KIT306/606 Tutorial 5

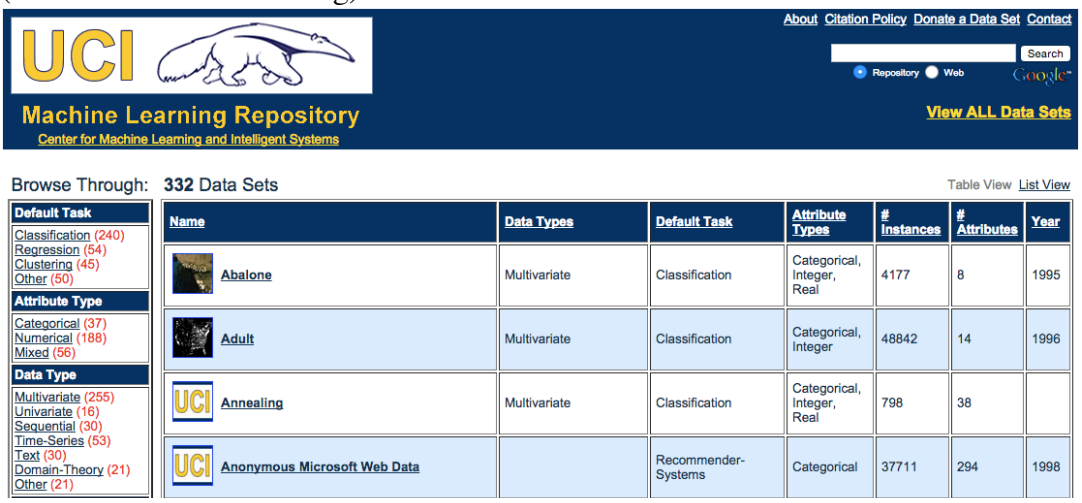
Student ID	Name

The following tutorial work should be completed by tutorial 6 (week7).

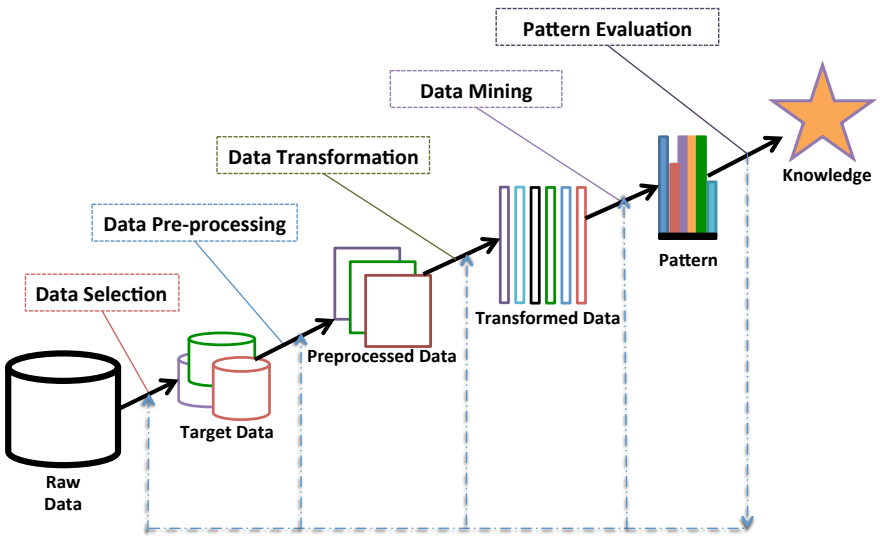
Datasets

Last week, we learned how to collect the web data by scraping the webpage. In this week, we will learn where to find the well-structured data and how to process it.

In UCI machine learning repository (http://archive.ics.uci.edu/ml/datasets/Wine), you can find various types of dataset that is donated for the machine learning/ data mining education or research. You can find and use different types of data based on the task (classification or clustering)



For the first part, we will learn how to collect, pre-process, transform and mine the data by going through each process in the following image.



Data collection and selection

The data we will use for this first part is Wine data, which is the result of chemical analysis to determine the origin of wines. (http://archive.ics.uci.edu/ml/datasets/Wine)

Wine Data Set

Download: Data Folder, Data Set Description

Abstract: Using chemical analysis determine the origin of wines

Data Set Characteristics:	Multivariate	**Number of Instances:**	178	**Area:**	Physical
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	13	**Date Donated**	1991-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	442655

You can find two different downloadable links: data folder and data set description
Data folder will show you the file structures:

Parent Directory		-
Index	03-Dec-1996 04:07	105
wine.data	30-Oct-1995 12:17	11K
wine.names	21-Sep-1998 10:40	3.0K

The link wine.data will show you the following dataset.

```
1,14.23,1.71,2.43,15.6,127,2.8,3.06,.28,2.29,5.64,1.04,3.92,1065
1,13.2,1.78,2.14,11.2,100,2.65,2.76,.26,1.28,4.38,1.05,3.4,1050
1,13.16,2.36,2.67,18.6,101,2.8,3.24,.3,2.81,5.68,1.03,3.17,1185
1,14.37,1.95,2.5,16.8,113,3.85,3.49,.24,2.18,7.8,.86,3.45,1480
1,13.24,2.59,2.87,21,118,2.8,2.69,.39,1.82,4.32,1.04,2.93,735
1,14.2,1.76,2.45,15.2,112,3.27,3.39,.34,1.97,6.75,1.05,2.85,1450
1,14.39,1.87,2.45,14.6,96,2.5,2.52,.3,1.98,5.25,1.02,3.58,1290
1,14.06,2.15,2.61,17.6,121,2.6,2.51,.31,1.25,5.05,1.06,3.58,1295
```

