:

Decision tree is a tree structure which looks similar as flow chart. Every node in the tree represents the test of an attribute, every branch represents the output of the test, and every leaf means a class or distribution of classes. The top node is root node, from root node to one leaf consists a classification rule. So decision tree is easy to be transferred to classification rules. It has many algorithms, but the main idea is using from top to bottom induction method. And the most important part is choosing which attribute to be the node as well as the evaluation of whether the tree is correct. There are mainly two steps in decision tree, build a decision tree and do pruning. The thought of building decision tree is called CLS.

In our data set we use tree() function to fit a classification tree, and by using summary() function we will see the number of terminal nodes and training error rate.

```
Classification tree:
tree(formula = as.factor(class) ~ ., data = train)
Variables actually used in tree construction:
 [1] "SOD1_N"     "P38_N"      "BRAF_N"     "pNR1_N"     "BCL2_N"     "S6_N"       "RAPTOR_N"
 [8] "nNOS_N"     "pCAMKII_N"  "NUMB_N"     "GluR4_N"    "PKCA_N"     "pPKCAB_N"   "pPKCG_N"
[15] "APP_N"      "AKT_N"      "ITSN1_N"    "ERBB4_N"    "Bcatenin_N"
Number of terminal nodes:  22
Residual mean deviance:  0.8567 = 675.1 / 788
Misclassification error rate: 0.1617 = 131 / 810
```

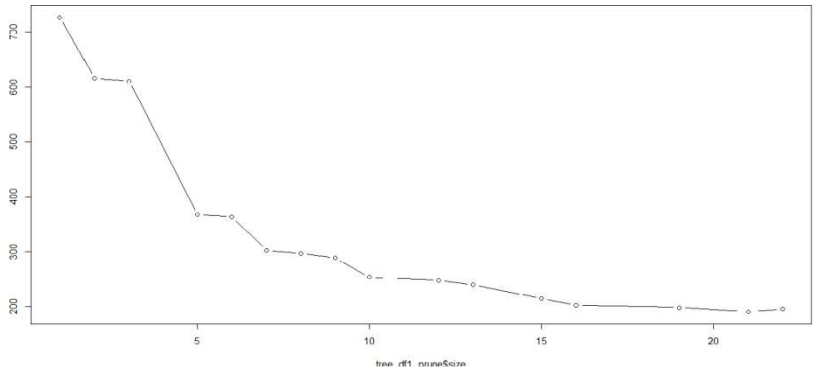
The training error rate is equal to 16% and accuracy is equal to 0.72. Now we display the structure of tree graphically and display the node labels.



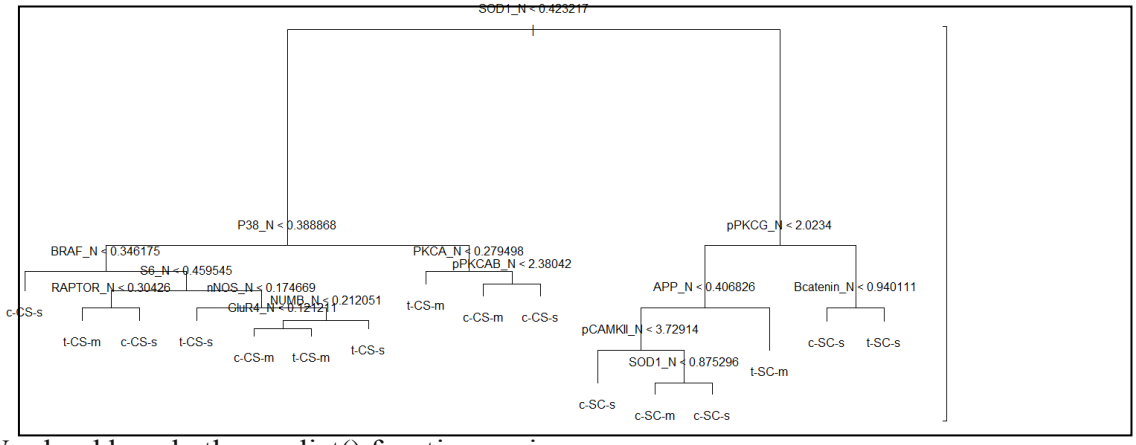
Now, we consider whether pruning the tree might lead to improved results. Pruning is using statistic ways to delete the least reliable branches of a tree to improve the speed and ability of future data classification. The function cv.tree() performs cross-validation in order to determine the optimal level of tree complexity.

```
tree_df1_prune
$size
 [1] 22 21 19 16 15 13 12 10  9  8  7  6  5  3  2  1

$dev
 [1] 195 190 198 202 214 239 247 253 289 296 302 363 367 611 616 727

$k
 [1]  -Inf   0.0   5.0   6.0   8.0   9.5  10.0  11.0  16.0  17.0  30.0  40.0  44.0  74.5  80.0
[16] 106.0

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

Now the tree has 16 nodes with 195 cross-validation error. We plot the error rate as a function of size.



We now apply the prune.misclass() function in order to prune the tree to prune obtain the nine-node tree.



We should apply the predict() function again.

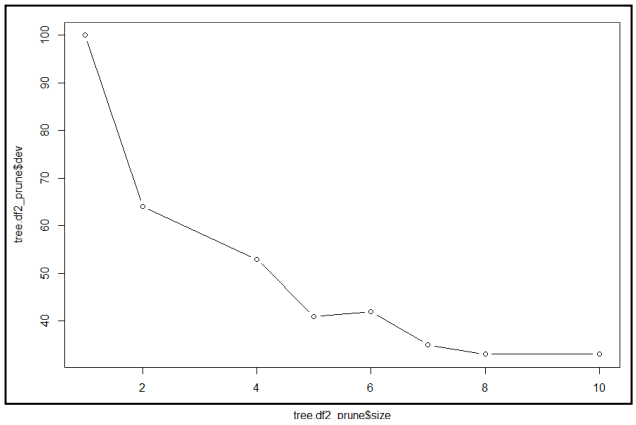
```
> table(test_tree_pred, test$class)

