Data analysis in R and KNIME

Task1

Analysis in R

First, we will load the data into R, and show the summary of all variables:

```
teeth <- read.csv('teeth.csv', header = TRUE, row.names = 1)
summary(teeth)
```

```
##      TopInc          BotInc          TopCan           BotCan      
##  Min.   :0.000   Min.   :1.000   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:1.000   1st Qu.:1.500   1st Qu.:0.5000   1st Qu.:0.0000  
##  Median :2.000   Median :3.000   Median :1.0000   Median :1.0000  
##  Mean   :2.097   Mean   :2.419   Mean   :0.7419   Mean   :0.6452  
##  3rd Qu.:3.000   3rd Qu.:3.000   3rd Qu.:1.0000   3rd Qu.:1.0000  
##  Max.   :3.000   Max.   :4.000   Max.   :1.0000   Max.   :1.0000  
##      TopPre          BotPre          TopMol          BotMol     
##  Min.   :0.000   Min.   :0.000   Min.   :1.000   Min.   :1.000  
##  1st Qu.:2.000   1st Qu.:2.000   1st Qu.:1.000   1st Qu.:2.000  
##  Median :3.000   Median :3.000   Median :3.000   Median :3.000  
##  Mean   :2.806   Mean   :2.677   Mean   :2.194   Mean   :2.419  
##  3rd Qu.:4.000   3rd Qu.:3.500   3rd Qu.:3.000   3rd Qu.:3.000  
##  Max.   :4.000   Max.   :4.000   Max.   :3.000   Max.   :3.000
```

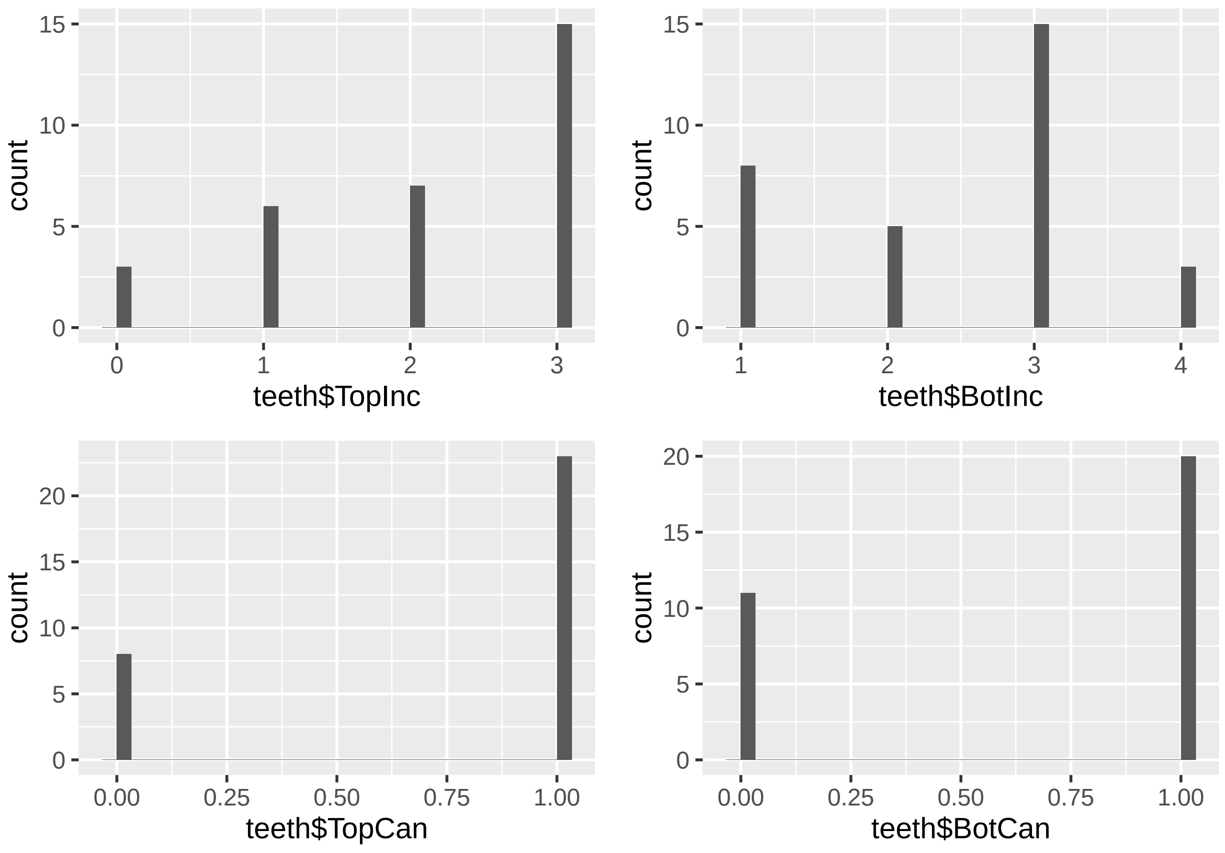
In this dataset there is 31 observations - animals. As we can see, 8 numerical variables is present for each animal. All of the variables have integer values. Range for each of the variables is easily calculated as:

```
sapply(teeth, max) - sapply(teeth, min)
```