If you click the link wine.names, you can find the detailed information of each attribute.
1) Alcohol, 2) Malic acid, 3) Ash, 4) Alcalinity of ash, 5) Magnesium, 6) Total phenols, 7) Flavanoids, 8) Nonflavanoid phenols, 9) Proanthocyanins, 10) Color intensity, 11) Hue, 12) OD280/OD315 of diluted wines, 13) Proline

You can find the values of each row are related to the attribute. For your information, the first row of above data can be described as the following.

Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Proline
t1	14.23	1.71	2.43	15.6	127	2.8	3.06	0.28	2.29	5.64	1.04	3.92	1065

Let's download this file and use this dataset for data mining.
If you directly download the row in the website, it is difficult to determine each column.

```
> wine<-read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data")
> wine
                                                                V1
1   1,14.23,1.71,2.43,15.6,127,2.8,3.06,.28,2.29,5.64,1.04,3.92,1065
2    1,13.2,1.78,2.14,11.2,100,2.65,2.76,.26,1.28,4.38,1.05,3.4,1050
3    1,13.16,2.36,2.67,18.6,101,2.8,3.24,.3,2.81,5.68,1.03,3.17,1185
4     1,14.37,1.95,2.5,16.8,113,3.85,3.49,.24,2.18,7.8,.86,3.45,1480
```

Then, we can separate the columns by using ‘ (comma) separation as follows:

```
> wine<-read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data",sep=",")
> wine
   V1    V2   V3   V4   V5  V6   V7   V8   V9  V10       V11   V12  V13  V14
1   1 14.23 1.71 2.43 15.6 127 2.80 3.06 0.28 2.29  5.640000 1.040 3.92 1065
2   1 13.20 1.78 2.14 11.2 100 2.65 2.76 0.26 1.28  4.380000 1.050 3.40 1050
3   1 13.16 2.36 2.67 18.6 101 2.80 3.24 0.30 2.81  5.680000 1.030 3.17 1185
4   1 14.37 1.95 2.50 16.8 113 3.85 3.49 0.24 2.18  7.800000 0.860 3.45 1480
5   1 13.24 2.59 2.87 21.0 118 2.80 2.69 0.39 1.82  4.320000 1.040 2.93  735
6   1 14.20 1.76 2.45 15.2 112 3.27 3.39 0.34 1.97  6.750000 1.050 2.85 1450
7   1 14.39 1.87 2.45 14.6  96 2.50 2.52 0.30 1.98  5.250000 1.020 3.58 1290
```

It is much better now but still non-sense.
Why don’t we put the column name for this dataset based on the data set description?

```
> wine<-read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data",sep=",",
+ col.names=c("Type","Alcohol","Malic","Ash","Alcalinity","Magnesium","Phenols","Flavanoids","Nonflavanoids","Proanthocyanins","Color","Hue","Dilution","Proline"))
>
> wine
   Type Alcohol Malic  Ash Alcalinity Magnesium Phenols Flavanoids Nonflavanoids Proanthocyanins     Color   Hue Dilution Proline
1     1   14.23  1.71 2.43       15.6       127    2.80       3.06          0.28            2.29  5.640000 1.040     3.92    1065
2     1   13.20  1.78 2.14       11.2       100    2.65       2.76          0.26            1.28  4.380000 1.050     3.40    1050
3     1   13.16  2.36 2.67       18.6       101    2.80       3.24          0.30            2.81  5.680000 1.030     3.17    1185
4     1   14.37  1.95 2.50       16.8       113    3.85       3.49          0.24            2.18  7.800000 0.860     3.45    1480
```