test_tree_pred c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
        c-CS-m     25      6      1      0      1      3      0      0
        c-CS-s     10     17      0      0      5      3      0      1
        c-SC-m      0      0     28      0      0      0      4      0
        c-SC-s      0      0      3     27      0      0      7      0
        t-CS-m      5      0      0      0     14      2      0      0
        t-CS-s      0     11      0      0     11     15      0      0
        t-SC-m      1      2      8      1      0      0     22      7
        t-SC-s      0      0      0      0      0      0      0     30
```

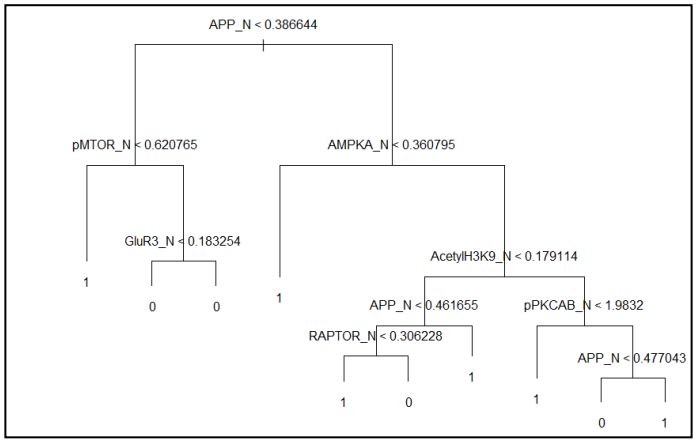
```
Classification tree:
snip.tree(tree = tree_df1, nodes = c(24L, 38L, 50L, 8L, 13L))
Variables actually used in tree construction:
 [1] "SOD1_N"     "P38_N"      "BRAF_N"     "S6_N"       "RAPTOR_N"   "nNOS_N"     "NUMB_N"
 [8] "GluR4_N"    "PKCA_N"     "pPKCAB_N"   "pPKCG_N"    "APP_N"      "pCAMKII_N"  "Bcatenin_N"
Number of terminal nodes:  16
Residual mean deviance:  1.182 = 938.3 / 794
Misclassification error rate: 0.1963 = 159 / 810
```

The misclassification rate has just slightly increased with the pruning of the tree and Classification accuracy in pruning process is 0.65% hence we see by decreasing the number of nodes or value of best, we obtain smaller pruned tree with higher classification accuracy.

Binary classification

For Binary classification , we select Genotype as response , but by using ifelse() function, we create a new variable named Geno and change the value of Genotype. At first , we apply tree() function for binary classification, the accuracy after prediction is equal to 0.85. Then I apply prunning to our model. Like multi classification that describe above, I apply cv.tree() function for cross validation.





Classification accuracy in pruning process is 0.85% . hence we see by decreasing the number of nodes or value of best , we can not see any difference in the accuracy.

