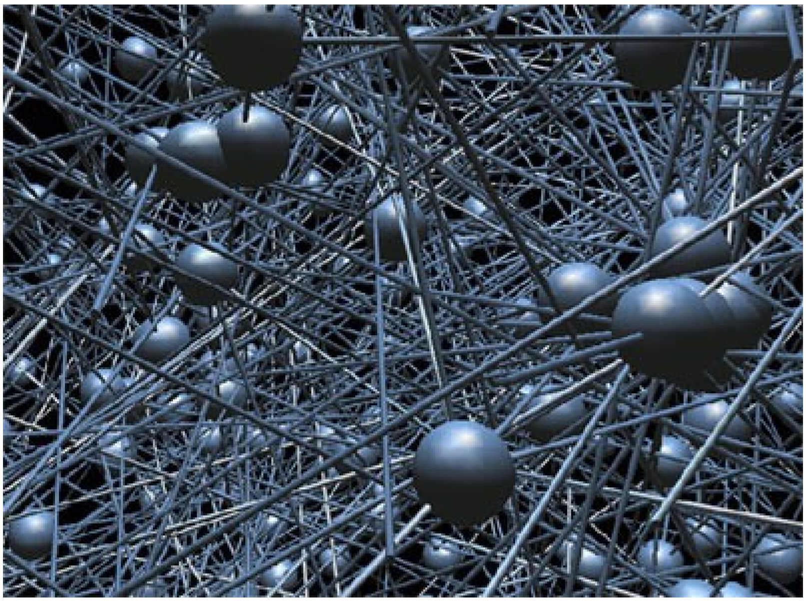
KB – Neural Data Mining with Python sources

Introduction

The aim of this work is to present and describe in detail the algorithms to extract the knowledge hidden inside data using Python language, which allows us to read and easily understand the nature and the characteristics of the rules of the computing utilized, as opposed to what happens in commercial applications, which are available only in the form of running codes, which remain impossible to modify.

The algorithms of computing contained within the work, are minutely described, documented and available in the Python source format, and serve to extract the hidden knowledge within the data whether they are textual or numerical kinds. There are also various examples of usage, underlining the characteristics, method of execution and providing comments on the obtained results.



The KB application consists of three programs of computing:

- KB_CAT: for the extraction of knowledge from the data and the cataloging of records in homogeneous groups within them
- KB_STA: for the statistical analysis of the homogeneity of the groups between them and in the groups within them in order to identify the groups most significant and the most important variables that characterize each group
- KB_CLA: for the almost instantaneous classification of new records in catalogued groups before found by the program KB_CAT

The programs have been written in Python language using the most easily understood commands, instructions and functions, and those most similar to those of other languages (e.g. C, C++, PHP, Java, Ruby); however, the programs are full of comments and explanations.
The KB application to acquire hidden knowledge in data is the result of almost five years of study, programming and testing, also of other languages (Clipper, Fortran,

Ruby, C e C++).
The cost of the book is low considering the importance of the included algorithms of computing and the hard work in its programming and in the subsequent repeated and thorough testing of the input data of files containing thousands of records; the files used arrived from various sectors of interest (medicine, pharmacology, food industry, political polls, sport performances, market researches, etc.).
The author has chosen to use simple forms of communication, avoiding complicated mathematical formulas, observing that the important concepts can be expressed in an easy but not simplistic way.
Albert Einstein said: *Do things in the easiest way possible, but without simplifying.*
The author hopes to give a small contribution to encouraging young people to regain a love for maths, but above all hopes they will regain the desire to run programs on their computers, and therefore avoid using them just to consume their fingers surfing facebook or downloading music and films from Internet.
In his professional experience, firstly in computer fields and then as an industrial manager, he has repeatedly realized programs with mathematical contents, statistics and operations research which have considerably contributed to the economic results and good management of the companies that have seen him as a significant protagonist in important projects.

Business Intelligence and the bad use of statistics

Statistics are often wrong, or rather, the people who use them make mistakes. They make mistakes when they apply statistical aggregation instruments to pieces of information from sources coming from completely different objects or situations.
First of all they cut, then mix and finally put them together. And to finish off they expect to pass judgement on this.
In this way the researcher in political trends break up the opinions of the people interviewed, mixing the single answers, joining them, crossing them and finally passing judgement with certainties that can only be attributed to *virtual people interviewed* that they have created, subjects that do not exist in real life and certainly are not traceable to individual people or to homogeneous groups of people who have been interviewed.
Similarly the *Business Intelligence* makes available the tools of data analysis that are able to cut the data and then reassembling them into multidimensional structures in which the peculiarities of information starting positions were destroyed.
So Business Intelligence mixes companies from different sectors with turnovers not compatible, with very different sizes, belonging to different markets, etc., thereby abusing the will to change from time to time variables for data mining.
Which decisions on subjects (or situations) could be applied to, having destroyed the global informative world of the original subjects (or situations)?
To give an example, if we had a file of mammals where men and primates were included, we could obtain, as a result, that mammals, on average, have three legs.
Where can I find a mammal that has an average of three legs?
To have real statistics we need to conserve, as much as possible, intact the informative property of the starting data of the subject or the situation .
Techniques derived from neural networks use an analysis approach to data which respect the informative properties of the starting data.
In fact they do not ask the user to define the variables to cross, and therefore do not allow to occur absurd crossed values.

Quite simply they require that the maximum number of groups that the algorithm has to create is inserted
The original informative contents are not destroyed, the subject's data are processed in relationship to the data of other subjects (or situations).
Retain all the information attributable to the subject and create the categories of membership of the subjects (or situations) in which the subjects (or situations) will be similar to each other.
Other techniques are able to point out what are the significant variables of aggregation and aggregate values which are important for each group created.
Also indicate what are the variables that are not influential in cataloging.
More sophisticated techniques can process any kind of data set highlighting if there is information in the file or if they contain only numbers or characters not related to each other by internal relations: *the model must follow the data and not vice versa (JB Benzecri).*

Learning by induction and the neural networks

Induction is a very important method of learning for living creatures.
One of the first philosophers to resort to this concept was Aristotle, who attributed the merit of having discovered it to Socrates, who maintained that induction was in fact, *"the process of the particular that leads to the universal" (Top., I, 12, 105 a 11)*.
Still according to Aristotle it is neither the senses through induction nor rationality through deduction that gives a guarantee of truth, but only intellectual intuition: this allows to collect the essence of reality, forming valid and universal principles, from which syllogistic reasoning will draw coherent conclusions with premises.
Learning, life and evolution are linked together.
In fact life is evolution and evolution is learning what is necessary for survival. Learning is the capacity to elaborate information with critical intelligence. Therefore, critical elaboration of information is life. (Roberto Bello).
A simple example can illustrate how one learns by induction.
Let's imagine a person who had never seen containers such as glasses, bottles, jars, cups, vases, boxes, flagons, jugs, chalices, tetra pack and so on.
Without saying anything I will show him real examples of objects that belong to the above mentioned categories.
The person can look at, smell, touch and weigh the objects shown to him.
After having examined a sufficient number of objects the person will easily be able to put the objects into categories containing the objects which on the whole are similar to each other, favouring some characteristics rather than others which are not considered relevant.
When the learning has taken place, I could show another object in the shape of a glass, which is of a different colour, made of a different material and of a different weight, still obtaining the cataloging of the object in the category of glasses.
With the help of induction, the person in training could make two categories of glasses: one with handles (beer mugs) and the other without handles.
Learning has allowed the person to recognize the distinctive aspects of the object to go from the specific to the universal ignoring the non relevant aspects.
The algorithms based on the neural networks, and in particular referring to the map of Kohonen (SOM Self Organizing Map), are based on the principals which have just been illustrated in this example.
Such a model of neural networks demonstrates in an important way the biological

mechanisms of the central nervous system; many studies have demonstrated that precise zones exist on the surface of the cranial cortex, each of which respond to a precise sensory or muscular function.
Each neuron specializes in responding to precise stimulus through a continual interaction with the neighbouring neurons.
We have zones reserved for hearing, sight, muscular activity etc., and the spacial demarcation between the different groups is so clear that we talk of the formation of bubbles of activity.
The neural networks model presented by Kohonen imitates the behaviour described above.
The architecture is quite simple; the network is formed by a rectangular grate, also known as Kohonen's layer, made up of neurons from the output level, each one occupying a precise position and connected to all the entry units.
The weight of the connections between the input and output levels are kept up to date thanks to the process of learning, where the connections between the neurons of the output level have weights which produce excitement among the surrounding neurons and inhibition in distant ones.

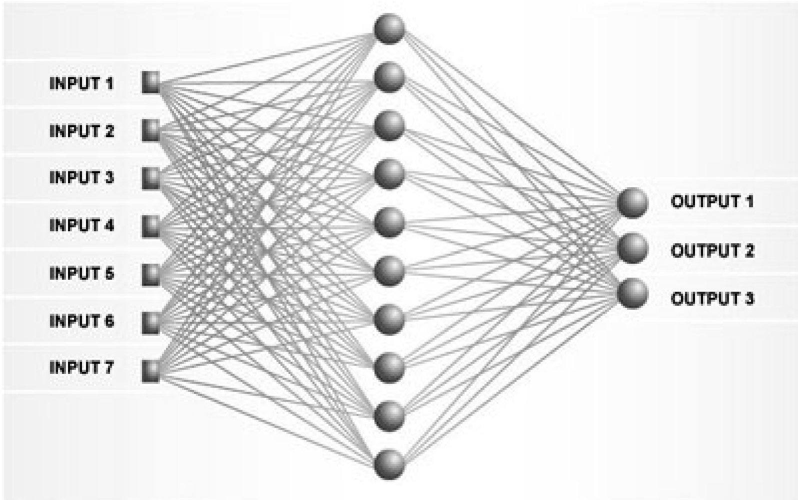
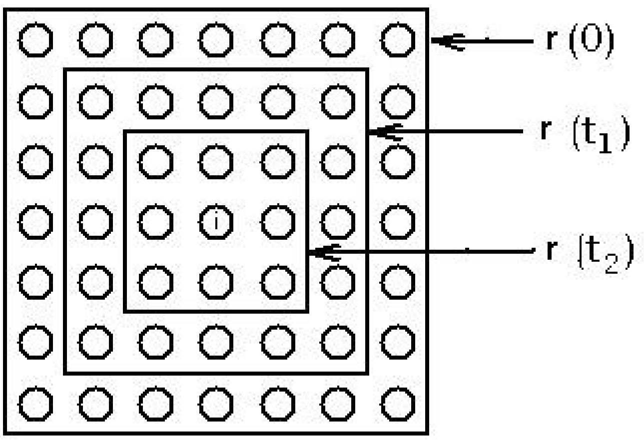


Diagram of a neural network

The *SOM networks* are applied to many practical problems; they are able to discover important properties autonomously in input data and therefore they are especially useful in the process of *Data Mining*, above all for problems of cataloging.
The algorithms of learning of the Kohonen network begin from the start-up phase of the synapse weights, which must have casual values in space (0.0 – 0.99999) and be different for each neuron.
Subsequently the weights are presented to the network as input values and the algorithm allows the network to self-organize and correct the weights after each data input, until a state of equilibrium is reached. Kohonen's network is also known as a competitive network since it is based on the principle of competition between neurons to win and to remain active; only the weight of the active units are updated.
The winning unit i* is that which possesses the potential for major activation; the more a unit is active for a certain pattern of input data, the more the vector of the

synapse weight is similar inside the pattern.
On the basis of this idea it is possible to find the winning unit by calculating the euclidean distance between the input vector and the relevant vector of synapse weight. At this point is selected the neuron i* that corresponds to the minimum distance.
Once the winning neuron has been determined, is carried out an automatic learning of the weight of the neuron itself and of those which are part of its neighbourhood , based on a rule of *hebbian* type.
In particular, a formula of modification of the weights which derives from the original rule of Hebb is used; considering that this would increase the weight to infinity, so is introduced a factor of forgetfulness, pushing the weights toward the input vectors to which the unit responds more.
In this way a relative map of the characteristics of input is created where the neighbouring units respond to precise stimulus of admission thanks to the similarity of the synapse weights.
For this aim it is also necessary to introduce the concept of the function of proximity, that determines the area of size *r* around *i** where the units are active.
The less the dimension of the proximity, the lower the number of the units of the layer of Kohonen whose weights are modified significantly, therefore the higher the capacity of the neurons to differentiate and to acquire details but also to increase the complexity of the system of learning.
According to Kohonen the size of the function of proximity must be varied, initially choosing it to cover all of the units of the layer and gradually reducing it.
In this way you will go from learning the main features up to learning the details of the specialized areas in responding to particular stimuli.



Representation of the gradual reduction of proximity

Once the learning phase has been completed the network is able to supply answers in relation to the new input presented. The property of generalization derives from the fact that even the neurons near to those selected are modified.
The network must therefore self-organize in areas that are composed of a large set of values around the input from which the algorithm learns, this will ensure that if there is an input never seen before, but with similar characteristics, the network will be able to classify properly.

Besides this, compared to supervised algorithms, the self-organized processes of learning (SOM) result to be efficient even if are used incomplete or incorrect input data, a characteristic that makes this neural network particularly suitable to be used in the process of *Data Mining*.
In fact Kohonen algorithms, at the end of the phase of non supervised training, produces a three-dimensional matrix that can be used to classify new records in groups with the most similar characteristics.
While the training phase can require a lot of time to run, that of classifying new records in the groups with the most similarities is almost instantaneous, making this function especially useful for processes with real time reactions (e.g. quality control, productions in a continuous cycle, automation in industrial processes, control systems, monitoring the messages on the Net, etc.).
The algorithms of the neural networks have, as a common aspect, the inability to explain the characteristics of the groups obtained.
It is possible, using the information contained in the training matrix and resorting to other technical statistics, to provide information on the characteristics of every group helping the researcher to deepen the analysis of the results to gather better results of their research.
It is also possible to determine if the overall view of the records used in the training phase has knowledge contents or, on the contrary, it is made up of data which have little connection between them and therefore not suitable for the use of research: in fact it is possible to compute the global index of homogeneity of the groups on the whole (*Knowledge Index*), informing the researcher of the suitability of the output files to achieve the expected goals.