Now, we can have a look at the wine with recognizable column names.
The statistical details can be found with the following commands.

```
> nrow(wine)
[1] 178
> ncol(wine)
[1] 14
> summary(wine)
      Type          Alcohol          Malic              Ash            Alcalinity       Magnesium          Phenols          Flavanoids
 Min.   :1.000000   Min.   :11.03000   Min.   :0.740000   Min.   :1.360000   Min.   :10.60000   Min.   : 70.00000   Min.   :0.980000   Min.   :0.34000
 1st Qu.:1.000000   1st Qu.:12.36250   1st Qu.:1.602500   1st Qu.:2.210000   1st Qu.:17.20000   1st Qu.: 88.00000   1st Qu.:1.742500   1st Qu.:1.20500
 Median :2.000000   Median :13.05000   Median :1.865000   Median :2.360000   Median :19.50000   Median : 98.00000   Median :2.355000   Median :2.13500
 Mean   :1.938202   Mean   :13.00062   Mean   :2.336348   Mean   :2.366517   Mean   :19.49494   Mean   : 99.74157   Mean   :2.295112   Mean   :2.02927
 3rd Qu.:3.000000   3rd Qu.:13.67750   3rd Qu.:3.082500   3rd Qu.:2.557500   3rd Qu.:21.50000   3rd Qu.:107.00000   3rd Qu.:2.800000   3rd Qu.:2.87500
 Max.   :3.000000   Max.   :14.83000   Max.   :5.800000   Max.   :3.230000   Max.   :30.00000   Max.   :162.00000   Max.   :3.880000   Max.   :5.08000
 Nonflavanoids    Proanthocyanins       Color              Hue              Dilution          Proline
 Min.   :0.1300000   Min.   :0.410000   Min.   : 1.28000   Min.   :0.4800000   Min.   :1.270000   Min.   : 278.0000
 1st Qu.:0.2700000   1st Qu.:1.250000   1st Qu.: 3.22000   1st Qu.:0.7825000   1st Qu.:1.937500   1st Qu.: 500.5000
 Median :0.3400000   Median :1.555000   Median : 4.69000   Median :0.9650000   Median :2.780000   Median : 673.5000
 Mean   :0.3618539   Mean   :1.590899   Mean   : 5.05809   Mean   :0.9574494   Mean   :2.611685   Mean   : 746.8933
 3rd Qu.:0.4375000   3rd Qu.:1.950000   3rd Qu.: 6.20000   3rd Qu.:1.1200000   3rd Qu.:3.170000   3rd Qu.: 985.0000
 Max.   :0.6600000   Max.   :3.580000   Max.   :13.00000   Max.   :1.7100000   Max.   :4.000000   Max.   :1680.0000
```

The current class attribute (Type) is not a categorical data but it would be better to convert into the categorical in order to find the accurate pattern.

```
> wine$Type=factor(wine$Type)
> summary(wine)
 Type      Alcohol          Malic              Ash            Alcalinity       Magnesium          Phenols          Flavanoids       Nonflavanoids
 1:59   Min.   :11.03000   Min.   :0.740000   Min.   :1.360000   Min.   :10.60000   Min.   : 70.00000   Min.   :0.980000   Min.   :0.34000   Min.   :0.1300000
 2:71   1st Qu.:12.36250   1st Qu.:1.602500   1st Qu.:2.210000   1st Qu.:17.20000   1st Qu.: 88.00000   1st Qu.:1.742500   1st Qu.:1.20500   1st Qu.:0.2700000
 3:48   Median :13.05000   Median :1.865000   Median :2.360000   Median :19.50000   Median : 98.00000   Median :2.355000   Median :2.13500   Median :0.3400000
        Mean   :13.00062   Mean   :2.336348   Mean   :2.366517   Mean   :19.49494   Mean   : 99.74157   Mean   :2.295112   Mean   :2.02927   Mean   :0.3618539
        3rd Qu.:13.67750   3rd Qu.:3.082500   3rd Qu.:2.557500   3rd Qu.:21.50000   3rd Qu.:107.00000   3rd Qu.:2.800000   3rd Qu.:2.87500   3rd Qu.:0.4375000
        Max.   :14.83000   Max.   :5.800000   Max.   :3.230000   Max.   :30.00000   Max.   :162.00000   Max.   :3.880000   Max.   :5.08000   Max.   :0.6600000
 Proanthocyanins       Color              Hue              Dilution          Proline
 Min.   :0.410000   Min.   : 1.28000   Min.   :0.4800000   Min.   :1.270000   Min.   : 278.0000
 1st Qu.:1.250000   1st Qu.: 3.22000   1st Qu.:0.7825000   1st Qu.:1.937500   1st Qu.: 500.5000
 Median :1.555000   Median : 4.69000   Median :0.9650000   Median :2.780000   Median : 673.5000
 Mean   :1.590899   Mean   : 5.05809   Mean   :0.9574494   Mean   :2.611685   Mean   : 746.8933
 3rd Qu.:1.950000   3rd Qu.: 6.20000   3rd Qu.:1.1200000   3rd Qu.:3.170000   3rd Qu.: 985.0000
 Max.   :3.580000   Max.   :13.00000   Max.   :1.7100000   Max.   :4.000000   Max.   :1680.0000
```

The function factor is used to encode a vector as a factor (the terms ‘category’ and ‘enumerated type’ are also used for factors). If argument ordered is TRUE, the factor levels are assumed to be ordered. For compatibility with S there is also a function ordered.