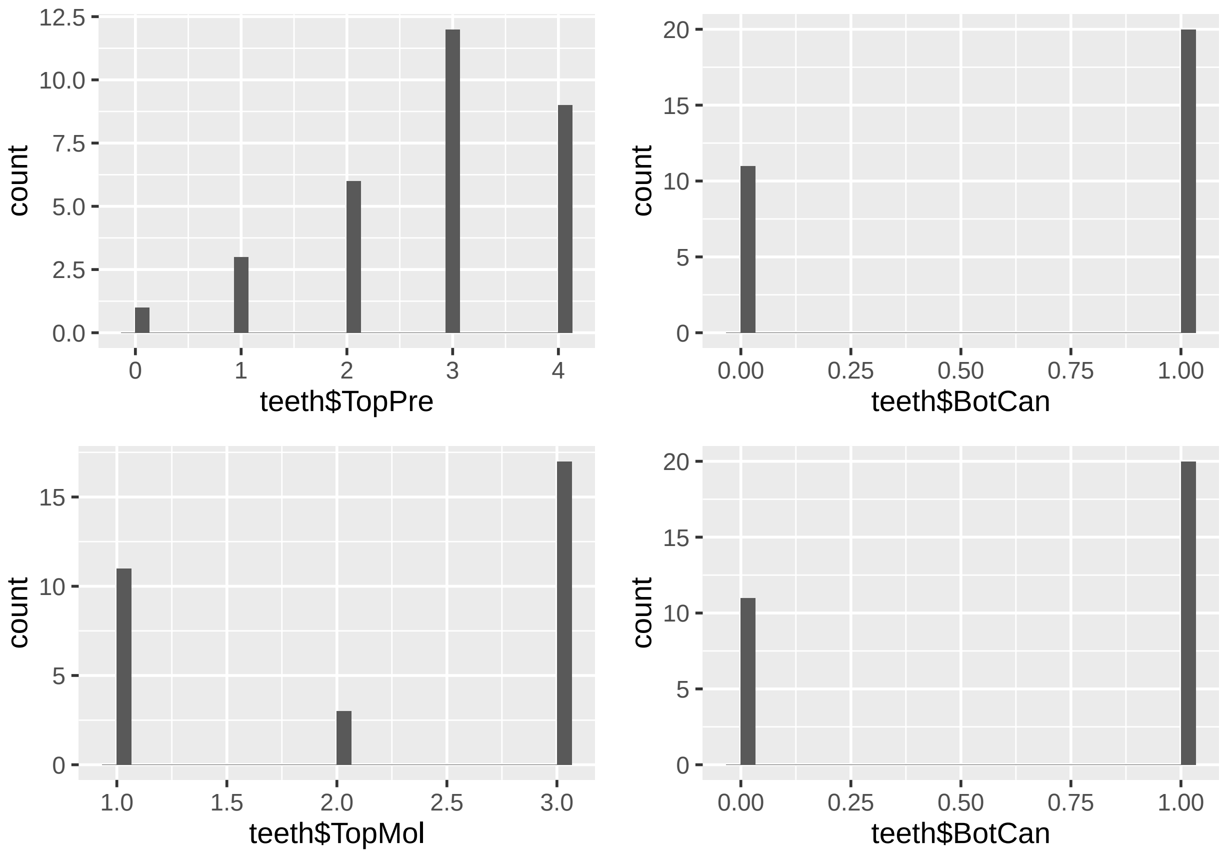
```
## TopInc BotInc TopCan BotCan TopPre BotPre TopMol BotMol 
##      3      3      1      1      4      4      2      2
```

Histograms of each attributes:

```
  library(ggplot2)
  library(gridExtra)
  p1 <- qplot(teeth$TopInc, geom = "histogram")
  p2 <- qplot(teeth$BotInc, geom = "histogram")
  p3<- qplot(teeth$TopCan, geom = "histogram")
  p4 <- qplot(teeth$BotCan, geom = "histogram")
  p5 <- qplot(teeth$TopPre, geom = "histogram")
  p6 <- qplot(teeth$BotCan, geom = "histogram")
  p7 <- qplot(teeth$TopMol, geom = "histogram")
  p8 <- qplot(teeth$BotCan, geom = "histogram")
  grid.arrange(p1, p2, p3, p4, ncol =2)
```