```
table(test_tree_pred, df2.test$Geno)(table with prunning)
```

```
test_tree_pred   0  1
             0  388  55
             1   77 360
> table(tree.pred, df2.test$Geno)(table before prunning)

tree.pred   0  1
       0  389  49
       1   76 366
```

Bagging:

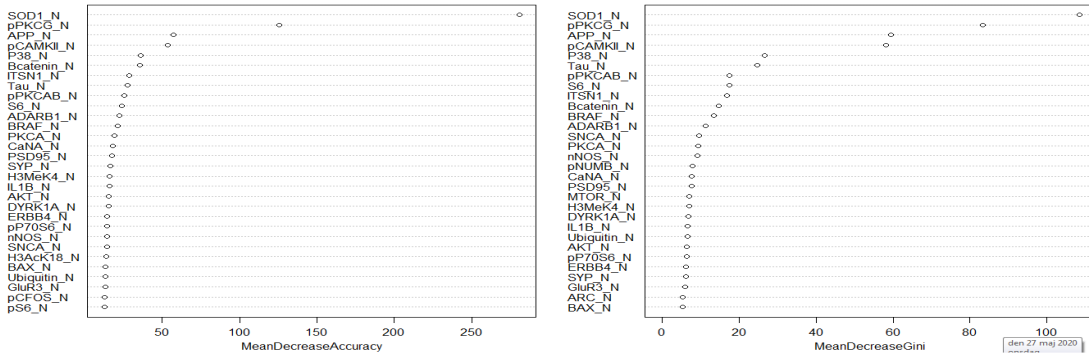
Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method and improve the stability and accuracy of machine learning algorithms, it also avoid over fitting. By using randomForest package in R, we apply bagging and randomForest model to this data set.

```
randomForest(formula = as.factor(class) ~ ., data = train, mtry = 77,      importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 77

        OOB estimate of  error rate: 4.07%
Confusion matrix:
       c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s class.error
c-CS-m    109      0      0      0      0      0      0      0 0.000000000
c-CS-s      4     93      1      0      0      0      0      1 0.060606061
c-SC-m      1      1     98      4      0      0      6      0 0.109090909
c-SC-s      0      0      0    105      0      0      2      0 0.018691589
t-CS-m      1      0      0      0    103      0      0      0 0.009615385
t-CS-s      0      0      0      0      1     81      0      0 0.012195122
t-SC-m      0      0      5      2      0      0     95      0 0.068627451
t-SC-s      0      0      1      0      0      0      3     93 0.041237113
```

By using predict() function , we will see:

```
test.bag c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
  c-CS-m     39      0      0      0      0      0      0      0
  c-CS-s      1     34      0      0      0      0      0      0
  c-SC-m      0      1     39      1      0      0      1      0
  c-SC-s      0      0      0     27      0      0      1      0
  t-CS-m      0      0      0      0     31      0      0      0
  t-CS-s      0      1      0      0      0     23      0      0
  t-SC-m      1      0      1      0      0      0     31      0
  t-SC-s      0      0      0      0      0      0      0     38
```

By using mean function , we find accuracy of this model : 0.9703704



RandomForest:

For randomForest we use smaller value for mtry argument. mtry value is the number of variables available for splitting at each tree node. The default value of this parameter depends on which R package is used to fit the model: in randomForest - For classification models, the default is the square root of the number of predictor variables (rounded down). Hence for 77 predictors, we select mtry=8, then apply randomForest() function.

```
No. of variables tried at each split: 8

        OOB estimate of  error rate: 0.99%
Confusion matrix:
       c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s class.error
c-CS-m    109      0      0      0      0      0      0      0 0.000000000
c-CS-s      3     95      0      0      1      0      0      0 0.040404040
c-SC-m      0      0    107      0      0      0      3      0 0.027272727
c-SC-s      0      0      0    107      0      0      0      0 0.000000000
t-CS-m      0      0      0      0    104      0      0      0 0.000000000
t-CS-s      0      0      0      0      0     82      0      0 0.000000000
t-SC-m      0      0      1      0      0      0    101      0 0.009803922
t-SC-s      0      0      0      0      0      0      0     97 0.000000000
```

By using predict() function, we will see:

```
test.forest c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
     c-CS-m     41      0      0      0      0      0      0      0
     c-CS-s      0     36      0      0      0      0      0      0
     c-SC-m      0      0     40      0      0      0      0      0
     c-SC-s      0      0      0     28      0      0      1      0
     t-CS-m      0      0      0      0     30      0      0      0
     t-CS-s      0      0      0      0      1     23      0      0
     t-SC-m      0      0      0      0      0      0     32      0
     t-SC-s      0      0      0      0      0      0      0     38
```

By using mean function , we find accuracy of this model : 0.9925926

Boosting:

Boosting is another approach to improve the predictions resulting from a decision tree. It can be applied to many statistical learning methods for regression or classification. The main concept of this method is to improve (boost) the model accuracy with a combined model. There are several boosting algorithms such as Gradient boosting, AdaBoost (Adaptive Boost), XGBoost and others. Here I classify data with a gbm (Generalized Boosted Model) package's gbm (Gradient Boosting Model) method. I use classification with caret train method. I use caret train method for model fitting. Train method requires train control parameter and then we'll define the model and train it with train data. Next we can predict test data with the fitted model. The output of predict is as below:

```
pred     c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
  c-CS-m     41      1      0      0      0      0      0      0
  c-CS-s      0     35      0      0      2      1      0      0
  c-SC-m      0      0     40      0      0      0      0      0
  c-SC-s      0      0      0     28      0      0      0      0
  t-CS-m      0      0      0      0     29      0      0      0
  t-CS-s      0      0      0      0      0     22      0      0
  t-SC-m      0      0      0      0      0      0     33      0
  t-SC-s      0      0      0      0      0      0      0     38
```

By using mean function , we find accuracy of this model : 0.9851852

Support Vector Machine (SVM)

SVM is a supervised learning algorithm to use for classification and regression . SVM has two types named as Linear SVM and NonLinear SVM classification using to classify binary and multiclass problems respectively. SVM finds an optimal hyperplane to classify new samples. We can perform cross-validation using tune() to select the best choice of γ and cost for a SVM with a radial kernel:

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
   10   0.5

- best performance: 0.4333333

- Detailed performance results:
    cost gamma     error dispersion
1  1e-01   0.0 0.8975309 0.02794114
2  1e+00   0.0 0.8975309 0.02794114
3  1e+01   0.0 0.8975309 0.02794114
4  1e+02   0.0 0.8975309 0.02794114
         5  1e+03   0.0 0.8975309 0.02794114
         6  1e-01   0.5 0.8901235 0.04333707
7  1e+00   0.5 0.4777778 0.09312604
8  1e+01   0.5 0.4333333 0.08949080
9  1e+02   0.5 0.4333333 0.08949080
10 1e+03   0.5 0.4333333 0.08949080
11 1e-01   1.0 0.8962963 0.02978924
12 1e+00   1.0 0.8061728 0.06911865
13 1e+01   1.0 0.7765432 0.07138465
14 1e+02   1.0 0.7765432 0.07138465
15 1e+03   1.0 0.7765432 0.07138465
16 1e-01   2.0 0.8975309 0.02794114
17 1e+00   2.0 0.8913580 0.03625138
18 1e+01   2.0 0.8864198 0.04106988
19 1e+02   2.0 0.8864198 0.04106988
20 1e+03   2.0 0.8864198 0.04106988
21 1e-01   3.0 0.8975309 0.02794114
22 1e+00   3.0 0.8962963 0.02978924
23 1e+01   3.0 0.8962963 0.02978924
24 1e+02   3.0 0.8962963 0.02978924
25 1e+03   3.0 0.8962963 0.02978924
26 1e-01   4.0 0.8975309 0.02794114
27 1e+00   4.0 0.8962963 0.02978924
28 1e+01   4.0 0.8962963 0.02978924
29 1e+02   4.0 0.8962963 0.02978924
30 1e+03   4.0 0.8962963 0.02978924
```

Therefore, the best choice of parameters involves cost=10 and gamma=0.5, MSE equal to 0.43 and RMSE equal to 0.65, We can view the test set predictions for this model by applying the predict() function to the data. We will see that the our best fitting model produces 63% accuracy in identifying classes.

```
        pred
true     c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
  c-CS-m     29      0     12      0      0      0      0      0
  c-CS-s      0     19     17      0      0      0      0      0
  c-SC-m      0      0     40      0      0      0      0      0
  c-SC-s      0      0      8     20      0      0      0      0
  t-CS-m      0      0      9      0     22      0      0      0
  t-CS-s      0      0     19      0      0      4      0      0
  t-SC-m      0      0     16      0      0      0     17      0
  t-SC-s      0      0     18      0      0      0      0     20
```