Can you see the difference now? ☺

Data pre-processing and transformation

As you can read the above summary information, the range of numeric variables is so Broad. In order to get more accurate data mining result, it would be better to change those scale to within a specific range (0 to 1).

```
> wine.scale <- cbind(wine[1], scale(wine[-1]))
> summary(wine.scale)
 Type      Alcohol              Malic                Ash                Alcalinity          Magnesium           Phenols             Flavanoids
 1:59   Min.   :-2.42738798   Min.   :-1.4289521   Min.   :-3.66881295   Min.   :-2.663504709   Min.   :-2.0823811   Min.   :-2.10131846   Min.   :-1.6911999
 2:71   1st Qu.:-0.78602749   1st Qu.:-0.6568956   1st Qu.:-0.57051311   1st Qu.:-0.687198682   1st Qu.:-0.8220960   1st Qu.:-0.88297744   1st Qu.:-0.8252115
 3:48   Median : 0.06082829   Median :-0.4219218   Median :-0.02375432   Median : 0.001514024   Median :-0.1219377   Median : 0.09568993   Median : 0.1058511
        Mean   : 0.00000000   Mean   : 0.0000000   Mean   : 0.00000000   Mean   : 0.000000000   Mean   : 0.0000000   Mean   : 0.00000000   Mean   : 0.0000000
        3rd Qu.: 0.83377666   3rd Qu.: 0.6679088   3rd Qu.: 0.69614477   3rd Qu.: 0.600394638   3rd Qu.: 0.5082048   3rd Qu.: 0.80672173   3rd Qu.: 0.8466967
        Max.   : 2.25341491   Max.   : 3.1004465   Max.   : 3.14744670   Max.   : 3.145637248   Max.   : 4.3590757   Max.   : 2.53237195   Max.   : 3.0542162
 Nonflavanoids        Proanthocyanins          Color                Hue                Dilution            Proline
 Min.   :-1.8629788   Min.   :-2.06321410   Min.   :-1.6296911   Min.   :-2.08884004   Min.   :-1.8897232   Min.   :-1.4889874
 1st Qu.:-0.7380592   1st Qu.:-0.59560339   1st Qu.:-0.7928659   1st Qu.:-0.76540333   1st Qu.:-0.9495697   1st Qu.:-0.7824306
 Median :-0.1755994   Median :-0.06272092   Median :-0.1587767   Median : 0.03303369   Median : 0.2370660   Median :-0.2330629
 Mean   : 0.0000000   Mean   : 0.00000000   Mean   : 0.0000000   Mean   : 0.00000000   Mean   : 0.0000000   Mean   : 0.0000000
 3rd Qu.: 0.6078267   3rd Qu.: 0.62740554   3rd Qu.: 0.4925666   3rd Qu.: 0.71115828   3rd Qu.: 0.7863692   3rd Qu.: 0.7561165
 Max.   : 2.3956454   Max.   : 3.47526919   Max.   : 3.4257682   Max.   : 3.29240673   Max.   : 1.9553990   Max.   : 2.9631140
```

The scale function is generic function whose default method centers and/or scales the columns of a numeric matrix.

```
> apply(wine.scale[-1], 2, sd)
       Alcohol          Malic            Ash      Alcalinity       Magnesium         Phenols      Flavanoids   Nonflavanoids Proanthocyanins
             1              1              1               1               1               1               1               1               1
         Color            Hue       Dilution         Proline
             1              1              1               1
```

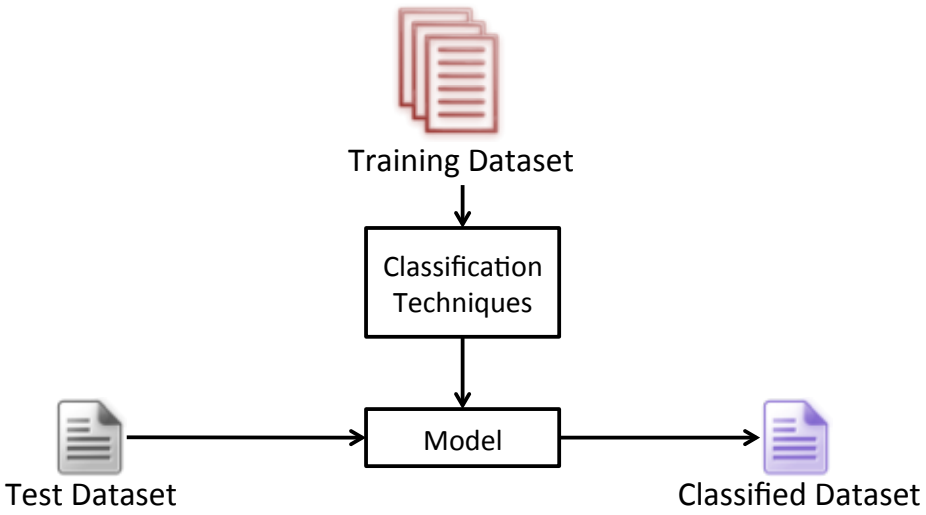
The apply function returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

So, now we got the all required and processed dataset. In data mining process, it is crucial to have two types of dataset: training dataset and testing dataset.