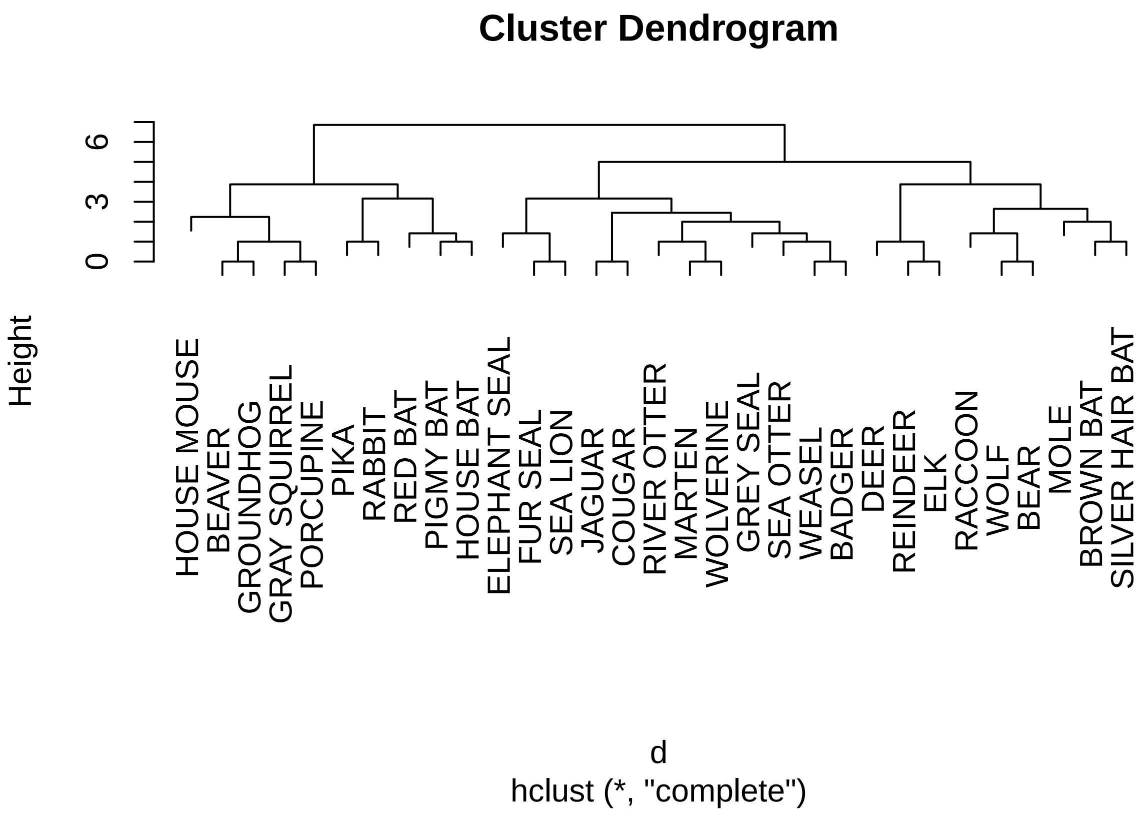
```r
grid.arrange(p5, p6, p7, p8, ncol =2)
```



Histograms provide us a nice overview of the distributions of values inside of the variables.

In order to create clusters of observations, we used the function 'hclust' in R. First we created a distance matrix for the observations. The distances are calculated as Eucleadian distances. Distance matrix is then used for clustering - the 'hclust' algorithm uses the complete linkage method.

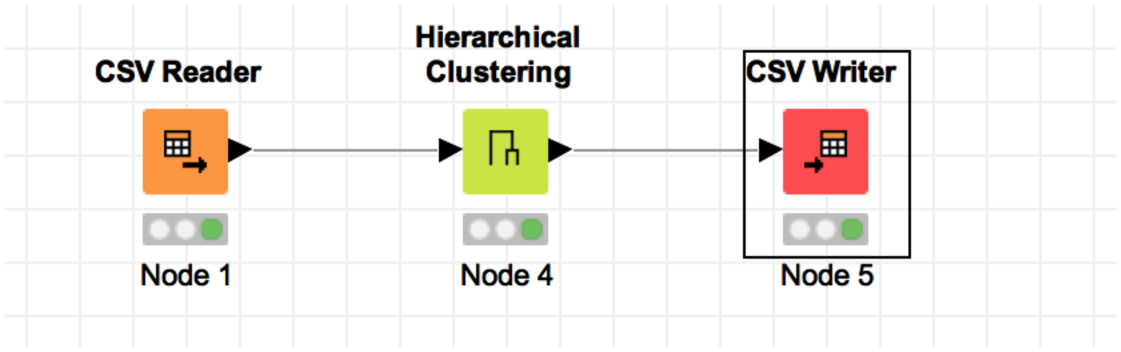
```r
d <- dist(as.matrix(teeth))
hc <-hclust(d)
plot(hc)
```