KB

Python for KB

The Python program language is a language that can be freely downloaded from Internet.
Python is compatible with Windows, Linux/Unix, Mac OS X, OS/2, Amiga and Smart-phones /Tablets.
Python is distributed on license Open-Source: its use is free of charge also for commercial products.
The site from where the Python language can be downloaded is *www.python.org/download*, choosing the compatible version for your computer.
Installing Python in Windows involves choosing the extended file *msi* to download from Internet.
To install Python in Linux (and in particular in Linux Ubuntu) use the *Software Manager* of the Linux distribution, which automatically connects to the official site (*repository*) , downloading what is necessary for a safe, complete and automatic installation; Linux distributions usually already contain the Python language pre-installed.
Whatever the operating system for the installation of Python may be, the programs can only be used in command mode option by opening the file containing the Python program (for example: program.py), typing *python program.py*:

- in a DOS window (with *execute*) in Windows
- in a terminal window in Linux

Details

The KB application is a system which extracts knowledge from data based on algorithms of the map of Kohonen revised and modified by the author.
KB can elaborate any table of numeric data and/or text, tables where the first line of the table is destined to the description of the columns / variables and the first column of the table is destined to the codes (arbitrary) of identification of the record / case.
In KB, functions are included with the aim of:

- normalizing the numeric data comparing it to the *standard deviation* or to the maximum value of the variable / column according to the user's choice
- transforming the alphanumeric data into numeric data conveniently returned equidistant between them
- inserting statistical functions able to judge the quality of the results of the cataloging for each group and globally
- writing different output files:
    - records / cases arranged by group code according to the chosen category
    - synthetic statistical information on the characteristics of the different groups also in relation to statistical indexes with reference to entire populations of records / cases
    - the final training matrix having the minimum error

The neural networks have the known defect of being *black boxes* in that they are able to catalog but don't *explain*:

- what the characteristics of each group are
- what the columns/variables are important in each group for the cataloging
- what the most homogeneous groups are on the inside
- if, in its global sense , the input table contains information in relation between the variables or if the table is purely a group of numbers and letters without any inductive value.

Appendixes contain the programs written in Python KB_CAT (for the cataloging), KB_STA (for the analysis of the results) and KB_CLA (for the classification).
They have to be converted into file in text form using *cut and paste*; the programs have to be stored with names:

- kb_cat.py
- kb_sta.py
- kb_cla.py

The name of the programs can also be different from kb_cat, kb_sta, kb_cla, as long as the *extension* is "*.py*" to allow the Python language to recognize the programs and run them.
Some of the test files are also reproduced and the results obtained are shown in the DOS window (Windows) or in the Terminal Window (Linux), results are contained in files in text format.

Collecting and arranging input data

The use of programs based on the algorithms of Kohonen require data to be

prepared and normalized.
To begin with it is important to carefully choose the data to be analysed.
Information from which the user intends to extract knowledge must be contained in tables that have the following characteristics:

- the format must be text (txt, csv)
- the fields must be separated by tabulation (tab)
- the first column is destined to identify each line with the identification code of every record (e.g. Client's code, product name, production lot, etc.)
- the first line must contain the descriptions of the columns separated by the tabulation (*tab*)
- values are contained in the cells from the second column to the last column and from the second line to the last line
- all the values of all the columns and all the lines must be separated by tabulation (*tab*)
- empty fields or those not containing anything cannot exist
- a column which contain numerical data cannot contain data with text

To convert tables into text you can resort to programs *xls* (*Excel*) or *OpenOfficeCalc* (*ods*) which are able to read the input formats and convert them into (*csv*) format, choosing the tabulation field (*tab*) and space (*empty*) to delimit the text.
For the quality of the results, the famous saying *garbage in, garbage out* is always valid; it is fundamental to collect good quality data that allows the research to be described and explained in the most complete way possible.
You also need to decide what size of the data is to be used as an input file (see the following suggestions).
The neural networks give the same weight to all of the variables inserted; if a variable oscillates in an interval (1000 - 10000) and the other in an interval (0 - 1), the variations of the first tend to reduce the importance of the second, even if the latter could be more significant in determining the results of the classification.
To do this transformation techniques exist which make the variables compatible among them, making them fall inside a certain interval (*range*).
The KB_CAT program can apply different techniques of normalization of the numeric values and text data.
Numeric values can be normalized through two methods:

- Normalization with the maximum: the new values of the column are obtained dividing the original values for the maximum value of the column, in this way the new values vary between zero and 1
- Standardization: the new values are obtained subtracting from the original value the mean of the column and dividing the result of the difference for the *standard deviation* of the column.

From the columns containing strings of characters are extracted the value of those containing different strings, they are sorted, counted and then are used to determine the attribution step of a numeric value between 0 and 1.
The KB_CAT program does not foresee the automatic transformation of the date or the time The date must be transformed by the user in pseudo continue numeric variables assigning the value 0 to the most remote date and increasing by a unit every subsequent date, or expressing the 365 days of the year in thousandths, according to the formula: 0,9999* days a year/365.
The year could also be indicated using another variable. The pair of variables

should preferably be expressed as a ratio, through a single value which will offer information which is clearer and more immediate; in this way the derived variables can be calculated starting from the input variables.
Let us imagine that two variables are present: the weight and height of a person.
Considered separately they have little meaning, it would be better to obtain the coefficient of the body mass which is definitely a good synthetic index of obesity (body weight expressed in kilogrammes divided by height in meters squared).
Another important step in preliminary elaboration of data is to try and simplify the problem you want to resolve.
To do this it may be useful to reorganize the space of the input values, space which grows essentially as grows the size of data.
A technique to reduce the number of variables and improve the ability of learning of the neural networks, which is often used, is the *principal component analysis*, that try to identify a sub-space *m* size which is the most significant possible in respect to the input space *n* size.
The *m* final variables are called *principal components* and are linear combinations of *n* initial variables.
Other methods used to reduce the size of the problem to resolve are the elimination of the variables which are strongly linked between them and not useful to achieve the desired result.
In the first case it is important to consider that the connection does not always imply a cause/effect relationship, therefore eliminating some variables must be done with extreme care by the user.
It is very common to reorganize file of input data that needs to be cataloged by examining the results of the first processing runs which often indicate that certain variables/columns are worthless: their elimination in subsequent processing often contribute to improve the cataloging having put an end to the noise of the useless variables/columns.
In the processing of data relating to clinical trials, it was verified that the personal data of gender, nationality, residence, education, etc., not giving in those cases no contribution to cataloging, could be omitted improving the quality of new learning.
A very important aspect to consider is related to the number of records contained in an input file to catalog.
Often better results are obtained with smaller files which are able to generalize better and produce training matrices which are more predictive.
On the contrary, a file containing a large number of records could produce an invalid training of *overfitting* causing a photographic effect which can only classify new records which are almost identical to those used in the phase of cataloging.
As scientists at Cern have already discovered, it's more important to properly analyse the fraction of the data that is important ("of interest") than to process all the data. TomHCAnderson
In statistics we talk about overfitting (excessive adaptation) when a statistics model fits the observed data (the sample) using an excessive number of parameters.
An absurd and wrong model converges perfectly if it is complex enough to adapt to the quantity of data available.
It is impossible to prove at first glance the best number of records to be contained in a file to catalog: too much depends on the number of variables and the informative contents of the variables for all of the records present in the file.
The best suggestion is to carry out distinct runs with the original file and with other

files obtained with a lesser number of records
To obtain a smaller sized file you can extract records from the original file by random choice, you can use the small program KB_RND which is present in appendix 4.

```
###############################################################################
# KB_RND KNOWLEDGE DISCOVERY IN DATA MINING (RANDOM FILE SIZE REDUCE)        #
# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                #
# Language used: PYTHON                                                      #
###############################################################################
InputFile                              : cancer.txt
OutputFile                             : cancer_rnd.txt
Out number of cells (<= 90000)         : 10000
Nun. Records Input 65535
Elapsed time (microseconds)            :   235804
```

Indicate in **InputFile** the file from which you want to extract the smaller sized output file (**OutputFile**).
Indicate in **Out number of cells** the number of cells (lines x columns) of the output file.
Other times it is convenient to remove from the initial input file, the records which clearly contain values contradictory, absurd or missing: in so doing you reduce the size of the file and improve the quality by reducing the noise.

General warnings for using the KB programs

It is important that the files that are in input and output, while processing the three programs kb_cat, kb_sta, kb_cla are not open in other *windows* for reading or writing: if this happens kb_cat, kb_sta, kb_cla would go into error.
Processing the three programs can be interrupted by pressing ctrl and the c keys.

KB_CAT Knowledge Data Mining and cataloging into homogeneous groups

Generality, aims and functions

KB_CAT is the first of the three programs to use and it is the most important.
Its purpose is to analyse any kind of textual file structured in two-dimensional table containing numeric values and/or text data.
KB_CAT:

- controls that the table to process does not contain errors of format
- normalizes the numeric values and the text data
- starts the training phase searching for the minimum error which decreases during the processing until it reaches the minimum value of the *alpha* chosen by the user.

Once the processing has been completed, the program will write the output file containing the results which can also be used by the other two programs KB_STA and KB_CLA.

Source of KB_CAT (see attachment 1)

Test Input file (copy and paste then save with name *vessels.txt*); fields separated by tabulation

Description	Shape	material	height	colour	weight	haft	plug
glass_1	cut_cone	pewter	10	pewter	20	no	no
glass_2	cut_cone	plastic	9	white	4	no	no
glass_3	cut_cone	terracotta	8	grey	20	no	no
beer_jug	cut_cone	porcelain	18	severals	25	no	no
dessert_glass	cut_cone	glass	17	transparent	17	no	no
wine_glass	cut_cone	glass	15	transparent	15	no	no
jug	cylinder	terracotta	25	white	40	yes	no
bottle_1	cylinder_cone	glass	40	green	120	no	cork
bottle_2	cylinder_cone	glass	40	transparent	125	no	cork
bottle_3	cylinder_cone	glass	45	opaque	125	no	plastic
bottle_4	cylinder_cone	glass	35	green	125	no	metal
magnum_bottle	cylinder_cone	glass	50	green	170	no	metal
carboy	ball_cone	glass	80	green	15000	no	cork
ancient_bottle	ball_cone	glass	40	green	150	no	cork
champagne_glass	cut_cone	crystal	17	transparent	17	no	no
cup_1	cut_cone	ceramic	10	white	30	yes	no
milk_cup	cut_cone	terracotta	15	blue	35	yes	no
tea_cup	cut_cone	terracotta	7	white	30	yes	no
cup_2	cut_cone	glass	20	transparent	35	yes	no
coffee_cup	cut_cone	ceramic	6	white	20	yes	no
tetrapack1	parallelepiped	mixed	40	severals	20	no	plastic
tetrapack2	parallelepiped	plastic	40	severals	21	no	plastic
tetrapack3	parallelepiped	millboard	40	severals	22	no	no
cleaning_1	parall_cone	plastic	30	white	50	yes	plastic
cleaning_2	cylinder_cone	plastic	30	blue	60	yes	plastic

Description	Shape	material	height	colour	weight	haft	plug
tuna_can	cylinder	metal	10	severals	10	no	no
tuna_tube	cylinder	plastic	15	severals	7	no	plastic
perfume	parallelepiped	glass	7	transparent	15	no	plastic
cleaning_3	Cone	plastic	100	severals	110	yes	plastic
visage_cream	cylinder	metal	15	white	7	no	no
cd	parallelepiped	plastic	1	transparent	4	no	no
trousse	cylinder	plastic	1	silver	7	no	yes
watering_can	Irregular	plastic	50	green	400	yes	no
umbrella_stand	cylinder	metal	100	grey	3000	no	no
pot_1	cylinder	metal	40	grey	500	two	yes
pot_2	cut_cone	metal	7	grey	200	yes	yes
toothpaste	cylinder	plastic	15	severals	7	no	plastic
pyrex	parallelepiped	glass	10	transparent	300	two	glass
plant_pot	cut_cone	terracotta	30	brown	200	no	no
pasta_case	parallelepiped	glass	35	transparent	150	no	metal

How to run

Being positioned in the file containing kb_cat.py and the input file to process, start KB_CAT typing in the commands window of DOS Windows (or in the Terminal of Linux), the command:

python kb_cat.py

python runs the program (with python language) ***kb_cat.py.***

The program will start subsequently asking

Input File = *vessels.txt*

vessels.txt is the file in format txt containing the table of the records / cases to catalog, shown above.
If you want to give more importance to one particular variable/column, all you have to do is to duplicate the value, one or more times, in additional variables/columns: if you want to make the variable important for three times its original weight, create another two variables/columns calling them for example *shape1* and *shape2* with values which are identical to the original variable.

Number of Groups (3 – 20) = 3

The value 3 is the square root of the maximum number of groups to catalog (in this

case 9); since the training matrix has a cube form base; the maximum number of training groups can only be the square of the value that has been entered.
There are no useful rules for fixing the best number of the parameter Number of Groups: it is advisable to initially try with low values and gradually carry out other processing with higher values if are obtained groups containing too many records, and on the other hand, reduce the value of the parameter Number of Groups if are obtained groups containing few records.
Sometimes though, the researcher is interested in analysing groups with few numbers of records but with rare and singular characteristics: in this case groups containing few records are welcome.

Normalization (Max, Std, None) = m

The value m (M) indicates the request to normalize numerical data dividing them by the maximum value of the column / variable.
The value s (S) indicates the request to normalize numerical data subtracting from each input value the average of the variable / column and dividing the result by the *standard deviation* of the variable / column.
It is not advisable to insert the value *N* (None) above all in the presence of variables which are very different among them with a large difference between the minimum and maximum value (*range*).