Training dataset and testing dataset

In the data mining process, we should think about the following idea:

- **We have a training dataset**: Given a collection of historical records
- Then, apply **classification** techniques (see lecture 5) in order to find a **model** (unique pattern) for <u>class attribute</u> as a function of the <u>values of other attributes</u>
- A **test set** is used to determine the accuracy of the model.
  - Usually, the **given data set is divided into training and test sets**, with <u>**training set** used to build the model</u> and <u>**test set** used to validate it</u>



So, from the given data, let's divide into training and test datasets. In this example, we will divide into 7:3 (7- training dataset and 3- testing dataset). As mentioned above, the class attribute (Type) has three different value,

```
> # Partitioning the data into training and test data
> data.size <- nrow(wine.scale)
> set.seed(1111)
> samp <- c(sample(1:data.size, data.size * 0.7))
> data.tr <- wine.scale[samp, ]
> data.test <- wine.scale[-samp, ]
```

.seed is an integer vector, containing the random number generator (RNG) **state** for random number generation in **R**. It can be saved and restored, but should not be altered by the user.

```
> summary(data.tr)
 Type      Alcohol              Malic                Ash                 Alcalinity          Magnesium           Phenols
 1:41   Min.   :-1.897718177   Min.   :-1.29468144   Min.   :-3.66881295   Min.   :-2.66350471   Min.   :-2.08238105   Min.   :-2.10131846
 2:50   1st Qu.:-0.816822248   1st Qu.:-0.68151186   1st Qu.:-0.57051311   1st Qu.:-0.74708674   1st Qu.:-0.62955249   1st Qu.:-0.79110255
 3:33   Median : 0.023874586   Median :-0.47115441   Median :-0.02375432   Median :-0.14820613   Median :-0.08692977   Median : 0.16759202
        Mean   : 0.004801705   Mean   :-0.02069784   Mean   :-0.01052628   Mean   :-0.04002124   Mean   : 0.08867446   Mean   : 0.04968805
        3rd Qu.: 0.778346103   3rd Qu.: 0.69923861   3rd Qu.: 0.56856771   3rd Qu.: 0.60039464   3rd Qu.: 0.71825232   3rd Qu.: 0.80672173
        Max.   : 2.253414907   Max.   : 2.96617577   Max.   : 3.14744670   Max.   : 3.14563725   Max.   : 4.35907571   Max.   : 2.53237195
   Flavanoids          Nonflavanoids         Proanthocyanins          Color                Hue                 Dilution              Proline
 Min.   :-1.69119985   Min.   :-1.86297878   Min.   :-2.06321410   Min.   :-1.629691113   Min.   :-2.08884004   Min.   :-1.8897232   Min.   :-1.38101917
 1st Qu.:-0.79267435   1st Qu.:-0.81841060   1st Qu.:-0.59560339   1st Qu.:-0.827374184   1st Qu.:-0.69977837   1st Qu.:-0.9636544   1st Qu.:-0.77925511
 Median : 0.13588543   Median :-0.17559941   Median : 0.05957997   Median :-0.176030871   Median : 0.05490867   Median : 0.2863625   Median :-0.21559748
 Mean   : 0.02123845   Mean   :-0.05118434   Mean   : 0.08015132   Mean   : 0.009590142   Mean   :-0.02602878   Mean   : 0.0155728   Mean   : 0.01083872
 3rd Qu.: 0.88423953   3rd Qu.: 0.54756319   3rd Qu.: 0.66671654   3rd Qu.: 0.519526143   3rd Qu.: 0.59084585   3rd Qu.: 0.8039751   3rd Qu.: 0.72039173
 Max.   : 3.05421616   Max.   : 2.15459116   Max.   : 3.47526919   Max.   : 3.425768243   Max.   : 2.15490741   Max.   : 1.9553990   Max.   : 2.96311399
```

```
> summary(data.test)
 Type      Alcohol              Malic                Ash                 Alcalinity          Magnesium           Phenols
 1:18   Min.   :-2.42738798   Min.   :-1.42895215   Min.   :-2.42949302   Min.   :-2.48384052   Min.   :-1.5222544   Min.   :-1.6219712
 2:21   1st Qu.:-0.76139169   1st Qu.:-0.62109004   1st Qu.:-0.58873840   1st Qu.:-0.59736659   1st Qu.:-0.8220960   1st Qu.:-1.0187925
 3:15   Median : 0.06082829   Median :-0.21156437   Median :-0.18778195   Median : 0.15123418   Median :-0.4370089   Median :-0.1839293
        Mean   :-0.01102614   Mean   : 0.04752837   Mean   : 0.02417146   Mean   : 0.09190064   Mean   :-0.2036228   Mean   :-0.1140985
        3rd Qu.: 0.90768408   3rd Qu.: 0.61196265   3rd Qu.: 0.92396093   3rd Qu.: 0.60039464   3rd Qu.: 0.1581256   3rd Qu.: 0.7468033
        Max.   : 1.69910930   Max.   : 3.10044648   Max.   : 2.01747852   Max.   : 2.69647679   Max.   : 2.3986323   Max.   : 1.6056339
   Flavanoids          Nonflavanoids         Proanthocyanins          Color                Hue                 Dilution              Proline
 Min.   :-1.5610513142   Min.   :-1.7826274   Min.   :-1.6613683   Min.   :-1.36225214   Min.   :-1.82634019   Min.   :-1.84746912   Min.   :-1.48898739
 1st Qu.:-0.9828914517   1st Qu.:-0.7380592   1st Qu.:-0.7834226   1st Qu.:-0.74973061   1st Qu.:-0.88571576   1st Qu.:-0.80872272   1st Qu.:-0.77607957
 Median : 0.0007311716   Median :-0.2157751   Median :-0.2723796   Median :-0.13720908   Median : 0.01115870   Median : 0.18072723   Median :-0.38707642
 Mean   :-0.0487697690   Mean   : 0.1175344   Mean   :-0.1840512   Mean   :-0.02202181   Mean   : 0.05976978   Mean   :-0.03575977   Mean   :-0.02488891
 3rd Qu.: 0.7315652835   3rd Qu.: 1.0497594   3rd Qu.: 0.3609643   3rd Qu.: 0.42247168   3rd Qu.: 0.82053321   3rd Qu.: 0.75819981   3rd Qu.: 0.91092389
 Max.   : 1.6125707883   Max.   : 2.3956454   Max.   : 2.3920327   Max.   : 2.24386051   Max.   : 3.29240673   Max.   : 1.33567238   Max.   : 2.54076771
```