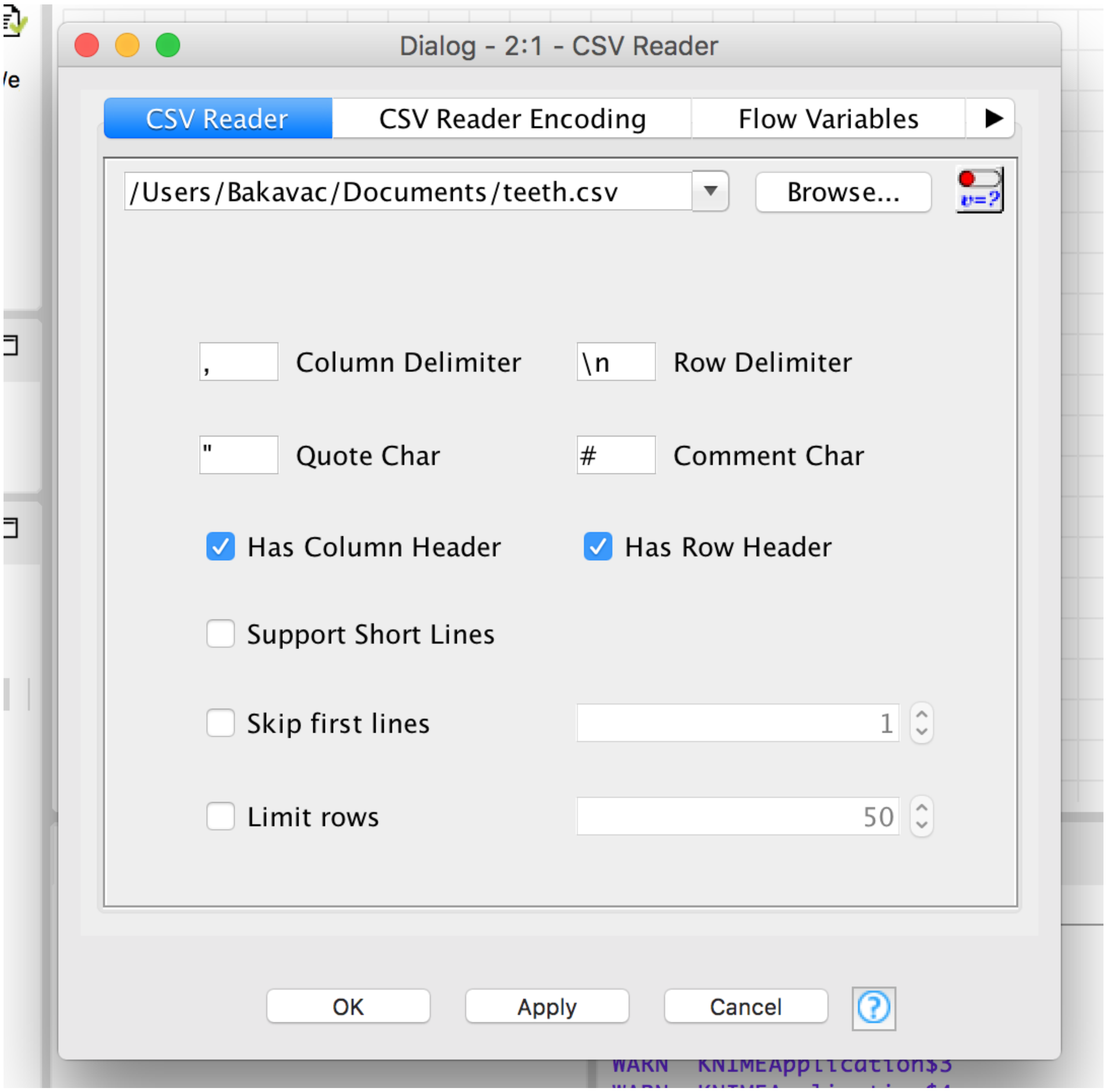
Clustering in KNIME

Similar to what we presented in the previous part, where hierarchical clustering has been done in R, we will show how we created clusters and dendogram in KNIME.

The workflow for hierarchical clustering in KNIME is quite simple - we have a node to read the data, one node for clustering, and additional node to write the clustering results to .csv file.



The CSV Reader node was set up to read the ‘teeth.csv’ file, with ‘,’ as a delimiter and to count on column and and row headers as shown on the screenshot below.