Start Value of alpha (from 1.8 to 0.9) = 0.9

KB_CAT, like all algorithms of the neural networks, runs cycles making loops which consider all the input.
In these loops the alpha parameter plays an important role from its initial value (Start Value) to its final value (End Value) also considering the value of the decreasing step.
Occasionally an excessive length of time for the processing can be noted having chosen a large number of groups for a file containing a lot of records and with distant start and end values of alpha and with a very small decreasing step of alpha; usually in these cases you will notice the minimum error remains the same in many loops.
It is advisable to stop the processing, by pressing the two keys ***ctrl*** e ***c***, together and repeat it using more suitable parameter values.

End Value of alpha (from 0.5 to 0.0001) = 0.001

The *alpha* parameter used by the KB_CAT to refine the cataloging of records into different groups: a low *alpha* value involves a longer cycle time of the computing with the possibility of obtain a lower final minimum error but also a hypothetical greater chance of *over fitting (photo effect)*.

Decreasing step of alpha (from 0.1 to 0.001) = 0.001

Choose the value of the step of decreasing alpha to be applied to each loop.

Forced shut down of processing

In the case of wanting to shut down the processing while it is running, you just

need to press the two keys ***ctrl*** and ***c*** at the same time.
Obviously the files that were in writing will not be valid.

KB_CAT produce the following output:

In the window DOS Windows (or the Terminal Linux)

```
###############################################################################
# KB_CAT KNOWLEDGE DISCOVERY IN DATA MINING (CATALOG PROGRAM)                 #
# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                 #
# Language used: PYTHON                                                       #
###############################################################################
InputFile                                : vessels.txt
Number of Groups (3 - 20)                : 3
Normalization(Max, Std, None)            : m
Start value of alpha (1.8 - 0.9)         : 0.9
End value of alpha (0.5 - 0.0001)        : 0.001
Decreasing step of alpha (0.1 - 0.001) : 0.001
Record 40 Columns 7
**** Epoch 15       WITH MIN ERROR      3.616    alpha      0.88650
**** Epoch 39       WITH MIN ERROR      3.612    alpha      0.86490
**** Epoch 41       WITH MIN ERROR      3.608    alpha      0.86310
**** Epoch 44       WITH MIN ERROR      3.460    alpha      0.86040
**** Epoch 46       WITH MIN ERROR      3.456    alpha      0.85860
**** Epoch 48       WITH MIN ERROR      3.451    alpha      0.85680
**** Epoch 50       WITH MIN ERROR      3.447    alpha      0.85500
**** Epoch 52       WITH MIN ERROR      3.443    alpha      0.85320
**** Epoch 54       WITH MIN ERROR      3.439    alpha      0.85140
**** Epoch 56       WITH MIN ERROR      3.435    alpha      0.84960
**** Epoch 58       WITH MIN ERROR      3.431    alpha      0.84780
**** Epoch 60       WITH MIN ERROR      3.426    alpha      0.84600
**** Epoch 62       WITH MIN ERROR      3.422    alpha      0.84420
**** Epoch 64       WITH MIN ERROR      3.418    alpha      0.84240
**** Epoch 66       WITH MIN ERROR      3.414    alpha      0.84060
**** Epoch 68       WITH MIN ERROR      3.410    alpha      0.83880
**** Epoch 70       WITH MIN ERROR      3.371    alpha      0.83700
**** Epoch 72       WITH MIN ERROR      3.366    alpha      0.83520
**** Epoch 74       WITH MIN ERROR      3.362    alpha      0.83340
**** Epoch 76       WITH MIN ERROR      3.358    alpha      0.83160
**** Epoch 78       WITH MIN ERROR      3.353    alpha      0.82980
**** Epoch 80       WITH MIN ERROR      3.349    alpha      0.82800
**** Epoch 82       WITH MIN ERROR      3.345    alpha      0.82620
**** Epoch 84       WITH MIN ERROR      3.341    alpha      0.82440
**** Epoch 86       WITH MIN ERROR      3.336    alpha      0.82260
```

```
**** Epoch 88       WITH MIN ERROR       3.332   alpha      0.82080
**** Epoch 90       WITH MIN ERROR       3.328   alpha      0.81900
**** Epoch 92       WITH MIN ERROR       3.324   alpha      0.81720
**** Epoch 94       WITH MIN ERROR       3.320   alpha      0.81540
**** Epoch 96       WITH MIN ERROR       3.316   alpha      0.81360
**** Epoch 98       WITH MIN ERROR       3.311   alpha      0.81180
**** Epoch 102       WITH MIN ERROR       3.229   alpha      0.80820
**** Epoch 107       WITH MIN ERROR       3.229   alpha      0.80370
**** Epoch 109       WITH MIN ERROR       3.225   alpha      0.80190
**** Epoch 111       WITH MIN ERROR       3.222   alpha      0.80010
**** Epoch 113       WITH MIN ERROR       3.218   alpha      0.79830
Epoch 125   min err        3.21823   min epoch 113   alpha      0.78840
**** Epoch 126       WITH MIN ERROR       3.218   alpha      0.78660
**** Epoch 128       WITH MIN ERROR       3.214   alpha      0.78480
**** Epoch 130       WITH MIN ERROR       3.211   alpha      0.78300
**** Epoch 133       WITH MIN ERROR       3.206   alpha      0.78030
**** Epoch 136       WITH MIN ERROR       3.201   alpha      0.77760
**** Epoch 139       WITH MIN ERROR       3.196   alpha      0.77490
**** Epoch 142       WITH MIN ERROR       3.191   alpha      0.77220
**** Epoch 146       WITH MIN ERROR       3.065   alpha      0.76860
**** Epoch 149       WITH MIN ERROR       3.060   alpha      0.76590
**** Epoch 165       WITH MIN ERROR       3.024   alpha      0.75150
**** Epoch 167       WITH MIN ERROR       3.008   alpha      0.74970
**** Epoch 169       WITH MIN ERROR       3.004   alpha      0.74790
**** Epoch 171       WITH MIN ERROR       3.000   alpha      0.74610
**** Epoch 173       WITH MIN ERROR       2.996   alpha      0.74430
**** Epoch 175       WITH MIN ERROR       2.993   alpha      0.74250
**** Epoch 177       WITH MIN ERROR       2.989   alpha      0.74070
**** Epoch 179       WITH MIN ERROR       2.985   alpha      0.73890
**** Epoch 181       WITH MIN ERROR       2.982   alpha      0.73710
**** Epoch 183       WITH MIN ERROR       2.978   alpha      0.73530
**** Epoch 185       WITH MIN ERROR       2.974   alpha      0.73350
**** Epoch 187       WITH MIN ERROR       2.971   alpha      0.73170
**** Epoch 189       WITH MIN ERROR       2.967   alpha      0.72990
**** Epoch 191       WITH MIN ERROR       2.964   alpha      0.72810
**** Epoch 193       WITH MIN ERROR       2.960   alpha      0.72630
**** Epoch 195       WITH MIN ERROR       2.957   alpha      0.72450
**** Epoch 197       WITH MIN ERROR       2.953   alpha      0.72270
**** Epoch 199       WITH MIN ERROR       2.950   alpha      0.72090
**** Epoch 201       WITH MIN ERROR       2.946   alpha      0.71910
**** Epoch 203       WITH MIN ERROR       2.943   alpha      0.71730
**** Epoch 205       WITH MIN ERROR       2.940   alpha      0.71550
```

```
**** Epoch 207       WITH MIN ERROR       2.936   alpha      0.71370
**** Epoch 209       WITH MIN ERROR       2.933   alpha      0.71190
**** Epoch 211       WITH MIN ERROR       2.921   alpha      0.71010
**** Epoch 213       WITH MIN ERROR       2.918   alpha      0.70830
**** Epoch 215       WITH MIN ERROR       2.915   alpha      0.70650
**** Epoch 217       WITH MIN ERROR       2.912   alpha      0.70470
**** Epoch 219       WITH MIN ERROR       2.909   alpha      0.70290
**** Epoch 221       WITH MIN ERROR       2.906   alpha      0.70110
**** Epoch 223       WITH MIN ERROR       2.903   alpha      0.69930
**** Epoch 225       WITH MIN ERROR       2.863   alpha      0.69750
**** Epoch 227       WITH MIN ERROR       2.861   alpha      0.69570
**** Epoch 229       WITH MIN ERROR       2.858   alpha      0.69390
**** Epoch 231       WITH MIN ERROR       2.855   alpha      0.69210
**** Epoch 233       WITH MIN ERROR       2.852   alpha      0.69030
**** Epoch 235       WITH MIN ERROR       2.849   alpha      0.68850
**** Epoch 241       WITH MIN ERROR       2.843   alpha      0.68310
**** Epoch 243       WITH MIN ERROR       2.840   alpha      0.68130
Epoch 250   min err        2.83977   min epoch 243   alpha      0.67590
**** Epoch 281       WITH MIN ERROR       2.783   alpha      0.64710
**** Epoch 283       WITH MIN ERROR       2.780   alpha      0.64530
**** Epoch 285       WITH MIN ERROR       2.777   alpha      0.64350
**** Epoch 287       WITH MIN ERROR       2.774   alpha      0.64170
**** Epoch 289       WITH MIN ERROR       2.772   alpha      0.63990
**** Epoch 291       WITH MIN ERROR       2.769   alpha      0.63810
**** Epoch 293       WITH MIN ERROR       2.766   alpha      0.63630
**** Epoch 295       WITH MIN ERROR       2.764   alpha      0.63450
**** Epoch 297       WITH MIN ERROR       2.761   alpha      0.63270
**** Epoch 299       WITH MIN ERROR       2.758   alpha      0.63090
**** Epoch 301       WITH MIN ERROR       2.756   alpha      0.62910
**** Epoch 303       WITH MIN ERROR       2.753   alpha      0.62730
**** Epoch 305       WITH MIN ERROR       2.751   alpha      0.62550
**** Epoch 307       WITH MIN ERROR       2.748   alpha      0.62370
**** Epoch 309       WITH MIN ERROR       2.746   alpha      0.62190
**** Epoch 311       WITH MIN ERROR       2.687   alpha      0.62010
**** Epoch 320       WITH MIN ERROR       2.636   alpha      0.61200
**** Epoch 323       WITH MIN ERROR       2.632   alpha      0.60930
**** Epoch 326       WITH MIN ERROR       2.628   alpha      0.60660
Epoch 375   min err        2.62765   min epoch 326   alpha      0.56340
**** Epoch 485       WITH MIN ERROR       2.558   alpha      0.46350
Epoch 500   min err        2.55849   min epoch 485   alpha      0.45090
**** Epoch 539       WITH MIN ERROR       2.554   alpha      0.41490
**** Epoch 551       WITH MIN ERROR       2.394   alpha      0.40410
```

```
**** Epoch 621       WITH MIN ERROR        2.362   alpha      0.34110
Epoch 625   min err        2.36245   min epoch 621   alpha      0.33840
**** Epoch 702       WITH MIN ERROR        2.186   alpha      0.26820
**** Epoch 744       WITH MIN ERROR        2.160   alpha      0.23040
Epoch 750   min err        2.15974   min epoch 744   alpha      0.22590
Epoch 875   min err        2.15974   min epoch 744   alpha      0.11340
**** Epoch 941       WITH MIN ERROR        1.859   alpha      0.05310
Epoch 1000   min err        1.85912   min epoch 941   alpha      0.00100

Min alpha reached

###############################################################################
# KB_CAT KNOWLEDGE DISCOVERY IN DATA MINING (CATALOG PROGRAM)                 #
# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                 #
# Language used: PYTHON                                                      #
###############################################################################
EPOCH 941   WITH MIN ERROR       1.859 starting alpha      0.90000   ending
alpha      0.00100 Iterations 39960 Total Epochs 999
Output File Catalog.original vessels_M_g3_out.txt
Output File Catalog.sort     vessels_M_g3_outsrt.txt
Output File Summary sort     vessels_M_g3_sort.txt
Output File Matrix Catal.    vessels_M_g3_catal.txt
Output File Means, STD, CV.  vessels_M_g3_medsd.txt
Output File CV of the Groups vessels_M_g3_cv.txt
Output File Training Grid    vessels_M_g3_grid.txt
Output File Run Parameters   vessels_M_g3_log.txt
Elapsed time (seconds)   :   15
*KIndex*     0.8438
```

As you can see, during the processing, the minimum error decreases from 3.616 (epoch 15) to 1.859 epoch 941).

The processing was completed at the epoch 1000, when the parameter value *alpha* reaches a minimum value of 0.001.

References to output files are also listed:

- Catalog.original = input file *cataloged,* NOT in order of groups and with original values (NOT normalized)
- Catalog.sort = input file *cataloged*, IN ORDER of groups and with original values (NOT normalized)
- Summary.sort = input file *cataloged*, IN ORDER of groups and with NORMALIZED values.
- Matrix Catal. = files with three columns (progressive number of records, group codes and subgroup codes)
- Means, STD, CV = files with a column for every variable and with three lines (mean, standard deviation and coefficient of variation)
- CV of the Groups = files of the coefficient of variations of the groups and

of the variables / columns with totals of the records classified into groups

- Training Grid = files containing the values of the training matrix with minimum error
- Run Parameters = files containing references to input files, parameters of computing and output files
- KIndex (Knowledge Index) is a KB index that measures how much knowledge is contained in the cataloged groups: if KIndex reached its maximum value of 1, every group would be made up of records with constant values in all variables / columns and each group would different from the other groups.

KIndex is calculated using means of CV of the variables / columns of the groups of input files before cataloging: see the source program KB_CAT for the computing details.