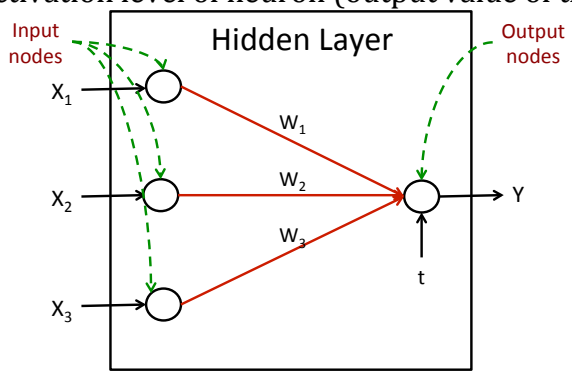
We finished set up training and testing dataset for this data-mining task. Now, it is time to select the type of classification technique in order to find the unique pattern (model). The found unique pattern will help you to predict/ classify the class (Type of wine) based on the other 13 attributes (e.g. Alcohol, Malic, Ash, etc.)

Data mining using a Neural Network Classification Technique

In this example, we will use Neural Network approach as a classification technique in order to find the unique pattern from the training dataset.

Forget about neural network? Please have a look at the lecture 5 slide.

- Artificial neural network (ANN) is a machine learning approach that models human brain and consists of a number of artificial neurons. Each neuron in ANN receives a number of inputs. An activation function is applied to these inputs, which results in activation level of neuron (output value of the neuron).



R provides the neural network package ("nnet"), which allows you to use neural network in creating the model.
First, install the package ("nnet"). Please select Melbourne mirror.

```
> install.packages("nnet")
--- Please select a CRAN mirror for use in this session ---
```

Australia (Canberra)
Australia (Melbourne)
Austria [https]

Then, you will see the following result:

```
trying URL 'http://cran.ms.unimelb.edu.au/bin/macosx/mavericks/contrib/3.2/nnet_7.3-10.tgz'
Content type 'application/x-gzip' length 106222 bytes (103 KB)
==================================================
downloaded 103 KB


The downloaded binary packages are in
	/var/folders/3l/pnv29h0d5bn39t71b1j19blh0000gn/T//RtmpNi6RtP/downloaded_packages
```

Now, it is ready to use neural network. Let's start with the command: library("nnet")
The training data was previously defined **data.tr**, and you can use **nnet** function in order to apply neural network. The class attribute is **Type (which has three categories)** , and other variables in data.tr are other input attributes. The number of hidden layers is 2 (size=2).

- decay : parameter for weight decay. Default 0.
- maxit : maximum number of iterations. Default 100.

Other arguments can be found using help(nnet)

```
> library(nnet)
> model.nnet <- nnet(Type ~ ., data = data.tr, size = 2, decay = 5e-04, maxit = 200)
# weights:  37
initial  value 164.152084
iter  10 value 7.066118
iter  20 value 1.286566
iter  30 value 0.542334
iter  40 value 0.444873
iter  50 value 0.365821
iter  60 value 0.335803
iter  70 value 0.311584
iter  80 value 0.298015
iter  90 value 0.288199
iter 100 value 0.281759
iter 110 value 0.276935
iter 120 value 0.270048
iter 130 value 0.261452
iter 140 value 0.258495
iter 150 value 0.257073
iter 160 value 0.256035
iter 170 value 0.255792
iter 180 value 0.255743
iter 190 value 0.255725
iter 200 value 0.255714
final  value 0.255714
stopped after 200 iterations
```

You can use summary function in order to have a look at the learned neural network model from the training dataset.
The result shows the number of output, hidden, input nodes and weights, as well as the weights from input to hidden nodes and from hidden to output nodes.

```
> summary(model.nnet)
a 13-2-3 network with 37 weights
options were - softmax modelling  decay=0.0005
  b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1
  -1.67    1.72    0.45    1.33   -2.00    0.06    0.15    1.37
 i8->h1  i9->h1 i10->h1 i11->h1 i12->h1 i13->h1
  -0.15   -0.21   -0.26   -0.01    1.38    2.47
  b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2
   2.13    1.87    1.89    1.59   -1.03    0.27   -0.28   -0.78
 i8->h2  i9->h2 i10->h2 i11->h2 i12->h2 i13->h2
  -1.24   -0.48    3.58   -0.83   -0.24    2.39
 b->o1 h1->o1 h2->o1
 -3.79   9.57   1.02
 b->o2 h1->o2 h2->o2
  6.22  -2.01  -9.19
 b->o3 h1->o3 h2->o3
 -2.42  -7.56   8.17
```

Pattern (model) Evaluation

It is not the end yet. It is very important to evaluate your extracted pattern (learned model) whether it predicts/classifies accurately. The testing data we processed above will be used in this part. We will use predict function, which is a generic function for predictions from the results of various model fitting functions. The function invokes particular *methods*, which depend on the class of the first argument.