Dialog - 2:1 - CSV Reader

CSV Reader | CSV Reader Encoding | Flow Variables

/Users/Bakavac/Documents/teeth.csv

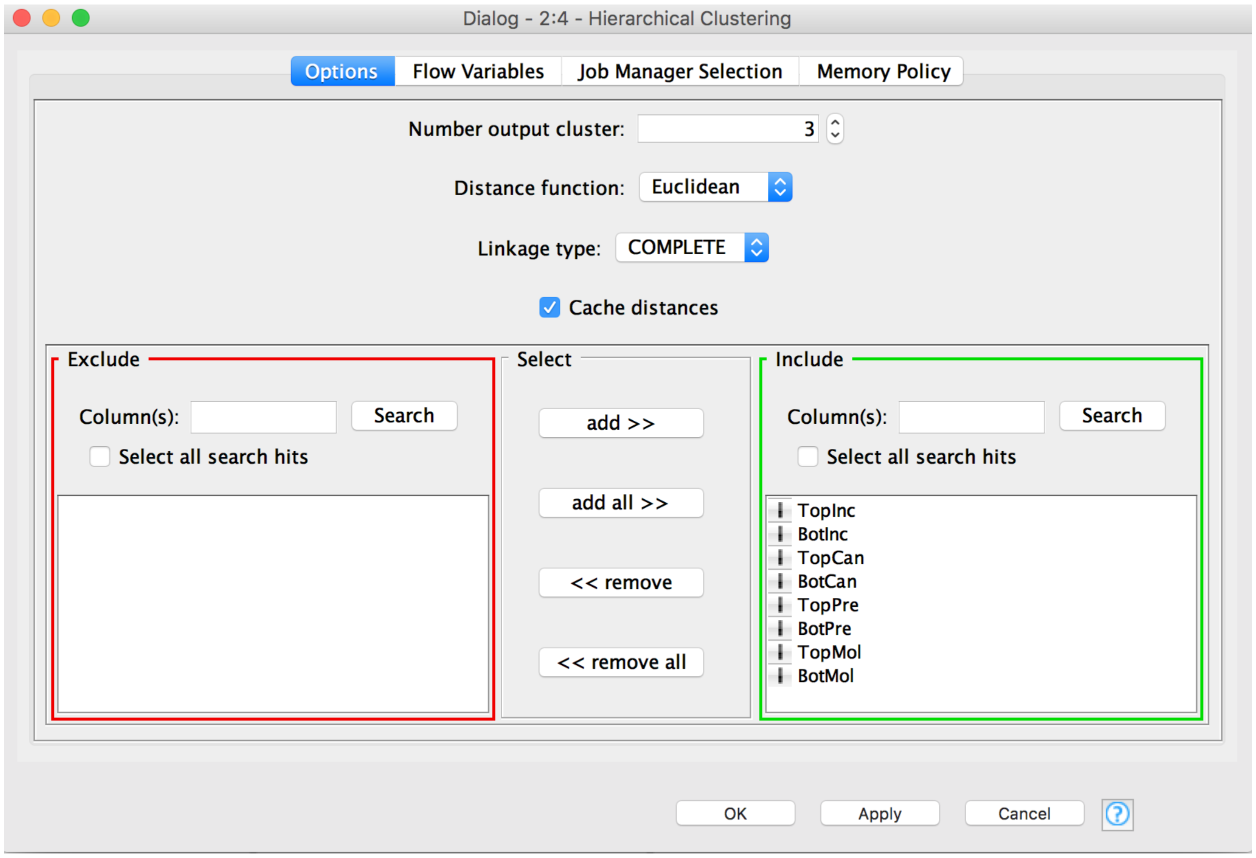
Browse...

,	Column Delimiter	\n	Row Delimiter
"	Quote Char	#	Comment Char

- [✓] Has Column Header
- [✓] Has Row Header
- [ ] Support Short Lines
- [ ] Skip first lines: 1
- [ ] Limit rows: 50

OK | Apply | Cancel

The Hierarchical Clustering node was set up to produce 3 clusters, using the Euclidean distance and complete linkage, same as in the R analysis. Configuration window is shown below.