Now, we perform cross-validation using tune() to select the best choice of γ and cost for a SVM with a polynomial kernel:

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
   10      3

- best performance: 0.002469136

- Detailed performance results:
    cost degree       error  dispersion
1  1e-01      0 0.897530864 0.027941144
2  1e+00      0 0.897530864 0.027941144
3  1e+01      0 0.897530864 0.027941144
```

```
4  1e+02      0 0.897530864 0.027941144
5  1e+03      0 0.897530864 0.027941144
6  1e-01      1 0.267901235 0.050417821
7  1e+00      1 0.025925926 0.022875625
8  1e+01      1 0.007407407 0.008632172
9  1e+02      1 0.007407407 0.008632172
10 1e+03      1 0.007407407 0.008632172
11 1e-01      2 0.553086420 0.084697717
12 1e+00      2 0.011111111 0.010809815
13 1e+01      2 0.004938272 0.008632172
14 1e+02      2 0.004938272 0.008632172
15 1e+03      2 0.004938272 0.008632172
16 1e-01      3 0.477777778 0.088845059
17 1e+00      3 0.049382716 0.020983619
18 1e+01      3 0.002469136 0.005205395
19 1e+02      3 0.003703704 0.008332698
20 1e+03      3 0.003703704 0.008332698
21 1e-01      4 0.619753086 0.068565154
22 1e+00      4 0.202469136 0.050131497
23 1e+01      4 0.020987654 0.014314837
24 1e+02      4 0.014814815 0.017264343
25 1e+03      4 0.014814815 0.017264343
```

Therefore, the best choice of parameters involves cost=1 and degree=3, MSE=0.0025 and RMSE= 0.05, We can view the test set predictions for this model by applying the predict() function to the data. We will see that the our best fitting model produces 99% accuracy in identifying classes.

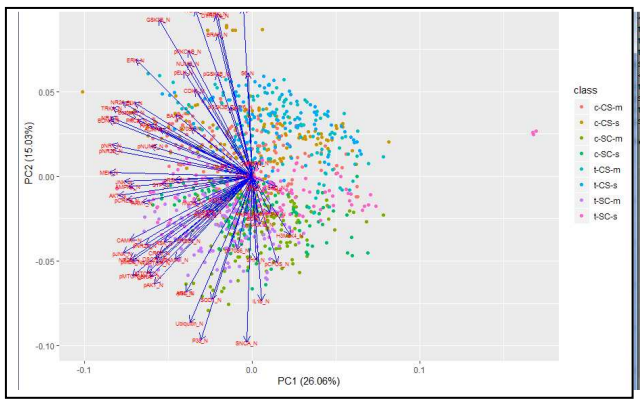
```
        pred
true     c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
  c-CS-m     41      0      0      0      0      0      0      0
  c-CS-s      0     36      0      0      0      0      0      0
  c-SC-m      0      0     40      0      0      0      0      0
  c-SC-s      0      0      0     27      0      0      0      1
  t-CS-m      0      0      0      0     31      0      0      0
  t-CS-s      0      0      0      0      0     23      0      0
  t-SC-m      0      0      0      0      0      0     33      0
  t-SC-s      0      0      0      0      0      0      0     38
```

Compare the results of models:

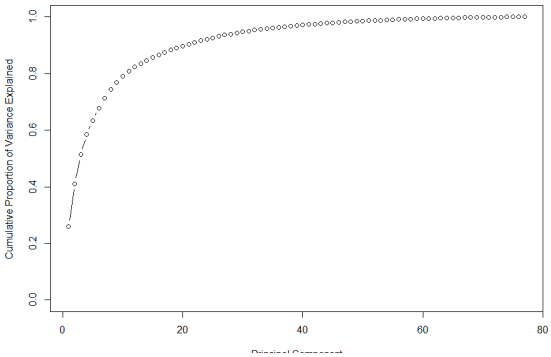
Model	Accuracy
Decision Tree	**0.65**
Bagging	**0.97**
Random Forest	**0.99**
Boosting	**0.98**
SVM with radial kernel	**0.63**
SVM with polynomial kernel	**0.99**