In the case under examination, the Kindex value, not particularly high (0.8438), suggests to run a new processing increasing, for example, the number of groups from 3 to 4 obtaining a certain improvement of Kindex.

File - Output/Catalog.original (vessels_M_g3_out.txt)

It is identical to the input file with the addition of the column for the input of the code of the group it belongs to.

The Output/Catalog.sort file is more interesting, in that it shows the classified records that each group belong to.

File of Output/Catalog.sort (vessels_*M_g3_outsrt.txt*)

This is identical to the previous file but the records / cases are in order according to the code of the group it belongs to.

Group	description	shape	material	height	colour	weight	haft	plug
G_00_00	ancient_bottle	ball_cone	glass	40	Green	150	no	cork
G_00_00	bottle_1	cylinder_cone	glass	40	Green	120	no	cork
G_00_00	bottle_4	cylinder_cone	glass	35	Green	125	no	metal
G_00_00	carboy	ball_cone	glass	80	Green	15000	no	cork
G_00_00	magnum_bottle	cylinder_cone	glass	50	Green	170	no	metal
G_00_00	plant_pot	cut_cone	terracotta	30	Brown	200	no	no
G_00_00	umbrella_stand	cylinder	metal	100	Grey	3000	no	no
G_00_01	pot_1	cylinder	metal	40	Grey	500	two	yes
G_00_02	coffee_cup	cut_cone	ceramic	6	White	20	yes	no
G_00_02	cup_1	cut_cone	ceramic	10	White	30	yes	no
G_00_02	cup_2	cut_cone	glass	20	transparent	35	yes	no
G_00_02	pot_2	cut_cone	metal	7	Grey	200	yes	yes
G_01_00	beer_jug	cut_cone	porcelain	18	severals	25	no	no
G_01_00	bottle_2	cylinder_cone	glass	40	transparent	125	no	cork
G_01_00	bottle_3	cylinder_cone	glass	45	opaque	125	no	plastic
G_01_00	glass_1	cut_cone	pewter	10	pewter	20	no	no
G_01_00	glass_3	cut_cone	terracotta	8	Grey	20	no	no
G_01_00	tuna_can	cylinder	metal	10	severals	10	no	no
G_02_00	cd	parallelepiped	plastic	1	transparent	4	no	no
G_02_00	champagne_glass	cut_cone	crystal	17	transparent	17	no	no

Group	description	shape	material	height	colour	weight	haft	plug
G_02_00	dessert_glass	cut_cone	glass	17	transparent	17	no	no
G_02_00	glass_2	cut_cone	plastic	9	White	4	no	no
G_02_00	pasta_case	parallelepiped	glass	35	transparent	150	no	metal
G_02_00	perfume	parallelepiped	glass	7	transparent	15	no	plastic
G_02_00	tetrapack1	parallelepiped	mixed	40	severals	20	no	plastic
G_02_00	tetrapack2	parallelepiped	plastic	40	severals	21	no	plastic
G_02_00	tetrapack3	parallelepiped	millboard	40	severals	22	no	no
G_02_00	toothpaste	cylinder	plastic	15	severals	7	no	plastic
G_02_00	trousse	cylinder	plastic	1	silver	7	no	yes
G_02_00	tuna_tube	cylinder	plastic	15	severals	7	no	plastic
G_02_00	visage_cream	cylinder	metal	15	White	7	no	no
G_02_00	wine_glass	cut_cone	glass	15	transparent	15	no	no
G_02_01	pyrex	parallelepiped	glass	10	transparent	300	two	glass
G_02_02	cleaning_1	parall_cone	plastic	30	White	50	yes	plastic
G_02_02	cleaning_2	cylinder_cone	plastic	30	Blue	60	yes	plastic
G_02_02	cleaning_3	cone	plastic	100	severals	110	yes	plastic
G_02_02	jug	cylinder	terracotta	25	White	40	yes	no
G_02_02	milk_cup	cut_cone	terracotta	15	Blue	35	yes	no
G_02_02	tea_cup	cut_cone	terracotta	7	White	30	yes	no
G_02_02	watering_can	irregular	plastic	50	Green	400	yes	no

On first sight you can see that the program KB_CAT is able to catalog records in homogeneous groups for content.
It is important to note that the vessels.txt files are formed by just 40 records which are all quite different.
For example:

- the group G_00_00 is characterised by objects that are primarily of a green colour, and with haft
- the group G_00_02 is primarily formed by objects of a cut_cone shape, with haft and without a plug
- the group G_02_00 is characterised by objects that are parallelepiped / cylinder / cut_cone shape and without haft
- the group G_02_02 is made up of plastic and terracotta objects with haft

If the processed input file had been formed with numerous records and with many variables / columns, it would not have been so easy to draw conclusions on the results of the cataloging only visually examining the files.
The KB_STA program is dedicated to resolving the problem which has just been highlighted.

Output/Means, Std, CV (vessels_M_g3_medsd.txt)

File containing the Means, the Maximums, the Std and the CV with normalized values of the whole population.
Low values of the CV (*coefficient of variation*) indicate that the values of the variables / columns are not dispersed.

shape	material	height	colour	weight	haft	plug
Mean1	Mean2	Mean3	Mean4	Mean5	Mean6	Mean7

shape	material	height	colour	weight	haft	plug
0.4892	0.5000	28.075	0.6222	530.32	0.3000	0.5900
Max1 1.0000	Max2 1.0000	Max3 100.0	Max4 1.0000	Max5 15000.	Max6 1.0000	Max7 1.0000
Std1 0.7371	Std2 0.8164	Std3 60.592	Std4 0.8210	Std5 6103.1	Std6 1.1474	Std7 0.6526
CV_1 1.5066	CV_2 1.6329	CV_3 2.1582	CV_4 1.3194	CV_5 11.508	CV_6 3.8248	CV_7 1.1062

Output/CV files (*vessels_M_g3_cv.txt*)

Groups	shape	material	height	colour	weight	haft	plug	Means	N_recs
G_00_00	0.69	0.77	0.45	0.27	1.91	0	0.91	0.71	7
G_00_01	0	0	0	0	0	0	0	0	1
G_00_02	0	1.04	0.52	0.34	1.05	0	0.25	0.46	4
G_01_00	0.32	0.57	0.69	0.30	0.93	0	0.47	0.47	6
G_02_00	0.51	0.52	0.71	0.15	1.61	0	0.21	0.53	14
G_02_01	0	0	0	0	0	0	0	0	1
G_02_02	0.51	0.13	0.78	0.79	1.19	0	0.14	0.51	7
Means	0.44	0.53	0.62	0.32	1.35	0	0.35	0.51	40
Total	1.51	1.63	2.16	1.32	11.51	3.82	1.11	3.29	40

The file contains information relevant for measuring the quality of the cataloging. The value contained in every cell represents the importance of the values of the variables / columns in the group: the more the value is close to zero, the more the variable / column is important in the cataloging.
If the value is equal to zero, the variable / column in that group will have an identical value: for example all groups have identical values in the variable *haft*.
The values in the cells of the penultimate column (Means) indicate if the groups are internally homogeneous considering all the variables / columns: the higher the value is close to zero, the greater the similarity of the record / cases to each other within the group under consideration.
The groups G_00_02 and G_01_00 are homogeneous, while the group G_00_00 is not, due to the important CV values of the variables *weight* and *plug*.
It is also important to compare the values contained in every line / column with the value contained in the last two lines: **Means** and **Total** (referring to the all records before the cataloging).

Output/Training Grid (*vessels_M_g3_grid.txt*)

The file contains the values of the three-dimensional training matrix with minimum error; this matrix is used by the KB_CLA program used to classify new records /

cases that can be recognised and classified according to what has previously been learnt by the program KB_CAT.

Group	SubGroup	Variable/Column	Values
0	0	0	0,3976159
0	0	1	0,4249143
0	0	2	0,4221095
0	0	3	0,3706712
0	0	4	0,1070639
0	0	5	0,0721792
0	0	6	0,4288610
0	1	0	0,3760895
0	1	1	0,3555886
0	1	2	0,3351283
0	1	3	0,4836650
0	1	4	0,0767009
0	1	5	0,3319249
0	1	6	0,5141450
0	2	0	0,3522021
0	2	1	0,1886213
0	2	2	0,1638941
0	2	3	0,6998640
0	2	4	0,0115530
0	2	5	0,8734927
0	2	6	0,7434203
1	0	0	0,5722823
1	0	1	0,4691723
1	0	2	0,2864130
1	0	3	0,6216960
1	0	4	0,0428225
1	0	5	0,0569301
1	0	6	0,5809196
1	1	0	0,5466298
1	1	1	0,5135355
1	1	2	0,2899887
1	1	3	0,6104640

Group	SubGroup	Variable/Column	Values
1	1	4	0,0296109
1	1	5	0,3230673
1	1	6	0,6348858
1	2	0	0,4737209
1	2	1	0,5358513
1	2	2	0,2805610
1	2	3	0,6148486
1	2	4	0,0108941
1	2	5	0,9004934
1	2	6	0,6831299
2	0	0	0,6283160
2	0	1	0,4785080
2	0	2	0,2024570
2	0	3	0,7459708
2	0	4	0,0055453
2	0	5	0,0683992
2	0	6	0,6433004
2	1	0	0,6078937
2	1	1	0,5633861
2	1	2	0,2537548
2	1	3	0,6914334
2	1	4	0,0067944
2	1	5	0,2961828
2	1	6	0,6576649
2	2	0	0,5420435
2	2	1	0,7055653
2	2	2	0,3505488
2	2	3	0,5606647
2	2	4	0,0126543
2	2	5	0,8661729
2	2	6	0,6630445