Dialog - 2:4 - Hierarchical Clustering

Options | Flow Variables | Job Manager Selection | Memory Policy

Number output cluster: 3

Distance function: Euclidean

Linkage type: COMPLETE

[✓] Cache distances

Exclude

Column(s): Search

[ ] Select all search hits

Select

add >>

add all >>

<< remove

<< remove all

Include

Column(s): Search

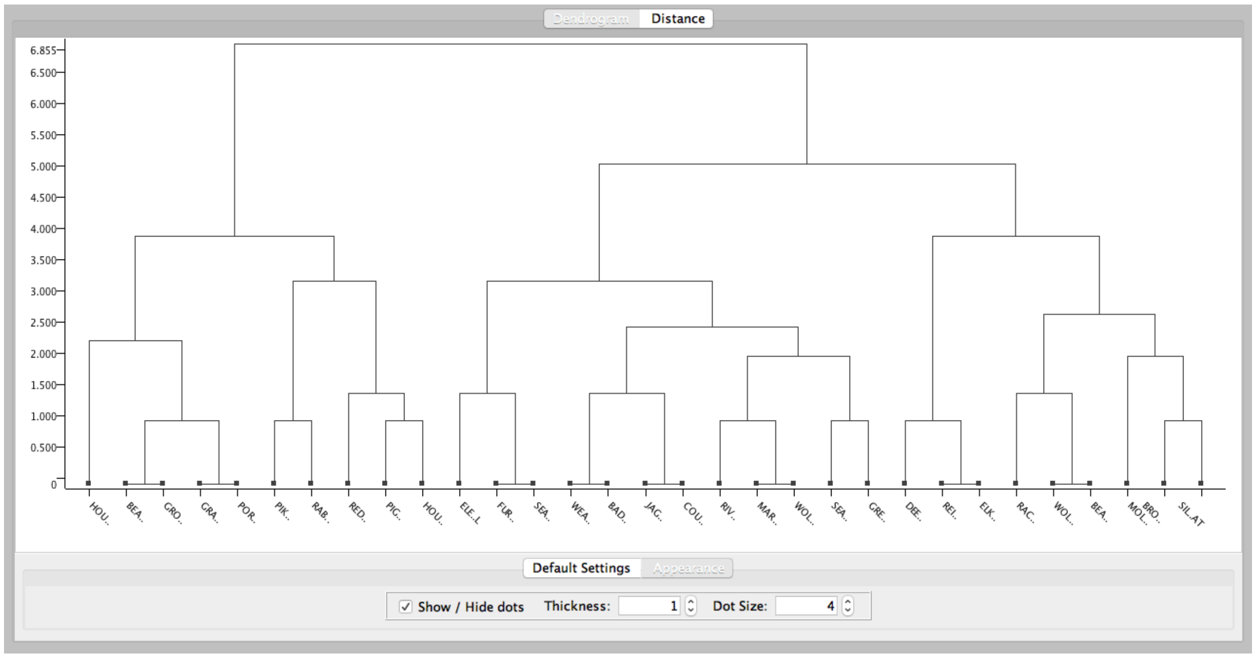
[ ] Select all search hits

TopInc
BotInc
TopCan
BotCan
TopPre
BotPre
TopMol
BotMol

OK | Apply | Cancel

The third node was simply configured with the path to write the output results.

After running the program, in order to see the dendogram, we can right-click on the 'Hierarchical Clustering' node and choose "View: Dendogram/Distance View" which gives us the produced dendoogram shown below. We can notice it is the same as the one we obtained in R.



Problem 2 - Data classification in R

In this problem we will classify data and compare different evaluation methods. In all cases, 'Decision Trees' will be used as classification algorithm. Evaluation methods compared are Resusbstitution error, hold-out-10%, 10-fold-cross-validation, and leave-one-out-cross-validation.

Wine data First we will do the classification of the wine data. First we have to load the data and do rename the columns based on the provided code-book. We will make the 'class' variable as factor, to be suitable for the algorithm.

```r
#load the data from the file and rename the columns
wine = read.csv("wine.data.txt")
names(wine) <- c("class","Alcohol","Malic_acid", "Ash", "Alcalinity_of_ash"
                 , "Magnesium","Total_phenols","Flavanoids","Nonflavanoid_phenols"
                 ,"Proanthocyanins","Color_intensity","Hue"
                 ,"OD280/OD315_of_diluted_wines","Proline")
wine$class <- factor(wine$class) #transform the class into factor type,
#suitable for the prediction
```

Resusbstitution error In this part, we will use the whole dataset to train and test the model. This kind of training and prediction is not usually good as the test data is already 'known' to the model, hence accuracy is not showing the true state.

```r
library(rpart)
#Train and predict on the whole set with decision trees,
#calculate the accuracy - repeated 20 times
acc_resub = rep(NA,20)
for (i in 1:20){
  wine_model_resub = rpart(class~., data=wine) #form the model
  class_predict_resub = predict(wine_model_resub, wine, type = "class") #make the predictions
  tab_resub <-table(class_predict_resub, wine$class) #table of true end predicted classes
  acc_resub[i] = sum(diag(tab_resub))/sum(tab_resub) #itteration accuracy
}
wine_acc_resub_mean <- (mean(acc_resub)) #mean accuracy
wine_acc_resub_sd <- (sd(acc_resub)) #standard devation of the accuracy
```