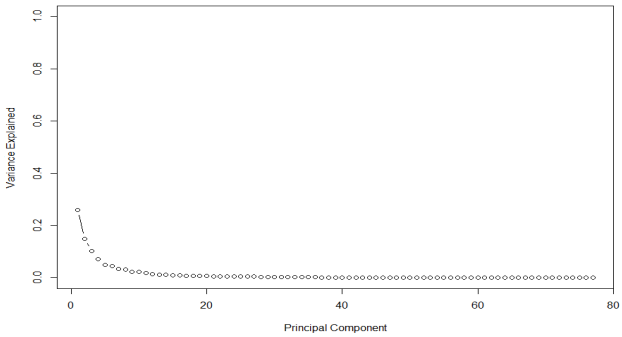
Performing Principal Component Analysis:

PCA is used when we're working with "wide" data sets. In such cases, where many variables are present, we cannot easily plot the data in its raw format, making it difficult to get a sense of the trends present within. PCA allows us to see the overall "shape" of the data, identifying which samples are similar to one another and which are very different. This can enable us to identify groups of samples that are similar and work out which variables make one group different from another. We create a new data frame that consists all our numeric independent variables. We will have a matrix of 77 columns and 1080 rows. By using prcomp() function, we apply PCA on our dataset. By applying summary() function , we see that we have 77 PCA.

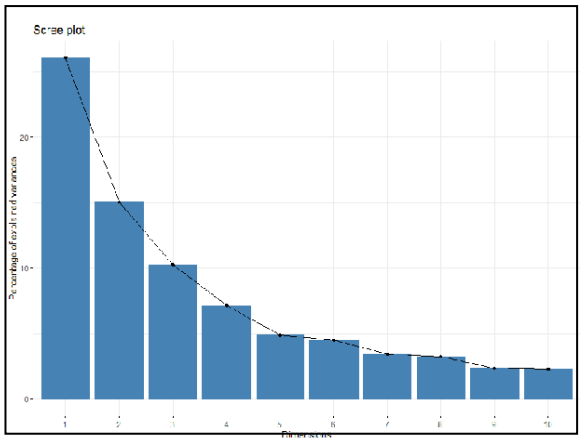


Below we see a scree plot depicting the proportion of variance explained by each of the 77 principal components in the df.cluster dataset.
And below we see the cumulative proportion of variance explained by the 77 principal components in the df.cluster dataset.





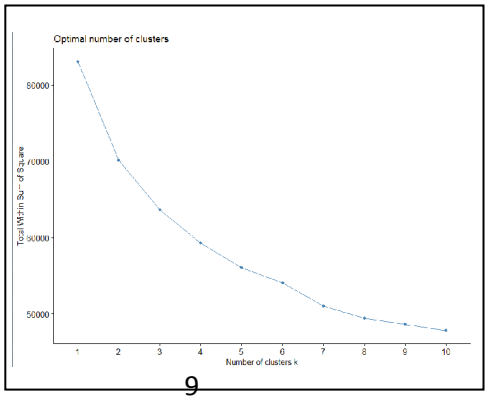
The most common technique for determining how many principal components to keep is eyeballing the scree plot, which is shown above and stored in the ggplot object PVEplot. To determine the number of components, we look for the "elbow point", where the PVE significantly drops off. The above plots shows that 12 components results in variance close to ~ 98%. Therefore, in this case, we'll select number of components as 12 [PC1 to PC12] and proceed to the modeling stage.
Visualize eigenvalues (scree plot). Show the percentage of variances explained by each principal component.



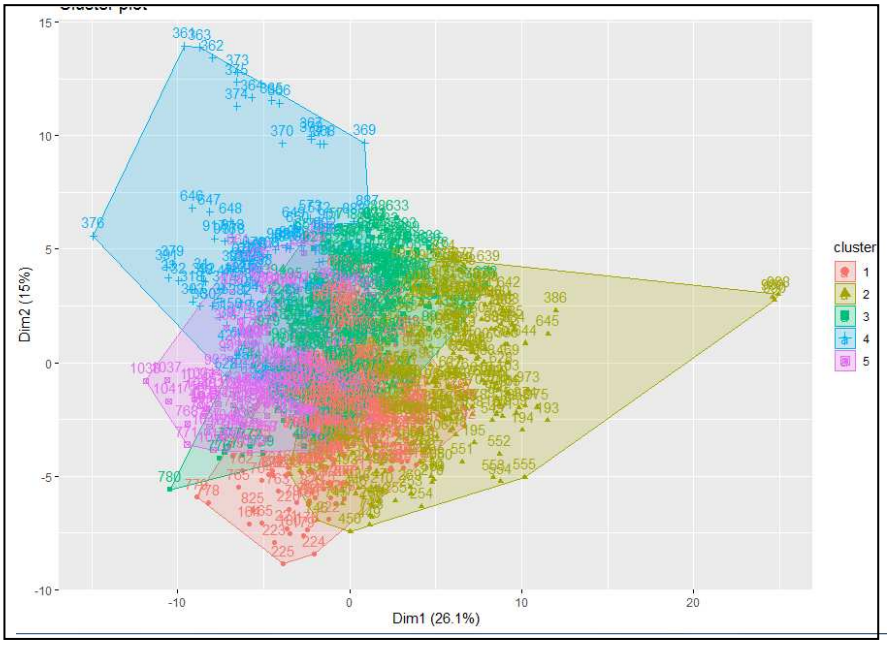
Now by considering the PCA number=12, we make a data frame and consider the 12 first pca and use this feature reduction to apply k_means clustering.

k_means clustering:

Determining the optimal number of clusters in a data set is a fundamental issue in partitioning clustering, such as k-means clustering, which requires the user to specify the number of clusters k to be generated. The Elbow method looks at the total WSS as a function of the number of clusters: One should choose a number of clusters so that adding another cluster doesn't improve much better the total WSS.



Through the elbow method , we consider the number of cluster equal to 5.



By using table we can see what each cluster describes:

```
table(km.out$cluster, df$Treatment)
```

```
   Memantine Saline
1         54     58
2        120    143
3        181     58
4         89     63
5        126    188
```

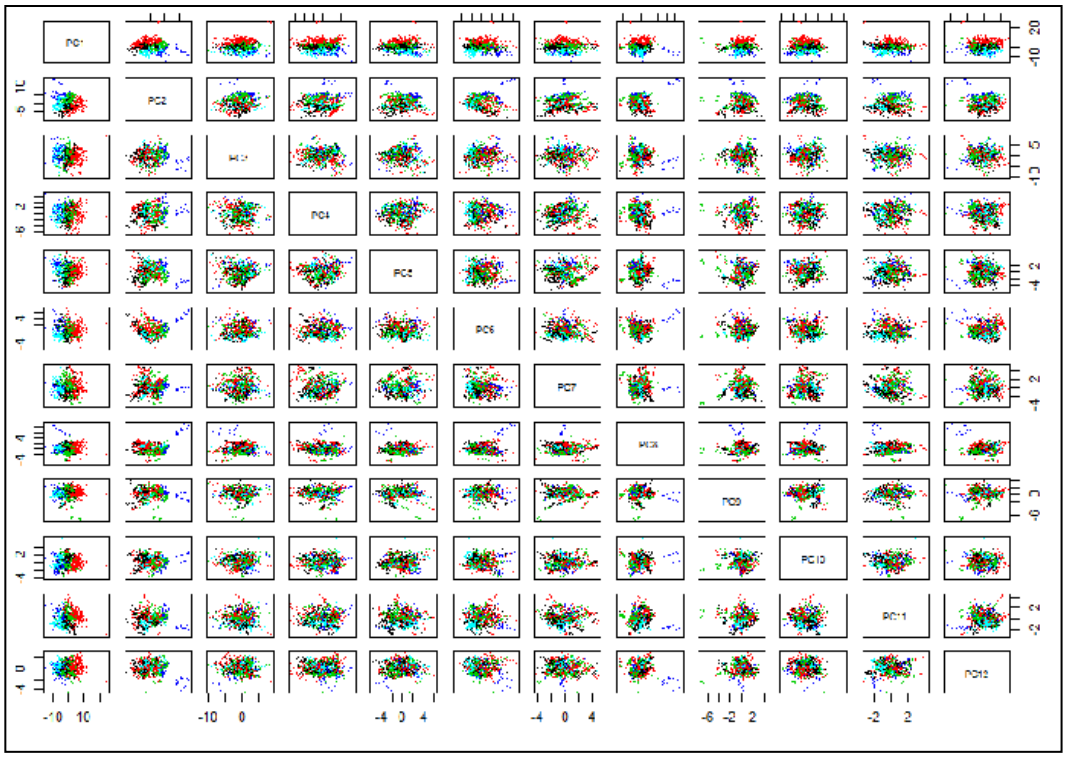
```
table(km.out$cluster, df$Behavior)
```