Statistical analysis of the results of the cataloging

The file contains the results of the processing of KB_CAT statistically analysed running the program KB_STA, using the parameters below listed.

```
###########################################################################
# KB_STA KNOWLEDGE DISCOVERY IN DATA MINING (STATISTICAL PROGRAM)        #
# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)            #
# Language used: PYTHON                                                  #
###########################################################################
INPUT - Catalogued Records File (_outsrt.txt)        ->
vessels_M_g3_outsrt.txt INPUT - Groups / CV File (_cv.txt)            ->
vessels_M_g3_cv.txt
Group Consistency (% from 0 to 100)                  ->  0
Variable Consistency (% from 0 to 100)               ->  0
Select groups containing records >=                  ->  4
Select groups containing records <=                  ->  1000
Summary / Detail report (S / D)                      ->  D
Display Input Records (Y / N)                        ->  Y
=========================OUTPUT============================================
Report File                                          ->  vessels_M_g3_sta.txt

KB_STA - Statistical Analysis from:  vessels_M_g3_outsrt.txt
and from:  vessels_M_g3_cv.txt
Min Perc. of group Consistency:  0  Min Perc. of variable Consistency:  0
Min Number of records:  4  Max Number of records:  1000
by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)
===========================================================================
G_00_00 Consistency 0.7140  %Consistency  0.0 Records  7    %Records    17.50
***  shape  Consistency  0.6910 %Consistency      3.22
G_00_00      ID record    ancient_bottle  Value ball_cone
G_00_00      ID record    bottle_1        Value cylinder_cone
G_00_00      ID record    bottle_4        Value cylinder_cone
G_00_00      ID record    carboy          Value ball_cone
G_00_00      ID record    magnum_bottle   Value cylinder_cone
G_00_00      ID record    plant_pot       Value cut_cone
G_00_00      ID record    umbrella_stand  Value cylinder
Value  cylinder_cone    Frequency         3    Percentage   42.00
Value  ball_cone        Frequency         2    Percentage   28.00
Value  cylinder         Frequency         1    Percentage   14.00
Value  cut_cone         Frequency         1    Percentage   14.00
***  material Consistency          0.7687       %Consistency              0.00
G_00_00      ID record    ancient_bottle  Value glass
G_00_00      ID record    bottle_1        Value glass
G_00_00      ID record    bottle_4        Value glass
G_00_00      ID record    carboy          Value glass
G_00_00      ID record    magnum_bottle   Value glass
G_00_00      ID record    plant_pot       Value terracotta
G_00_00      ID record    umbrella_stand  Value metal
Value  glass            Frequency         5    Percentage   71.00
Value  terracotta       Frequency         1    Percentage   14.00
Value  metal            Frequency         1    Percentage   14.00
***  height   Consistency          0.4537       %Consistency             36.46
G_00_00      ID record    ancient_bottle  Value 40.0
G_00_00      ID record    bottle_1        Value 40.0
G_00_00      ID record    bottle_4        Value 35.0
G_00_00      ID record    carboy          Value 80.0
G_00_00      ID record    magnum_bottle   Value 50.0
G_00_00      ID record    plant_pot       Value 30.0
G_00_00      ID record    umbrella_stand  Value 100.0
Min     30.00      Max     100.00     Step    17.50
First  Quartile (end)        47.50    Frequency %        57.14
Second Quartile (end)        65.00    Frequency %        14.29
Third  Quartile (end)        82.50    Frequency %        14.29
Fourth Quartile (end)       100.00    Frequency %        14.29
```

```
***  colour    Consistency        0.2673      %Consistency           62.56
G_00_00     ID record    ancient_bottle  Value green
G_00_00     ID record    bottle_1        Value green
G_00_00     ID record    bottle_4        Value green
G_00_00     ID record    carboy          Value green
G_00_00     ID record    magnum_bottle   Value green
G_00_00     ID record    plant_pot       Value brown
G_00_00     ID record    umbrella_stand  Value grey
Value  green           Frequency          5    Percentage   71.00
Value  grey            Frequency          1    Percentage   14.00
Value  brown           Frequency          1    Percentage   14.00
***  weight   Consistency         1.9116      %Consistency            0.00
G_00_00     ID record    ancient_bottle  Value 150.0
G_00_00     ID record    bottle_1        Value 120.0
G_00_00     ID record    bottle_4        Value 125.0
G_00_00     ID record    carboy          Value 15000.0
G_00_00     ID record    magnum_bottle   Value 170.0
G_00_00     ID record    plant_pot       Value 200.0
G_00_00     ID record    umbrella_stand  Value 3000.0
Min     120.00     Max     15000.00   Step    3720.00
First  Quartile (end)       3840.00    Frequency %        85.71
Fourth Quartile (end)      15000.00    Frequency %        14.29
***  haft   Consistency      0.0000      %Consistency          100.00
G_00_00     ID record    ancient_bottle  Value no
G_00_00     ID record    bottle_1        Value no
G_00_00     ID record    bottle_4        Value no
G_00_00     ID record    carboy          Value no
G_00_00     ID record    magnum_bottle   Value no
G_00_00     ID record    plant_pot       Value no
G_00_00     ID record    umbrella_stand  Value no
Value  no              Frequency          7    Percentage   100.00
***  plug   Consistency      0.9055      %Consistency            0.00
G_00_00     ID record    ancient_bottle  Value cork
G_00_00     ID record    bottle_1        Value cork
G_00_00     ID record    bottle_4        Value metal
G_00_00     ID record    carboy          Value cork
G_00_00     ID record    magnum_bottle   Value metal
G_00_00     ID record    plant_pot       Value no
G_00_00     ID record    umbrella_stand  Value no
Value  cork            Frequency          3    Percentage   42.00
Value  no              Frequency          2    Percentage   28.00
Value  metal           Frequency          2    Percentage   28.00
=============================================================================
G_00_02   Consistency  0.4559 %Consistency   12 Records  4  %Records    10.00
***  shape    Consistency         0.0000      %Consistency          100.00
G_00_02     ID record    coffee_cup      Value cut_cone
G_00_02     ID record    cup_1           Value cut_cone
G_00_02     ID record    cup_2           Value cut_cone
G_00_02     ID record    pot_2           Value cut_cone
Value  cut_cone        Frequency          4    Percentage   100.00
***  material   Consistency   1.0392      %Consistency            0.00
G_00_02     ID record    coffee_cup      Value ceramic
G_00_02     ID record    cup_1           Value ceramic
G_00_02     ID record    cup_2           Value glass
G_00_02     ID record    pot_2           Value metal
Value  ceramic         Frequency          2    Percentage   50.00
Value  metal           Frequency          1    Percentage   25.00
Value  glass           Frequency          1    Percentage   25.00
***  height  Consistency          0.5153      %Consistency            0.00
G_00_02     ID record    coffee_cup      Value 6.0
```

```
G_00_02     ID record    cup_1           Value 10.0
G_00_02     ID record    cup_2           Value 20.0
G_00_02     ID record    pot_2           Value 7.0
Min    6.00       Max    20.00      Step   3.50
First  Quartile (end)        9.50   Frequency %        50.00
Second Quartile (end)       13.00   Frequency %        25.00
Fourth Quartile (end)       20.00   Frequency %        25.00
***  colour  Consistency          0.3431       %Consistency            24.74
G_00_02     ID record    coffee_cup      Value white
G_00_02     ID record    cup_1           Value white
G_00_02     ID record    cup_2           Value transparent
G_00_02     ID record    pot_2           Value grey
Value  white             Frequency           2     Percentage   50.00
Value  transparent        Frequency          1     Percentage   25.00
Value  grey              Frequency           1     Percentage   25.00
***  weight  Consistency          1.0460       %Consistency             0.00
G_00_02     ID record    coffee_cup      Value 20.0
G_00_02     ID record    cup_1           Value 30.0
G_00_02     ID record    cup_2           Value 35.0
G_00_02     ID record    pot_2           Value 200.0
Min    20.00      Max    200.00     Step   45.00
First  Quartile (end)       65.00   Frequency %        75.00
Fourth Quartile (end)      200.00   Frequency %        25.00
***  haft     Consistency         0.0000       %Consistency           100.00
G_00_02     ID record    coffee_cup      Value yes
G_00_02     ID record    cup_1           Value yes
G_00_02     ID record    cup_2           Value yes
G_00_02     ID record    pot_2           Value yes
Value  yes               Frequency           4     Percentage   100.00
***  plug   Consistency       0.2474       %Consistency            45.73
G_00_02     ID record    coffee_cup      Value no
G_00_02     ID record    cup_1           Value no
G_00_02     ID record    cup_2           Value no
G_00_02     ID record    pot_2           Value yes
Value  no                Frequency           3     Percentage   75.00
Value  yes               Frequency           1     Percentage   25.00
================================================================================
G_01_00  Consistency  0.4666 %Consistency 10 Records    6    %Records    15.00
***  shape   Consistency          0.3168       %Consistency            32.10
G_01_00     ID record    beer_jug        Value cut_cone
G_01_00     ID record    bottle_2        Value cylinder_cone
G_01_00     ID record    bottle_3        Value cylinder_cone
G_01_00     ID record    glass_1         Value cut_cone
G_01_00     ID record    glass_3         Value cut_cone
G_01_00     ID record    tuna_can        Value cylinder
Value  cut_cone          Frequency           3     Percentage   50.00
Value  cylinder_cone     Frequency           2     Percentage   33.00
Value  cylinder          Frequency           1     Percentage   16.00
***  material Consistency       0.5657    %Consistency            0.00
G_01_00     ID record    beer_jug        Value porcelain
G_01_00     ID record    bottle_2        Value glass
G_01_00     ID record    bottle_3        Value glass
G_01_00     ID record    glass_1         Value pewter
G_01_00     ID record    glass_3         Value terracotta
G_01_00     ID record    tuna_can        Value metal
Value  glass             Frequency           2     Percentage   33.00
Value  terracotta        Frequency           1     Percentage   16.00
Value  porcelain         Frequency           1     Percentage   16.00
Value  pewter            Frequency           1     Percentage   16.00
Value  metal             Frequency           1     Percentage   16.00
```

```
***  height   Consistency          0.6877       %Consistency        0.00
G_01_00      ID record    beer_jug         Value 18.0
G_01_00      ID record    bottle_2         Value 40.0
G_01_00      ID record    bottle_3         Value 45.0
G_01_00      ID record    glass_1          Value 10.0
G_01_00      ID record    glass_3          Value 8.0
G_01_00      ID record    tuna_can         Value 10.0
Min    8.00         Max    45.00      Step   9.25
First  Quartile (end)        17.25   Frequency %        50.00
Second Quartile (end)        26.50   Frequency %        16.67
Fourth Quartile (end)        45.00   Frequency %        33.33
***  colour    Consistency         0.2997       %Consistency            35.77
G_01_00      ID record    beer_jug         Value severals
G_01_00      ID record    bottle_2         Value transparent
G_01_00      ID record    bottle_3         Value opaque
G_01_00      ID record    glass_1          Value pewter
G_01_00      ID record    glass_3          Value grey
G_01_00      ID record    tuna_can         Value severals
Value  severals         Frequency          2    Percentage   33.00
Value  transparent       Frequency         1    Percentage   16.00
Value  pewter           Frequency          1    Percentage   16.00
Value  opaque           Frequency          1    Percentage   16.00
Value  grey             Frequency          1    Percentage   16.00
***  weight    Consistency         0.9283       %Consistency             0.00
G_01_00      ID record    beer_jug         Value 25.0
G_01_00      ID record    bottle_2         Value 125.0
G_01_00      ID record    bottle_3         Value 125.0
G_01_00      ID record    glass_1          Value 20.0
G_01_00      ID record    glass_3          Value 20.0
G_01_00      ID record    tuna_can         Value 10.0
Min    10.00        Max    125.00     Step   28.75
First  Quartile (end)        38.75   Frequency %        66.67
Fourth Quartile (end)       125.00   Frequency %        33.33
***  haft  Consistency       0.0000       %Consistency          100.00
G_01_00      ID record    beer_jug         Value no
G_01_00      ID record    bottle_2         Value no
G_01_00      ID record    bottle_3         Value no
G_01_00      ID record    glass_1          Value no
G_01_00      ID record    glass_3          Value no
G_01_00      ID record    tuna_can         Value no
Value  no          Frequency           6    Percentage   100.00
***  plug   Consistency        0.4677       %Consistency             0.00
G_01_00      ID record    beer_jug         Value no
G_01_00      ID record    bottle_2         Value cork
G_01_00      ID record    bottle_3         Value plastic
G_01_00      ID record    glass_1          Value no
G_01_00      ID record    glass_3          Value no
G_01_00      ID record    tuna_can         Value no
Value  no               Frequency          4    Percentage   66.00
Value  plastic          Frequency          1    Percentage   16.00
Value  cork             Frequency          1    Percentage   16.00
===============================================================================
G_02_00  Consistency 0.5300 %Consistency  0.0 Records 14    %Records    35.00
***  shape Consistency       0.5100       %Consistency            3.77
G_02_00      ID record    cd               Value parallelepiped
G_02_00      ID record    champagne_glass  Value cut_cone
G_02_00      ID record    dessert_glass    Value cut_cone
G_02_00      ID record    glass_2          Value cut_cone
G_02_00      ID record    pasta_case       Value parallelepiped
G_02_00      ID record    perfume          Value parallelepiped
```

```
G_02_00     ID record   tetrapack1      Value parallelepiped
G_02_00     ID record   tetrapack2      Value parallelepiped
G_02_00     ID record   tetrapack3      Value parallelepiped
G_02_00     ID record   toothpaste      Value cylinder
G_02_00     ID record   trousse         Value cylinder
G_02_00     ID record   tuna_tube       Value cylinder
G_02_00     ID record   visage_cream    Value cylinder
G_02_00     ID record   wine_glass      Value cut_cone
Value  parallelepiped  Frequency         6    Percentage   42.00
Value  cylinder        Frequency         4    Percentage   28.00
Value  cut_cone        Frequency         4    Percentage   28.00
***  material Consistency         0.5228       %Consistency            1.36
G_02_00     ID record   cd              Value plastic
G_02_00     ID record   champagne_glass Value crystal
G_02_00     ID record   dessert_glass   Value glass
G_02_00     ID record   glass_2         Value plastic
G_02_00     ID record   pasta_case      Value glass
G_02_00     ID record   perfume         Value glass
G_02_00     ID record   tetrapack1      Value mixed
G_02_00     ID record   tetrapack2      Value plastic
G_02_00     ID record   tetrapack3      Value millboard
G_02_00     ID record   toothpaste      Value plastic
G_02_00     ID record   trousse         Value plastic
G_02_00     ID record   tuna_tube       Value plastic
G_02_00     ID record   visage_cream    Value metal
G_02_00     ID record   wine_glass      Value glass
Value  plastic         Frequency         6    Percentage   42.00
Value  glass           Frequency         4    Percentage   28.00
Value  mixed           Frequency         1    Percentage   7.00
Value  millboard       Frequency         1    Percentage   7.00
Value  metal           Frequency         1    Percentage   7.00
Value  crystal         Frequency         1    Percentage   7.00
***  height  Consistency          0.7067       %Consistency            0.00
G_02_00     ID record   cd              Value 1.0
G_02_00     ID record   champagne_glass Value 17.0
G_02_00     ID record   dessert_glass   Value 17.0
G_02_00     ID record   glass_2         Value 9.0
G_02_00     ID record   pasta_case      Value 35.0
G_02_00     ID record   perfume         Value 7.0
G_02_00     ID record   tetrapack1      Value 40.0
G_02_00     ID record   tetrapack2      Value 40.0
G_02_00     ID record   tetrapack3      Value 40.0
G_02_00     ID record   toothpaste      Value 15.0
G_02_00     ID record   trousse         Value 1.0
G_02_00     ID record   tuna_tube       Value 15.0
G_02_00     ID record   visage_cream    Value 15.0
G_02_00     ID record   wine_glass      Value 15.0
Min    1.00       Max    40.00      Step   9.75
First  Quartile (end)        10.75   Frequency %        28.57
Second Quartile (end)        20.50   Frequency %        42.86
Fourth Quartile (end)        40.00   Frequency %        28.57
***  colour    Consistency        0.1507       %Consistency           71.57
G_02_00     ID record   cd              Value transparent
G_02_00     ID record   champagne_glass Value transparent
G_02_00     ID record   dessert_glass   Value transparent
G_02_00     ID record   glass_2         Value white
G_02_00     ID record   pasta_case      Value transparent
G_02_00     ID record   perfume         Value transparent
G_02_00     ID record   tetrapack1      Value severals
G_02_00     ID record   tetrapack2      Value severals
```

```
G_02_00     ID record    tetrapack3       Value severals
G_02_00     ID record    toothpaste       Value severals
G_02_00     ID record    trousse          Value silver
G_02_00     ID record    tuna_tube        Value severals
G_02_00     ID record    visage_cream     Value white
G_02_00     ID record    wine_glass       Value transparent
Value  transparent       Frequency          6    Percentage   42.00
Value  severals         Frequency           5    Percentage   35.00
Value  white            Frequency           2    Percentage   14.00
Value  silver           Frequency           1    Percentage   7.00
***  weight   Consistency         1.6075      %Consistency             0.00
G_02_00     ID record    cd               Value 4.0
G_02_00     ID record    champagne_glass Value 17.0
G_02_00     ID record    dessert_glass    Value 17.0
G_02_00     ID record    glass_2          Value 4.0
G_02_00     ID record    pasta_case       Value 150.0
G_02_00     ID record    perfume          Value 15.0
G_02_00     ID record    tetrapack1       Value 20.0
G_02_00     ID record    tetrapack2       Value 21.0
G_02_00     ID record    tetrapack3       Value 22.0
G_02_00     ID record    toothpaste       Value 7.0
G_02_00     ID record    trousse          Value 7.0
G_02_00     ID record    tuna_tube        Value 7.0
G_02_00     ID record    visage_cream     Value 7.0
G_02_00     ID record    wine_glass       Value 15.0
Min     4.00        Max     150.00     Step   36.50
First  Quartile (end)          40.50    Frequency %          92.86
Fourth Quartile (end)         150.00    Frequency %           7.14
***  haft    Consistency      0.0000       %Consistency           100.00
G_02_00     ID record    cd               Value no
G_02_00     ID record    champagne_glass Value no
G_02_00     ID record    dessert_glass    Value no
G_02_00     ID record    glass_2          Value no
G_02_00     ID record    pasta_case       Value no
G_02_00     ID record    perfume          Value no
G_02_00     ID record    tetrapack1       Value no
G_02_00     ID record    tetrapack2       Value no
G_02_00     ID record    tetrapack3       Value no
G_02_00     ID record    toothpaste       Value no
G_02_00     ID record    trousse          Value no
G_02_00     ID record    tuna_tube        Value no
G_02_00     ID record    visage_cream     Value no
G_02_00     ID record    wine_glass       Value no
Value  no               Frequency          14    Percentage   100.00
***  plug    Consistency      0.2125       %Consistency            59.91
G_02_00     ID record    cd               Value no
G_02_00     ID record    champagne_glass Value no
G_02_00     ID record    dessert_glass    Value no
G_02_00     ID record    glass_2          Value no
G_02_00     ID record    pasta_case       Value metal
G_02_00     ID record    perfume          Value plastic
G_02_00     ID record    tetrapack1       Value plastic
G_02_00     ID record    tetrapack2       Value plastic
G_02_00     ID record    tetrapack3       Value no
G_02_00     ID record    toothpaste       Value plastic
G_02_00     ID record    trousse          Value yes
G_02_00     ID record    tuna_tube        Value plastic
G_02_00     ID record    visage_cream     Value no
G_02_00     ID record    wine_glass       Value no
Value  no               Frequency           7    Percentage   50.00
```

```
Value  plastic          Frequency          5    Percentage   35.00
Value  yes              Frequency          1    Percentage   7.00
Value  metal            Frequency          1    Percentage   7.00
===============================================================================
G_02_02 Consistency  0.5053 %Consistency  2 Records   7    %Records    17.50
***  shape     Consistency        0.5070       %Consistency            0.00
G_02_02     ID record    cleaning_1      Value parall_cone
G_02_02     ID record    cleaning_2      Value cylinder_cone
G_02_02     ID record    cleaning_3      Value cone
G_02_02     ID record    jug             Value cylinder
G_02_02     ID record    milk_cup        Value cut_cone
G_02_02     ID record    tea_cup         Value cut_cone
G_02_02     ID record    watering_can    Value irregular
Value  cut_cone         Frequency          2    Percentage   28.00
Value  parall_cone      Frequency          1    Percentage   14.00
Value  irregular        Frequency          1    Percentage   14.00
Value  cylinder_cone    Frequency          1    Percentage   14.00
Value  cylinder         Frequency          1    Percentage   14.00
Value  cone             Frequency          1    Percentage   14.00
***  material   Consistency       0.1260       %Consistency           75.06
G_02_02     ID record    cleaning_1      Value plastic
G_02_02     ID record    cleaning_2      Value plastic
G_02_02     ID record    cleaning_3      Value plastic
G_02_02     ID record    jug             Value terracotta
G_02_02     ID record    milk_cup        Value terracotta
G_02_02     ID record    tea_cup         Value terracotta
G_02_02     ID record    watering_can    Value plastic
Value  plastic          Frequency          4    Percentage   57.00
Value  terracotta       Frequency          3    Percentage   42.00
***  height    Consistency        0.7815       %Consistency            0.00
G_02_02     ID record    cleaning_1      Value 30.0
G_02_02     ID record    cleaning_2      Value 30.0
G_02_02     ID record    cleaning_3      Value 100.0
G_02_02     ID record    jug             Value 25.0
G_02_02     ID record    milk_cup        Value 15.0
G_02_02     ID record    tea_cup         Value 7.0
G_02_02     ID record    watering_can    Value 50.0
Min    7.00       Max    100.00     Step   23.25
First  Quartile (end)        30.25   Frequency %        71.43
Second Quartile (end)        53.50   Frequency %        14.29
Fourth Quartile (end)       100.00   Frequency %        14.29
***  colour   Consistency        0.7856       %Consistency            0.00
G_02_02     ID record    cleaning_1      Value white
G_02_02     ID record    cleaning_2      Value blue
G_02_02     ID record    cleaning_3      Value severals
G_02_02     ID record    jug             Value white
G_02_02     ID record    milk_cup        Value blue
G_02_02     ID record    tea_cup         Value white
G_02_02     ID record    watering_can    Value green
Value  white            Frequency          3    Percentage   42.00
Value  blue             Frequency          2    Percentage   28.00
Value  severals         Frequency          1    Percentage   14.00
Value  green            Frequency          1    Percentage   14.00
***  weight Consistency      1.1928       %Consistency            0.00
G_02_02     ID record    cleaning_1      Value 50.0
G_02_02     ID record    cleaning_2      Value 60.0
G_02_02     ID record    cleaning_3      Value 110.0
G_02_02     ID record    jug             Value 40.0
G_02_02     ID record    milk_cup        Value 35.0
G_02_02     ID record    tea_cup         Value 30.0
```

```
G_02_02      ID record    watering_can     Value 400.0
Min     30.00       Max     400.00      Step    92.50
First  Quartile (end)        122.50    Frequency %          85.71
Fourth Quartile (end)        400.00    Frequency %          14.29
***  haft    Consistency           0.0000       %Consistency          100.00
G_02_02      ID record    cleaning_1       Value yes
G_02_02      ID record    cleaning_2       Value yes
G_02_02      ID record    cleaning_3       Value yes
G_02_02      ID record    jug              Value yes
G_02_02      ID record    milk_cup         Value yes
G_02_02      ID record    tea_cup          Value yes
G_02_02      ID record    watering_can     Value yes
Value  yes                Frequency           7     Percentage   100.00
***  plug   Consistency      0.1443      %Consistency          71.44
G_02_02      ID record    cleaning_1       Value plastic
G_02_02      ID record    cleaning_2       Value plastic
G_02_02      ID record    cleaning_3       Value plastic
G_02_02      ID record    jug              Value no
G_02_02      ID record    milk_cup         Value no
G_02_02      ID record    tea_cup          Value no
G_02_02      ID record    watering_can     Value no
Value  no                 Frequency           4     Percentage   57.00
Value  plastic            Frequency           3     Percentage   42.00
===========================================================================
*Means* Consistency 0.5145 %Consistency 0 Records     40    %Records   100.00
***  shape       Consistency        0.4357       %Consistency          15.32
***  material    Consistency        0.5283       %Consistency           0.00
***  height      Consistency        0.6182       %Consistency           0.00
***  colour       Consistency       0.3163       %Consistency          38.52
***  weight      Consistency        1.3498       %Consistency           0.00
***  haft        Consistency        0.0000       %Consistency         100.00
***  plug        Consistency        0.3530       %Consistency          31.39
```