Hold-out-10% Next, we will split our data in the way that 90% is used for the training and 10% for the testing. This way, we are presenting our model with the new data, and the accuracy will better show how the model is behaving with the unknown data.

```r
library(caret)
```

```
## Loading required package: lattice
```

```r
#Train and predict on the 90% of the set with decision trees,
#calculate the accuracy - repeated 20 times
acc_HO = rep(NA,20)
for (i in 1:20){
  #create partition of 90% of the data based on the class
  train_index_HO <- createDataPartition(wine$class, p = .9, list = FALSE)
  #split the data on the train and test part
  wine_train_HO <- wine[train_index_HO,]
  wine_test_HO <- wine[-train_index_HO,]
  wine_model_HO <- rpart(class~., data = wine_train_HO)
  class_predict_HO = predict(wine_model_HO, wine_test_HO, type="class")
  tab_HO <- table(class_predict_HO, wine_test_HO$class)
  acc_HO[i] <- sum(diag(tab_HO))/sum(tab_HO)
}
wine_acc_HO_mean <- mean(acc_HO)
wine_acc_HO_sd <- sd(acc_HO)
```

10-fold-cross-validation In the third case, we will use cross validation, with 10 folds. This means we will split our data in 10 parts, train the model on 9 of them and test on the one that has not been used in training. This will be repeated 10 times, every time leaving different part of the data for testing and training on the rest of the set.

```r
#Preparing the CV - spliting data into blocks
n <- nrow(wine)
K <- 10 #number of folds
n_block <- n%/%K+1
acc_CV10 <- rep(NA, 20)

for (i in 1:20){
  #creating new blocks
  rand = runif(n)
  rang <- rank(rand)
  block <- (rang-1)%/%n_block
  block<- as.factor(block)
  acc <- numeric(0)
  for (k in 1:K-1){
    #training and prediction
    model_k <- rpart(class~., data = wine[block!=k,])
    pred_k <- predict(model_k, wine[block==k,], type = "class")
    tab_k <- table(pred_k, wine[block==k,]$class)
    acc_k <- sum(diag(tab_k))/sum(tab_k)
    acc <- rbind(acc, acc_k)
  }
  #calculating CV10 one itteration accuracy
 acc_CV10[i] <- mean(acc)
}

wine_acc_CV10_mean <- (mean(acc_CV10))
wine_acc_CV10_sd <- (sd(acc_CV10))
```

Leave-one-out-cross-validation Finally, leave-one-out-cross validation will be performed. In this scenario, every time we will exclude one observation from the training set, and perform the training on the rest. We will use only the one excluded as the test set. This is very realistic method for model evaluation, but it can take a lot of time.

```r
pred_k <-rep(NA,n)
acc_LOOCV <- rep(NA,20)
for (i in 1:20){
  for (k in 1:n){
    #train and predict with excluding one by one of row from training and predicting for it
    model_k <- rpart(class~., data = wine[-k,])
    pred_k[k] <- predict(model_k, wine[k,], type = "class")
  }
  tab_k <- table(pred_k, wine$class)
  acc_LOOCV[i] <- sum(diag(tab_k))/sum(tab_k)
}
wine_acc_LOOCV_mean <- mean(acc_LOOCV)
wine_acc_LOOCV_sd <- sd(acc_LOOCV)
```

Comparison of the evaluation methods Now we can compare all four methods. In the table below, we can see that the accuracy is highest for the Resubstitution error method, which was expected since it was measured on the whole training set. LOOCV method has the lowest accuracy, which is probably closest to the real situation since it was tested separately on all cases present in the dataset.

```r
wine_means <- cbind(wine_acc_resub_mean, wine_acc_HO_mean
                    , wine_acc_CV10_mean, wine_acc_LOOCV_mean)
wine_sds <- cbind(wine_acc_resub_sd, wine_acc_HO_sd
                  , wine_acc_CV10_sd, wine_acc_LOOCV_sd)

wine_stats <-rbind(wine_means, wine_sds)
colnames(wine_stats) <- c("Resub", "Hold_Out_10", "10_fold_CV", "LOOCV")
row.names(wine_stats) <- c("Mean", "Std_dev")
wine_stats
```

```
##              Resub Hold_Out_10 10_fold_CV     LOOCV
## Mean     0.9378531  0.86875000 0.87505556 0.8305085
## Std_dev  0.0000000  0.04488655 0.01903956 0.0000000
```

Cancer data

In this part we will perform the same as for the wine data. It will not be explain again, since everything holds as in that data.