In the predict function, we will evaluate our model (**model.nnet**) by using testing dataset (**data.test**). The type of this prediction is based on categorical class so it set up as "class". The result of function Predict will give the predicted wine type based on the 13 attributes value in the testing data.

```
> predicted <- predict(model.nnet, data.test, type = "class")
> predicted
 [1] "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1"
[17] "1" "1" "2" "2" "2" "2" "2" "2" "2" "2" "2" "2" "2" "2" "2" "2"
[33] "2" "2" "2" "2" "2" "2" "2" "3" "2" "3" "3" "3" "3" "3" "3" "3"
[49] "3" "3" "3" "3" "3" "3"
```

In order to visualise the confusion matrix, we draw the following comparison table, which can compare the actual and predicted wine type.

```
> actual <- data.test$Type
> model.confusion.matrix <- table(actual, predicted)
> model.confusion.matrix
      predicted
actual  1  2  3
     1 18  0  0
     2  0 21  0
     3  0  1 14
```

Except 1 case, all prediction results are correct! ☺

You can find show the error rate using prop.table function. Prop.table function allows to express Table Entries as Fraction of Marginal Table

```
> confusion.matrix.rate = prop.table(model.confusion.matrix) * 100
> round(confusion.matrix.rate, digit = 2)
      predicted
actual     1     2     3
     1 33.33  0.00  0.00
     2  0.00 38.89  0.00
     3  0.00  1.85 25.93
```

The total error rate can be calculate as follows:
Total confusion matrix – confusion matrix in (1,1), (2,2) and (3,3)

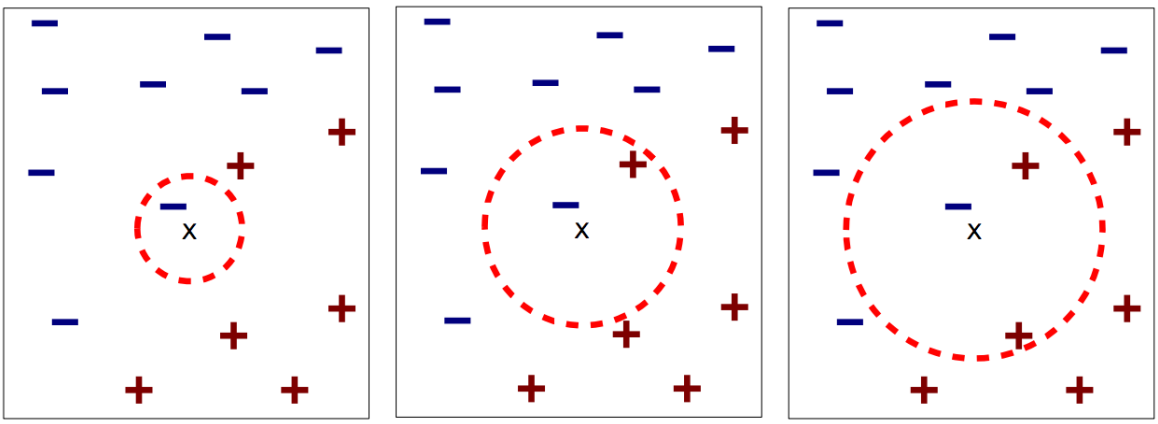
```
> diag.index <- cbind(1:3, 1:3)
>
> error.overall = sum(confusion.matrix.rate) -
sum(confusion.matrix.rate[diag.index])
> paste("Error Rate =", round(error.overall, digit = 2), "%")
```

Other Data mining task using a Nearest Neighbour Approach

Nearest Neighbour approach is one of the laziest classification approach. To classify the new example (testing dataset), the process should be as follows:

- Calculate the distance between X and all examples in the training set
- Select K-nearest examples to X in the training set
- Assign X to the most common class among its K-nearest neighbors

- K-nearest neighbors of a record x are data points that have the k smallest distance to x.



1-nearest neighbour 2-nearest neighbour 3-nearest neighbour

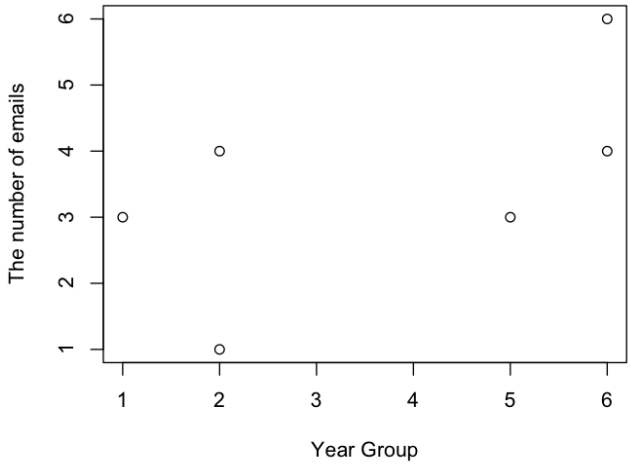
Here is the simple data mining task using K-nearest neighbour. First, assume that there is a CarenLover ☺ and CarenHater ☹
X is year group (1,2,3 is bachelor students, 4,5,6 is master/honours students), and Y is the number of emails that the student sent to Caren.

```
> # Class Caren Lover students
> P1=c(1,3)
> P2=c(2,4)
> P3=c(6,6)
>
> # Class Caren Hater students
> N1=c(2,1)
> N2=c(5,3)
> N3=c(6,4)
```