Other input files to KB_CAT (*animals.txt*)

The *animals.txt* file is formed by 84 records with 15 variables / columns (Fur, Feather, Eggs, Milk, Flying, Aquatic, Predatory, Teeth, Invertebrate, Lungs, Poisonous, Flippers, Legs, Tail, Domestic).

ANIMAL	FUR	FEATHER	EGGS	MILK	FLYING	AQUATIC	PREDATORY	TEETH	INVERT.	LUNGS	POIS.	FLIP.	LEGS	TAIL	DOM.
SKYLARK	0	1	1	0	1	0	0	0	1	1	0	0	2	1	0
DUCK	0	1	1	0	1	1	0	0	1	1	0	0	2	1	0
ANTELOPE	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
BEE	1	0	1	0	1	0	0	0	0	1	1	0	6	0	1
LOBSTER	0	0	1	0	0	1	1	0	0	0	0	0	6	0	0
HERRING	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
FIELD_MOUSE	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
HAWK	0	1	1	0	1	0	1	0	1	1	0	0	2	1	0
BUFFALO	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
KANGAROO	1	0	0	1	0	0	0	1	1	1	0	0	2	1	0
GOAT	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1
CARP	0	0	1	0	0	1	0	1	1	0	0	1	0	1	1
CHUB	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
CAVY	1	0	0	1	0	0	0	1	1	1	0	0	4	0	1
DEER	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
SWAN	0	1	1	0	1	1	0	0	1	1	0	0	2	1	0
BOAR	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0

ANIMAL	FUR	FEATHER	EGGS	MILK	FLYING	AQUATIC	PREDATORY	TEETH	INVERT.	LUNGS	POIS.	FLIP.	LEGS	TAIL	DOM.
LADYBIRD	0	0	1	0	1	0	1	0	0	1	0	0	6	0	0
DOVE	0	1	1	0	1	0	0	0	1	1	0	0	2	1	1
CROW	0	1	1	0	1	0	1	0	1	1	0	0	2	1	0
HAMSTER	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1
DOLPHIN	0	0	0	1	0	1	1	1	1	1	0	1	0	1	0
CODFISH	0	0	1	0	0	1	0	1	1	0	0	1	0	1	0
ELEPHANT	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
PHEASANT	0	1	1	0	1	0	0	0	1	1	0	0	2	1	0
FALCON	0	1	1	0	1	0	1	0	1	1	0	0	2	1	0
MOTH	1	0	1	0	1	0	0	0	0	1	0	0	6	0	0
FLAMINGO	0	1	1	0	1	0	0	0	1	1	0	0	2	1	0
SEAL	1	0	0	1	0	1	1	1	1	1	0	1	0	0	0
GULL	0	1	1	0	1	1	1	0	1	1	0	0	2	1	0
PRAWN	0	0	1	0	0	1	1	0	0	0	0	0	6	0	0
CHEETAH	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
GIRAFFE	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
GORILLA	1	0	0	1	0	0	0	1	1	1	0	0	2	0	0
CRAB	0	0	1	0	0	1	1	0	0	0	0	0	4	0	0
SEAHORSE	0	0	1	0	0	1	0	1	1	0	0	1	0	1	0
KIWI	0	1	1	0	0	0	1	0	1	1	0	0	2	1	0
LION	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
SEA_LION	1	0	0	1	0	1	1	1	1	1	0	1	2	1	0
LEOPARD	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
HARE	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
SNAIL	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
LYNX	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
PIKE	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
WOLF	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
MONGOOSE	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
CAT	1	0	0	1	0	0	1	1	1	1	0	0	4	1	1
MOLLUSK	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
FLY	1	0	1	0	1	0	0	0	0	1	0	0	6	0	0
MIDGE	0	0	1	0	1	0	0	0	0	1	0	0	6	0	0
OPOSSUM	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
DUCKBILL	1	0	1	1	0	1	1	0	1	1	0	0	4	1	0
BEAR	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0
SPARROW	0	1	1	0	1	0	0	0	1	1	0	0	2	1	0
STURGEON	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
PERCH	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
SHARK	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
PENGUIN	0	1	1	0	0	1	1	0	1	1	0	0	2	1	0
PIRANHA	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
POLYP	0	0	1	0	0	1	1	0	0	0	0	0	8	0	0
CHICKEN	0	1	1	0	1	0	0	0	1	1	0	0	2	1	1
PONY	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1
FLEA	0	0	1	0	0	0	0	0	0	1	0	0	6	0	0
PUMA	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
POLECAT	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
FROG	0	0	1	0	0	1	1	1	1	1	0	0	4	0	0
REINDEER	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1
TOAD	0	0	1	0	0	1	0	1	1	1	0	0	4	0	0
SQUIRREL	1	0	0	1	0	0	0	1	1	1	0	0	2	1	0
SCORPION	0	0	0	0	0	0	1	0	0	1	1	0	8	1	0

ANIMAL	FUR	FEATHER	EGGS	MILK	FLYING	AQUATIC	PREDATORY	TEETH	INVERT.	LUNGS	POIS.	FLIP.	LEGS	TAIL	DOM.
SEA_SNAKE	0	0	0	0	0	1	1	1	1	0	1	0	0	1	0
SOLE	0	0	1	0	0	1	0	1	1	0	0	1	0	1	0
STARFISH	0	0	1	0	0	1	1	0	0	0	0	0	5	0	0
OSTRICH	0	1	1	0	0	0	0	0	1	1	0	0	2	1	0
MOLE	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
TORTOISE	0	0	1	0	0	0	0	0	1	1	0	0	4	1	0
TERMITE	0	0	1	0	0	0	0	0	0	1	0	0	6	0	0
TUNA	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
TRITON	0	0	1	0	0	1	1	1	1	1	0	0	4	1	0
VAMPIRE	1	0	0	1	1	0	0	1	1	1	0	0	2	1	0
WORM	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
WASP	1	0	1	0	1	0	0	0	0	1	1	0	6	0	0
MINK	1	0	0	1	0	1	1	1	1	1	0	0	4	1	0
CALF	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1

Processing of animals.txt file with KB_CAT

The *animals.txt* file has been processed with the following parameters:
Input File -> animals.txt
Number of Groups (3 - 20) -> 4
Normalization (Max, Std, None) -> M
Start Value of alpha (from 1.8 to 0.9) -> 1.8
End Value of alpha (from 0.5 to 0.0001) -> 0.0001
Decreasing step of alpha (from 0.1 to 0.001) -> 0.001
The processing ended with a Minimum Error of 5.255 at the time 971 producing the results in the files:
Output File Catalog.original animals_M_g4_out.txt
Output File Catalog.sort animals_M_g4_outsrt.txt
Output File Summary sort animals_M_g4_sort.txt
Output File Matrix Catal. animals_M_g4_catal.txt
Output File Means, STD, CV. animals_M_g4_medsd.txt
Output File CV of the Groups animals_M_g4_cv.txt
Output File Training Grid animals_M_g4_grid.txt
Output File Run Parameters animals_M_g4_log.txt