```
   C/S S/C
1 106   6
2 137 126
3  36 203
4  29 123
5 217  97
```

```
table(km.out$cluster, df$Genotype)
```

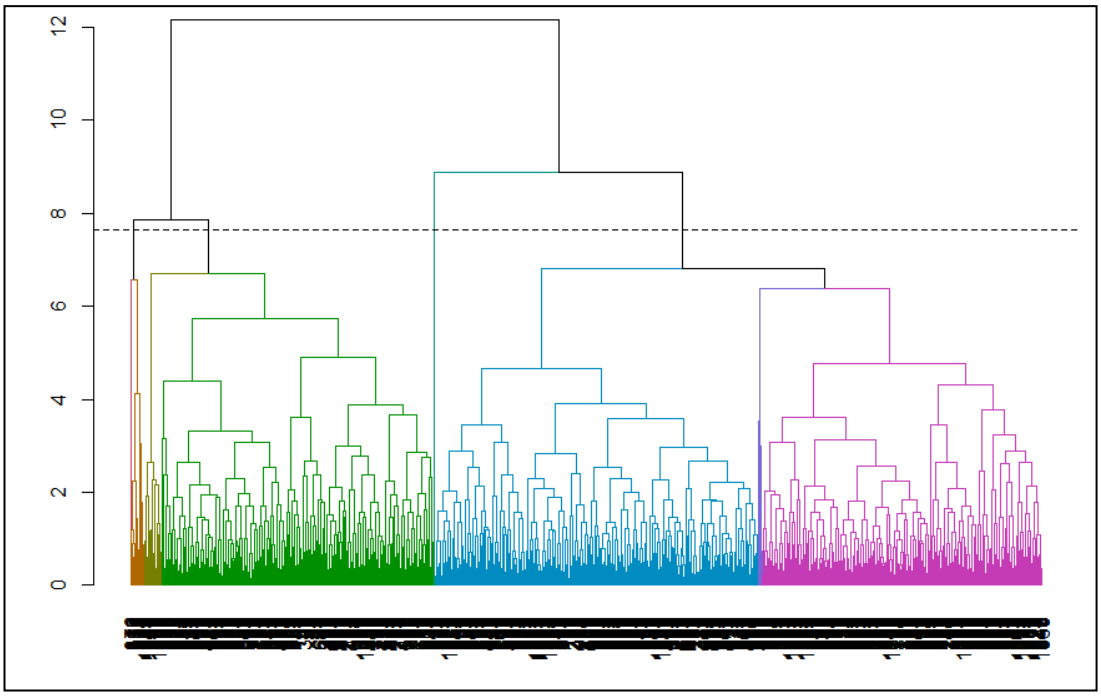
```
   Control Ts65Dn
1       88     24
2      115    148
3      131    108
4       52    100
5      184    130
```

Now by having the number of cluster=5 and the number of PCA=12, we apply clustering by these new data.



Hierarchical clustering:

This algorithm produces a hierarchy for the observations of the dataset. For assessing the similarity between observations, a distance matrix is required to quantify closeness. Referring the dendrogram, a suitable height shall be chosen where a horizontal line shall be plotted and the number of instances where the horizontal line cut the branches of dendrogram are clusters of our data. So, our hierarchical clustering analysis is producing 8 clusters.



In the dendrogram displayed above, each leaf corresponds to one observation. As we move up the tree, observations that are similar to each other are combined into branches, which are themselves fused at a higher height.The height of the fusion, provided on the vertical axis, indicates the (dis)similarity between two observations. The higher the height of the fusion, the less similar the observations are. Now we want to use the table function to see how many observations are in each cluster. (I show here two tables related to behavior and Treatment)

```
table(groups, df$Behavior)
```

```
groups C/S S/C
     1 248  74
     2 181 201
     3  77 254
     4   3  18
     5   0   5
     6  15   0
     7   1   0
     8   0   3
```

```
table(groups, df$Treatment)
```

```
groups Memantine Saline
     1       142    180
     2       170    212
     3       250     81
     4         7     14
     5         1      4
     6         0     15
     7         0      1
     8         0      3
```