Training dataset can be built with combining those students' rows. Rbind function allows to Combine R Objects by Rows or Columns.

```
> # Build the classification matrix
> train=rbind(P1,P2,P3,N1,N2,N3)
```

You can plot the training dataset as follows:



Then, the class attribute should be set up for training dataset.

```
> # Class labels vector (attached to each class instance)
> class=factor(c(rep("Lover",3),rep("Hater",3)))
```

Then set up the test data to predict or classify whether the student will love or hate Caren. The test student is first year and sends 4 mails to Caren.

```
> # Testing data to be classified
> test=c(1, 4)
```

In order to use knn (k-nearest neighbour approach), we should load the class package that holds the knn() function.

```
> library(class)
```

Now, call knn() function and get its summary.

```
> summary(knn(train, test, class, k = 3))
Hater Lover
    0     1
```

Try different student data to test it and see what happened

```
> # Testing data to be classified
> test=c(6, 4)
> summary(knn(train, test, class, k = 3))
Hater Lover
    1     0
```

Tutorial Question

1. Data Collection and Processing

i. Make a variable **simpsons**, which collects the html source code of the following web page (http://www.imdb.com/title/tt0096697/epdate)
NOTE: you should call library("rvest") in order to collect the html page.

ii. Scrap the following table and, save it into the variable **rating_table**. (Tips: use %>%html_table(header=TRUE))

table 830px × 13253px

#	Episode	User Rating	User Votes	
1.1	Simpsons Roasting on an Open Fire	7.7	3,548	7.7/10
1.2	Bart the Genius	7.5	1,734	7.5/10
1.3	Homer's Odyssey	7.4	1,458	7.4/10
1.4	There's No Disgrace Like Home	7.5	1,496	7.5/10
1.5	Bart the General	8.2	1,780	8.2/10
1.6	Moaning Lisa	7.5	1,427	7.5/10
1.7	The Call of the Simpsons	7.5	1,461	7.5/10
1.8	The Telltale Head	7.5	1,378	7.5/10

iii. From rating_table, extract the following columns **Episode**, **User Rating**, and **User Votes**, and store it as a table format (using **data.frame** function –learned in tutorial 2) with changing the column names into **Title**, **Ratings**, and **Votes**. Then, save it in the variable **selected**.

iv. When you call variable **selected**, you should see the following result (574 rows).

```
> selected
                                                  Title Ratings Votes
1                                Simpsons Roasting on an Open Fire     7.7 3,548
2                                                Bart the Genius     7.5 1,734
3                                                Homer's Odyssey     7.4 1,458
4                                    There's No Disgrace Like Home     7.5 1,496
5                                                Bart the General     8.2 1,780
6                                                    Moaning Lisa     7.5 1,427
7                                        The Call of the Simpsons     7.5 1,461
8                                              The Telltale Head     7.5 1,378
9                                           Life on the Fast Lane     7.4 1,348
```

v. The title of Episode is too various to make it as attributes so it is necessary to make generalised. You should change the value of Title by using ifelse and grepl function. If the title of content contains "Bart" or "Homer", it should replace the value of the **selected$Title** column into "Bart" or "Homer". If not, the value of the **selected$Title** column should "Others" (Tips: You can find which title contains Bart or Homer using grep("Bart",selected$Title) or grep("Homer",selected$Title)
grepl returns TRUE if a string contains the pattern, otherwise FALSE; if the parameter is a string vector, returns a logical vector (match or not for each element of the vector).

vi. Set up the **selected$Title** into the categorical variable using **factor** function

vii. Divide the data selected into training and testing data (7:3). The training and testing data should be randomly selected using **set.seed(1111).** The result should be something like this **(NOTE: the data is randomly selected and divided the value may be a little bit different)**:

```
> summary(trainset)
    Title        Ratings          Votes
 Bart  : 31   Min.   :5.20   526     :  5
 Homer : 36   1st Qu.:7.10   513     :  4
 Others:334   Median :7.30   417     :  3
              Mean   :7.29   493     :  3
              3rd Qu.:7.50   528     :  3
              Max.   :9.20   536     :  3
                             (Other):380
>
> summary(testset)
    Title        Ratings          Votes
 Bart  : 11   Min.   :5.800   1,033  :  2
 Homer : 19   1st Qu.:7.100   1,056  :  2
 Others:143   Median :7.300   1,077  :  2
              Mean   :7.362   1,163  :  2
              3rd Qu.:7.500   378    :  2
              Max.   :9.300   491    :  2
                              (Other):161
```