Output file/Catalog.sort ordered by group using *animals.txt*

Group	ANIMAL	FUR	FEATH.	EGGS	MILK	FLYING	AQUAT.	PRED.	TEETH	VERT.	LUNGS	POIS.	FLIP.	LEGS	TAIL	DOM.
G_00_00	ANTELOPE	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
G_00_00	BUFFALO	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
G_00_00	CALF	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1
G_00_00	CAT	1	0	0	1	0	0	1	1	1	1	0	0	4	1	1
G_00_00	DEER	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
G_00_00	ELEPHANT	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
G_00_00	FLD_MOUSE	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
G_00_00	GIRAFFE	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
G_00_00	GOAT	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1
G_00_00	HAMSTER	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1
G_00_00	HARE	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0
G_00_00	KANGAROO	1	0	0	1	0	0	0	1	1	1	0	0	2	1	0

Group	ANIMAL	FUR	FEATH.	EGGS	MILK	FLYING	AQUAT.	PRED.	TEETH	VERT.	LUNGS	POIS.	FLIP.	LEGS	TAIL	DOM.
G_00_00	PONY	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1
G_00_00	REINDEER	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1
G_00_00	SQUIRREL	1	0	0	1	0	0	0	1	1	1	0	0	2	1	0
G_00_00	VAMPIRE	1	0	0	1	1	0	0	1	1	1	0	0	2	1	0
G_00_01	CAVY	1	0	0	1	0	0	0	1	1	1	0	0	4	0	1
G_00_01	GORILLA	1	0	0	1	0	0	0	1	1	1	0	0	2	0	0
G_00_02	BEE	1	0	1	0	1	0	0	0	0	1	1	0	6	0	1
G_00_03	CRAB	0	0	1	0	0	1	1	0	0	0	0	0	4	0	0
G_00_03	FLY	1	0	1	0	1	0	0	0	0	1	0	0	6	0	0
G_00_03	LADYBIRD	0	0	1	0	1	0	1	0	0	1	0	0	6	0	0
G_00_03	LOBSTER	0	0	1	0	0	1	1	0	0	0	0	0	6	0	0
G_00_03	MIDGE	0	0	1	0	1	0	0	0	0	1	0	0	6	0	0
G_00_03	MOLLUSK	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
G_00_03	MOTH	1	0	1	0	1	0	0	0	0	1	0	0	6	0	0
G_00_03	POLYP	0	0	1	0	0	1	1	0	0	0	0	0	8	0	0
G_00_03	PRAWN	0	0	1	0	0	1	1	0	0	0	0	0	6	0	0
G_00_03	STARFISH	0	0	1	0	0	1	1	0	0	0	0	0	5	0	0
G_00_03	WASP	1	0	1	0	1	0	0	0	0	1	1	0	6	0	0
G_01_00	BEAR	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0
G_01_00	BOAR	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
G_01_00	CHEETAH	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
G_01_00	LEOPARD	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
G_01_00	LION	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
G_01_00	LYNX	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
G_01_00	MINK	1	0	0	1	0	1	1	1	1	1	0	0	4	1	0
G_01_00	MOLE	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
G_01_00	MONGOOSE	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
G_01_00	OPOSSUM	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
G_01_00	POLECAT	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
G_01_00	PUMA	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
G_01_00	WOLF	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0
G_01_02	SCORPION	0	0	0	0	0	0	1	0	0	1	1	0	8	1	0
G_01_03	FLEA	0	0	1	0	0	0	0	0	0	1	0	0	6	0	0
G_01_03	SNAIL	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
G_01_03	TERMITE	0	0	1	0	0	0	0	0	0	1	0	0	6	0	0
G_01_03	WORM	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
G_02_00	DOLPHIN	0	0	0	1	0	1	1	1	1	1	0	1	0	1	0
G_02_00	SEAL	1	0	0	1	0	1	1	1	1	1	0	1	0	0	0
G_02_00	SEA_LION	1	0	0	1	0	1	1	1	1	1	0	1	2	1	0
G_02_01	DUCKBILL	1	0	1	1	0	1	1	0	1	1	0	0	4	1	0
G_02_02	TOAD	0	0	1	0	0	1	0	1	1	1	0	0	4	0	0
G_02_02	TORTOISE	0	0	1	0	0	0	0	0	1	1	0	0	4	1	0
G_03_00	CARP	0	0	1	0	0	1	0	1	1	0	0	1	0	1	1
G_03_00	CHUB	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
G_03_00	CODFISH	0	0	1	0	0	1	0	1	1	0	0	1	0	1	0
G_03_00	HERRING	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
G_03_00	PERCH	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
G_03_00	PIKE	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
G_03_00	PIRANHA	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
G_03_00	SEAHORSE	0	0	1	0	0	1	0	1	1	0	0	1	0	1	0
G_03_00	SEA_SNAKE	0	0	0	0	0	1	1	1	1	0	1	0	0	1	0
G_03_00	SHARK	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
G_03_00	SOLE	0	0	1	0	0	1	0	1	1	0	0	1	0	1	0

Group	ANIMAL	FUR	FEATH.	EGGS	MILK	FLYING	AQUAT.	PRED.	TEETH	VERT.	LUNGS	POIS.	FLIP.	LEGS	TAIL	DOM.
G_03_00	STURGEON	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
G_03_00	TUNA	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0
G_03_01	FROG	0	0	1	0	0	1	1	1	1	1	0	0	4	0	0
G_03_01	TRITON	0	0	1	0	0	1	1	1	1	1	0	0	4	1	0
G_03_02	GULL	0	1	1	0	1	1	1	0	1	1	0	0	2	1	0
G_03_02	KIWI	0	1	1	0	0	0	1	0	1	1	0	0	2	1	0
G_03_02	PENGUIN	0	1	1	0	0	1	1	0	1	1	0	0	2	1	0
G_03_03	CHICKEN	0	1	1	0	1	0	0	0	1	1	0	0	2	1	1
G_03_03	CROW	0	1	1	0	1	0	1	0	1	1	0	0	2	1	0
G_03_03	DOVE	0	1	1	0	1	0	0	0	1	1	0	0	2	1	1
G_03_03	DUCK	0	1	1	0	1	1	0	0	1	1	0	0	2	1	0
G_03_03	FALCON	0	1	1	0	1	0	1	0	1	1	0	0	2	1	0
G_03_03	FLAMINGO	0	1	1	0	1	0	0	0	1	1	0	0	2	1	0
G_03_03	HAWK	0	1	1	0	1	0	1	0	1	1	0	0	2	1	0
G_03_03	OSTRICH	0	1	1	0	0	0	0	0	1	1	0	0	2	1	0
G_03_03	PHEASANT	0	1	1	0	1	0	0	0	1	1	0	0	2	1	0
G_03_03	SKYLARK	0	1	1	0	1	0	0	0	1	1	0	0	2	1	0
G_03_03	SPARROW	0	1	1	0	1	0	0	0	1	1	0	0	2	1	0
G_03_03	SWAN	0	1	1	0	1	1	0	0	1	1	0	0	2	1	0

Output file/CV ordered by group using *animals.txt*

Groups	FUR	FEATHER	EGGS	MILK	FLYING	AQUATIC	PRED.	TEETH	VERT.	LUNGS	POIS.	FLIP.	LEGS	TAIL	DOMEST.	*Mean*	N_recs
G_00_00	0	0	0	0	3,87	0	3,87	0	0	0	0	0	0,22	0	1,29	0,62	16
G_00_01	0	0	0	0	0	0	0	0	0	0	0	0	0,33	0	1	0,09	2
G_00_02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
G_00_03	1,63	0	0	0	1,1	1,1	0,76	0	0	1,1	3,16	0	0,36	0	0	0,61	11
G_01_00	0	0	0	0	0	3,46	0	0	0	0	0	0	0	0,29	0	0,25	13
G_01_02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
G_01_03	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0,07	4
G_02_00	0,71	0	0	0	0	0	0	0	0	0	0	0	1,41	0,71	0	0,19	3
G_02_01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
G_02_02	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0,2	2
G_03_00	0	0	0,29	0	0	0	0,67	0	0	0	3,46	0,29	0	0	3,46	0,54	13
G_03_01	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0,07	2
G_03_02	0	0	0	0	1,41	0,71	0	0	0	0	0	0	0	0	0	0,14	3
G_03_03	0	0	0	0	0,3	2,24	1,73	0	0	0	0	0	0	0	2,24	0,43	12
Means	0,24	0	0,04	0	0,97	1,05	1,19	0,02	0	0,14	0,95	0,04	0,19	0,12	1,13	0,41	84
Total	2,69	5,25	2,12	2,9	4,53	3,29	2,39	2,02	1,23	1,32	10,95	5,25	1,68	1,46	6,31	3,57	84

In this example you can see the importance of the difference of the CV of the whole file (CV_Pop 3,57) before the cataloging as opposed to the means of the CV of the groups (CV_Med 0,41).
The cataloging has produced an improvement of 88,5%.
Further confirmation derives from the presence of many zero values in the cells in the table, values which indicate that in that variable / column of the group there is just one value that is constantly repeated: the variable / column (for that constant value) is certainly important for the cataloging of records in that group.

Verify the validity of a manual cataloging

Sometimes it can be useful to certify the validity of a manual cataloging using KB_CAT.
In the example described below the variable / column D1 contains the input code of

the animal specie (bird, mammal, insect, fish, ...); the variable / column was then duplicated in another two columns (D2 e D3).
The modification of the original file has two different consequences:

- it reinforces the importance of the code of the animal specie in respect to all the other variables / columns
- the new structure of the file allows KB_CAT to process the data in a similar way to algorithms typical of supervised networks with different input variables / columns and a variable / column objective (in our case D1, D2, D3 considered globally)

Input file to KB_CAT (animals_*d.txt*)

ANIMAL	FUR	FEATHER	EGGS	MILK	XXX	D1	D2	D3
SKYLARK	0	1	1	0		bird	bird	Bird
DUCK	0	1	1	0		bird	bird	Bird
ANTELOPE	1	0	0	1		mammal	mammal	Mammal
BEE	1	0	1	0		insect	insect	Insect
LOBSTER	0	0	1	0		shellfish	shellfish	Shellfish
HERRING	0	0	1	0		fish	fish	Fish
FIELD_MOUSE	1	0	0	1		mammal	mammal	Mammal
HAWK	0	1	1	0		bird_of_prey	bird_of_prey	bird_of_prey
BUFFALO	1	0	0	1		mammal	mammal	Mammal
KANGAROO	1	0	0	1		mammal	mammal	Mammal
GOAT	1	0	0	1		mammal	mammal	Mammal
CARP	0	0	1	0		fish	fish	Fish
CHUB	0	0	1	0		fish	fish	Fish
CAVY	1	0	0	1		mammal	mammal	Mammal
DEER	1	0	0	1		mammal	mammal	Mammal
SWAN	0	1	1	0		bird	bird	Bird
BOAR	1	0	0	1		mammal	mammal	Mammal
LADYBIRD	0	0	1	0		insect	insect	Insect
DOVE	0	1	1	0		bird	bird	Bird
CROW	0	1	1	0		bird	bird	Bird
HAMSTER	1	0	0	1		mammal	mammal	Mammal
DOLPHIN	0	0	0	1		mammal	mammal	Mammal
CODFISH	0	0	1	0		fish	fish	Fish
ELEPHANT	1	0	0	1		mammal	mammal	Mammal
PHEASANT	0	1	1	0		bird	bird	Bird
FALCON	0	1	1	0		bird_of_prey	bird_of_prey	bird_of_prey
MOTH	1	0	1	0		insect	insect	Insect
FLAMINGO	0	1	1	0		bird	bird	Bird
SEAL	1	0	0	1		mammal	mammal	Mammal
GULL	0	1	1	0		bird	bird	Bird
PRAWN	0	0	1	0		shellfish	shellfish	Shellfish
CHEETAH	1	0	0	1		mammal	mammal	Mammal
GIRAFFE	1	0	0	1		mammal	mammal	Mammal
GORILLA	1	0	0	1		mammal	mammal	Mammal
CRAB	0	0	1	0		shellfish	shellfish	Shellfish
SEAHORSE	0	0	1	0		fish	fish	Fish

ANIMAL	FUR	FEATHER	EGGS	MILK	XXX	D1	D2	D3
KIWI	0	1	1	0		bird	bird	Bird
LION	1	0	0	1		mammal	mammal	Mammal
SEA_LION	1	0	0	1		mammal	mammal	Mammal
LEOPARD	1	0	0	1		mammal	mammal	Mammal
HARE	1	0	0	1		mammal	mammal	Mammal
SNAIL	0	0	1	0		invertebrate	invertebrate	Invertebrate
LYNX	1	0	0	1		mammal	mammal	Mammal
PIKE	0	0	1	0		fish	fish	Fish
WOLF	1	0	0	1		mammal	mammal	Mammal
MONGOOSE	1	0	0	1		mammal	mammal	Mammal
CAT	1	0	0	1		mammal	mammal	Mammal
MOLLUSK	0	0	1	0		invertebrate	invertebrate	Invertebrate
FLY	1	0	1	0		insect	insect	Insect
MIDGE	0	0	1	0		insect	insect	Insect
OPOSSUM	1	0	0	1		mammal	mammal	Mammal
DUCKBILL	1	0	1	1		mammal	mammal	Mammal
BEAR	1	0	0	1		mammal	mammal	Mammal
SPARROW	0	1	1	0		bird	bird	Bird
STURGEON	0	0	1	0		fish	fish	Fish
PERCH	0	0	1	0		fish	fish	Fish
SHARK	0	0	1	0		fish	fish	Fish
PENGUIN	0	1	1	0		bird	bird	Bird
PIRANHA	0	0	1	0		fish	fish	Fish
POLYP	0	0	1	0		invertebrate	invertebrate	Invertebrate
CHICKEN	0	1	1	0		bird	bird	Bird
PONY	1	0	0	1		mammal	mammal	Mammal
FLEA	0	0	1	0		insect	insect	Insect
PUMA	1	0	0	1		mammal	mammal	Mammal
POLECAT	1	0	0	1		mammal	mammal	Mammal
FROG	0	0	1	0		amphibian	amphibian	Amphibian
REINDEER	1	0	0	1		mammal	mammal	Mammal
TOAD	0	0	1	0		amphibian	amphibian	Amphibian
SQUIRREL	1	0	0	1		mammal	mammal	Mammal
SCORPION	0	0	0	0		arachinida	arachinida	Arachinida
SEA_SNAKE	0	0	0	0		reptiles	reptiles	Reptiles
SOLE	0	0	1	0		fish	fish	Fish
STARFISH	0	0	1	0		echinoderm	echinoderm	Echinoderm
OSTRICH	0	1	1	0		bird	bird	Bird
MOLE	1	0	0	1		mammal	mammal	Mammal
TORTOISE	0	0	1	0		reptiles	reptiles	Reptiles
TERMITE	0	0	1	0		insect	insect	Insect
TUNA	0	0	1	0		fish	fish	Fish
TRITON	0	0	1	0		amphibian	amphibian	Amphibian
VAMPIRE	1	0	0	1		mammal	mammal	Mammal
WORM	0	0	1	0		invertebrate	invertebrate	Invertebrate
WASP	1	0	1	0		insect	insect	Insect
MINK	1	0	0	1		mammal	mammal	Mammal