Importing and preparing the data First we will import the data and prepare it.

```r
cancer = read.csv("breast-cancer-wisconsin.data.txt")
names(cancer) <- c("Sample_number","Clump_Thickness", "Uniformity_of_Cell_Size"
                   , "Uniformity_of_Cell_Shape","Marginal_Adhesion"
                   , "Single_Epithelial_Cell_Size", "Bare_Nuclei"
                   , "Bland_Chromatin","Normal_Nucleoli","Mitoses", "class")
cancer$class <- factor(cancer$class)
```

Resubstitution error Similar to the wine data, we will train and test on the whole dataset to obtain the resubstitution error accuracy.

```r
#Train and predict on the whole set with decision trees,
#calculate the accuracy - repeated 20 times
acc_resub = rep(NA,20)
```

```r
for (i in 1:20){
  cancer_model_resub = rpart(class~., data=cancer) #form the model
  class_predict_resub = predict(cancer_model_resub, cancer, type = "class") #make the predictions
  tab_resub <-table(class_predict_resub, cancer$class) #table of true end predicted classes
  acc_resub[i] = sum(diag(tab_resub))/sum(tab_resub) #itteration accuracy
}
cancer_acc_resub_mean <- (mean(acc_resub)) #mean accuracy
cancer_acc_resub_sd <- (sd(acc_resub)) #standard devation of the accuracy
```

Hold-out-10% Again, similar to the wine case, we will split the data for train (90%) and test (10%) at random, 20 times and take the average.

```r
#Train and predict on the 90% of the set with decision trees,
#calculate the accuracy - repeated 20 times
acc_HO = rep(NA,20)
for (i in 1:20){
  #create partition of 90% of the data based on the class
  train_index_HO <- createDataPartition(cancer$class, p = .9, list = FALSE)
  #split the data on the train and test part
  cancer_train_HO <- cancer[train_index_HO,]
  cancer_test_HO <- cancer[-train_index_HO,]
  cancer_model_HO <- rpart(class~., data = cancer_train_HO)
  class_predict_HO = predict(cancer_model_HO, cancer_test_HO, type="class")
  tab_HO <- table(class_predict_HO, cancer_test_HO$class)
  acc_HO[i] <- sum(diag(tab_HO))/sum(tab_HO)
}
cancer_acc_HO_mean <- mean(acc_HO)
cancer_acc_HO_sd <- sd(acc_HO)
```

10-fold-cross-validation In this case, we will perform the 10-fold-CV, repeated 20 times.

```r
#Preparing the CV - spliting data into blocks
n <- nrow(cancer)
K <- 10 #number of folds
n_block <- n%/%K+1
acc_CV10 <- rep(NA, 20)

for (i in 1:20){
  #creating new blocks
  rand = runif(n)
  rang <- rank(rand)
  block <- (rang-1)%/%n_block
  block<- as.factor(block)
  acc <- numeric(0)
  for (k in 1:K-1){
    #training and prediction
    model_k <- rpart(class~., data = cancer[block!=k,])
    pred_k <- predict(model_k, cancer[block==k,], type = "class")
    tab_k <- table(pred_k, cancer[block==k,]$class)
    acc_k <- sum(diag(tab_k))/sum(tab_k)
    acc <- rbind(acc, acc_k)
  }
  #calculating CV10 one itteration accuracy
  acc_CV10[i] <- mean(acc)
}

cancer_acc_CV10_mean <- (mean(acc_CV10))
cancer_acc_CV10_sd <- (sd(acc_CV10))
```

Leave-one-out-cross-validation Fially, we do the LOOCV again, for the new set of data.

```
pred_k <-rep(NA,n)
acc_LOOCV <- rep(NA,20)
for (i in 1:20){
  for (k in 1:n){
    #train and predict with excluding one by one of row from training and predicting for it
    model_k <- rpart(class~., data = cancer[-k,])
    pred_k[k] <- predict(model_k, cancer[k,], type = "class")
  }
  tab_k <- table(pred_k, cancer$class)
  acc_LOOCV[i] <- sum(diag(tab_k))/sum(tab_k)
}
cancer_acc_LOOCV_mean <- mean(acc_LOOCV)
cancer_acc_LOOCV_sd <- sd(acc_LOOCV)
```

Comparison of the evaluation methods Again, here we have very optimistic results for the resubstitution error, while LOOCV again has the lowest accuracy of all methods. In both cases we can notice Resubstitution and LOOCV have 0 standard devation, which is expected, since in both methods the whole set was used both times, and there is no randomness. In other two methods, train and test parts are chosen randomly every times, so results obtained are every time different.

```
cancer_means <- cbind(cancer_acc_resub_mean, cancer_acc_HO_mean, cancer_acc_CV10_mean
                      , cancer_acc_LOOCV_mean)
cancer_sds <- cbind(cancer_acc_resub_sd, cancer_acc_HO_sd
                    , cancer_acc_CV10_sd, cancer_acc_LOOCV_sd)

cancer_stats <-rbind(cancer_means, cancer_sds)
colnames(cancer_stats) <- c("Resub", "Hold_Out_10", "10_fold_CV", "LOOCV")
row.names(cancer_stats) <- c("Mean", "Std_dev")
cancer_stats
```

```
##            Resub Hold_Out_10 10_fold_CV     LOOCV
## Mean    0.965616  0.93260870 0.93586975 0.9111748
## Std_dev 0.000000  0.03093197 0.00484623 0.0000000
```