ANIMAL	FUR	FEATHER	EGGS	MILK	XXX	D1	D2	D3
CALF	1	0	0	1		mammal	mammal	Mammal

The processing was carried out with the following parameters:

```
###############################################################################
# KB_CAT KNOWLEDGE DISCOVERY IN DATA MINING (CATALOG PROGRAM)                 #
# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                 #
# Language used: PYTHON                                                       #
###############################################################################
Input File                                         -> animals_d.txt
Number of Groups (3 - 20)                          -> 4
Normalization (Max, Std, None)                     -> M
Start Value of alpha (from 1.8 to 0.9)             -> 1.8
End Value of alpha (from 0.5 to 0.001)             -> 0.0001
Decreasing step of alpha (from 0.1 to 0.001)       -> 0.001
=========================OUTPUT===============================================
Output File Catalog.original     animals_d_M_g4_out.txt
Output File Catalog.sort         animals_d_M_g4_outsrt.txt
Output File Summary sort         animals_d_M_g4_sort.txt
Output File Matrix Catal.        animals_d_M_g4_catal.txt
Output File Means, STD, CV.      animals_d_M_g4_medsd.txt
Output File CV of the Groups     animals_d_M_g4_cv.txt
Output File Training Grid        animals_d_M_g4_grid.txt
Output File Run Parameters       animals_d_M_g4_log.txt
```

Results obtained processing animals_d.txt (*Output/Catalog.sort*)

Group	ANIMAL	FUR	FEATHER	EGGS	MILK	XXX	D1	D2	D3
G_00_00	BEE	1.0	0.0	1.0	0.0		insect	insect	Insect
G_00_00	CRAB	0.0	0.0	1.0	0.0		shellfish	shellfish	Shellfish
G_00_00	FLY	1.0	0.0	1.0	0.0		insect	insect	Insect
G_00_00	LADYBIRD	0.0	0.0	1.0	0.0		insect	insect	Insect
G_00_00	LOBSTER	0.0	0.0	1.0	0.0		shellfish	shellfish	Shellfish
G_00_00	MIDGE	0.0	0.0	1.0	0.0		insect	insect	Insect
G_00_00	MOLLUSK	0.0	0.0	1.0	0.0		invertebrate	invertebrate	Invertebrate
G_00_00	MOTH	1.0	0.0	1.0	0.0		insect	insect	Insect
G_00_00	POLYP	0.0	0.0	1.0	0.0		invertebrate	invertebrate	Invertebrate
G_00_00	PRAWN	0.0	0.0	1.0	0.0		shellfish	shellfish	Shellfish
G_00_00	SNAIL	0.0	0.0	1.0	0.0		invertebrate	invertebrate	Invertebrate
G_00_00	WASP	1.0	0.0	1.0	0.0		insect	insect	Insect
G_00_00	WORM	0.0	0.0	1.0	0.0		invertebrate	invertebrate	Invertebrate
G_00_01	FLEA	0.0	0.0	1.0	0.0		insect	insect	Insect
G_00_01	STARFISH	0.0	0.0	1.0	0.0		echinoderm	echinoderm	Echinoderm
G_00_01	TERMITE	0.0	0.0	1.0	0.0		insect	insect	Insect
G_00_03	CHICKEN	0.0	1.0	1.0	0.0		bird	bird	bird
G_00_03	CROW	0.0	1.0	1.0	0.0		bird	bird	bird
G_00_03	DOVE	0.0	1.0	1.0	0.0		bird	bird	bird
G_00_03	DUCK	0.0	1.0	1.0	0.0		bird	bird	bird
G_00_03	FALCON	0.0	1.0	1.0	0.0		bird_of_prey	bird_of_prey	bird_of_prey
G_00_03	FLAMINGO	0.0	1.0	1.0	0.0		bird	bird	bird
G_00_03	HAWK	0.0	1.0	1.0	0.0		bird_of_prey	bird_of_prey	bird_of_prey
G_00_03	OSTRICH	0.0	1.0	1.0	0.0		bird	bird	bird
G_00_03	PHEASANT	0.0	1.0	1.0	0.0		bird	bird	bird
G_00_03	SKYLARK	0.0	1.0	1.0	0.0		bird	bird	bird
G_00_03	SPARROW	0.0	1.0	1.0	0.0		bird	bird	bird
G_00_03	SWAN	0.0	1.0	1.0	0.0		bird	bird	bird
G_01_01	TORTOISE	0.0	0.0	1.0	0.0		reptiles	reptiles	reptiles
G_01_02	SCORPION	0.0	0.0	0.0	0.0		arachinida	arachinida	arachinida
G_01_02	TOAD	0.0	0.0	1.0	0.0		amphibian	amphibian	amphibian
G_01_03	GULL	0.0	1.0	1.0	0.0		bird	bird	bird
G_01_03	KIWI	0.0	1.0	1.0	0.0		bird	bird	bird
G_01_03	PENGUIN	0.0	1.0	1.0	0.0		bird	bird	bird

Group	ANIMAL	FUR	FEATHER	EGGS	MILK	XXX	D1	D2	D3
G_02_00	ANTELOPE	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_02_00	BUFFALO	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_02_00	DEER	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_02_00	ELEPHANT	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_02_00	FIELD_MOUSE	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_02_00	GIRAFFE	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_02_00	GORILLA	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_02_00	HARE	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_02_00	KANGAROO	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_02_00	SQUIRREL	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_02_00	VAMPIRE	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_02_02	DUCKBILL	1.0	0.0	1.0	1.0		mammal	mammal	mammal
G_02_03	FROG	0.0	0.0	1.0	0.0		amphibian	amphibian	amphibian
G_02_03	TRITON	0.0	0.0	1.0	0.0		amphibian	amphibian	amphibian
G_03_00	CALF	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_00	CAT	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_00	CAVY	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_00	GOAT	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_00	HAMSTER	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_00	PONY	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_00	REINDEER	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_01	BEAR	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_01	BOAR	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_01	CHEETAH	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_01	LEOPARD	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_01	LION	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_01	LYNX	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_01	MINK	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_01	MOLE	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_01	MONGOOSE	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_01	OPOSSUM	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_01	POLECAT	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_01	PUMA	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_01	WOLF	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_02	DOLPHIN	0.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_02	SEAL	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_02	SEA_LION	1.0	0.0	0.0	1.0		mammal	mammal	mammal
G_03_02	SEA_SNAKE	0.0	0.0	0.0	0.0		reptiles	reptiles	reptiles
G_03_03	CARP	0.0	0.0	1.0	0.0		fish	fish	fish
G_03_03	CHUB	0.0	0.0	1.0	0.0		fish	fish	fish
G_03_03	CODFISH	0.0	0.0	1.0	0.0		fish	fish	fish
G_03_03	HERRING	0.0	0.0	1.0	0.0		fish	fish	fish
G_03_03	PERCH	0.0	0.0	1.0	0.0		fish	fish	fish
G_03_03	PIKE	0.0	0.0	1.0	0.0		fish	fish	fish
G_03_03	PIRANHA	0.0	0.0	1.0	0.0		fish	fish	fish
G_03_03	SEAHORSE	0.0	0.0	1.0	0.0		fish	fish	fish
G_03_03	SHARK	0.0	0.0	1.0	0.0		fish	fish	fish
G_03_03	SOLE	0.0	0.0	1.0	0.0		fish	fish	fish
G_03_03	STURGEON	0.0	0.0	1.0	0.0		fish	fish	fish
G_03_03	TUNA	0.0	0.0	1.0	0.0		fish	fish	fish

The manual cataloging is confirmed except for the coloured records of the groups G_00_00, G_00_01, G_01_02 and G_03_02.

The questions that the researchers could ask in analogue cases, but which are much more complicated and important in real life of companies and organisations, could be:

- do errors occur during the gathering of data?
- do errors exist while inserting data?
- do mutations occur inside the groups?
- are there defects in production / working?
- Is there a lack in the design of the manual classification?
- Is it necessary to introduce other variables / columns?

Comparison of the results of the automatic cataloging of *iris.txt* to those recognized by botanists

The reliability of the algorithms of neural networks is often appreciated by the coincidence between the automatic cataloging of files *iris.txt* (150 records) and that carried out by botanists.
KB_CAT with a cataloging into 3 groups was made with the following results.

Group	RecId	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
G_00_00	versicolor100	5.7	2.8	4.1	1.3
G_00_00	versicolor54	5.5	2.3	4.0	1.3
G_00_00	versicolor56	5.7	2.8	4.5	1.3
G_00_00	versicolor60	5.2	2.7	3.9	1.4
G_00_00	versicolor63	6.0	2.2	4.0	1.0
G_00_00	versicolor69	6.2	2.2	4.5	1.5
G_00_00	versicolor70	5.6	2.5	3.9	1.1
G_00_00	versicolor72	6.1	2.8	4.0	1.3
G_00_00	versicolor73	6.3	2.5	4.9	1.5
G_00_00	versicolor81	5.5	2.4	3.8	1.1
G_00_00	versicolor83	5.8	2.7	3.9	1.2
G_00_00	versicolor85	5.4	3.0	4.5	1.5
G_00_00	versicolor88	6.3	2.3	4.4	1.3
G_00_00	versicolor90	5.5	2.5	4.0	1.3
G_00_00	versicolor91	5.5	2.6	4.4	1.2
G_00_00	versicolor93	5.8	2.6	4.0	1.2
G_00_00	versicolor95	5.6	2.7	4.2	1.3
G_00_00	versicolor97	5.7	2.9	4.2	1.3
G_00_00	virginica107	4.9	2.5	4.5	1.7
G_00_00	virginica120	6.0	2.2	5.0	1.5
G_00_01	versicolor84	6.0	2.7	5.1	1.6
G_00_01	virginica102	5.8	2.7	5.1	1.9
G_00_01	virginica112	6.4	2.7	5.3	1.9
G_00_01	virginica114	5.7	2.5	5.0	2.0
G_00_01	virginica122	5.6	2.8	4.9	2.0
G_00_01	virginica124	6.3	2.7	4.9	1.8
G_00_01	virginica127	6.2	2.8	4.8	1.8
G_00_01	virginica128	6.1	3.0	4.9	1.8
G_00_01	virginica135	6.1	2.6	5.6	1.4
G_00_01	virginica139	6.0	3.0	4.8	1.8
G_00_01	virginica143	5.8	2.7	5.1	1.9
G_00_01	virginica147	6.3	2.5	5.0	1.9
G_00_01	virginica150	5.9	3.0	5.1	1.8
G_00_02	virginica101	6.3	3.3	6.0	2.5
G_00_02	virginica103	7.1	3.0	5.9	2.1
G_00_02	virginica105	6.5	3.0	5.8	2.2
G_00_02	virginica106	7.6	3.0	6.6	2.1
G_00_02	virginica108	7.3	2.9	6.3	1.8

Group	RecId	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
G_00_02	virginica109	6.7	2.5	5.8	1.8
G_00_02	virginica110	7.2	3.6	6.1	2.5
G_00_02	virginica113	6.8	3.0	5.5	2.1
G_00_02	virginica115	5.8	2.8	5.1	2.4
G_00_02	virginica116	6.4	3.2	5.3	2.3
G_00_02	virginica118	7.7	3.8	6.7	2.2
G_00_02	virginica119	7.7	2.6	6.9	2.3
G_00_02	virginica121	6.9	3.2	5.7	2.3
G_00_02	virginica123	7.7	2.8	6.7	2.0
G_00_02	virginica125	6.7	3.3	5.7	2.1
G_00_02	virginica126	7.2	3.2	6.0	1.8
G_00_02	virginica129	6.4	2.8	5.6	2.1
G_00_02	virginica131	7.4	2.8	6.1	1.9
G_00_02	virginica132	7.9	3.8	6.4	2.0
G_00_02	virginica133	6.4	2.8	5.6	2.2
G_00_02	virginica136	7.7	3.0	6.1	2.3
G_00_02	virginica137	6.3	3.4	5.6	2.4
G_00_02	virginica140	6.9	3.1	5.4	2.1
G_00_02	virginica141	6.7	3.1	5.6	2.4
G_00_02	virginica142	6.9	3.1	5.1	2.3
G_00_02	virginica144	6.8	3.2	5.9	2.3
G_00_02	virginica145	6.7	3.3	5.7	2.5
G_00_02	virginica146	6.7	3.0	5.2	2.3
G_00_02	virginica148	6.5	3.0	5.2	2.0
G_00_02	virginica149	6.2	3.4	5.4	2.3
G_01_00	versicolor58	4.9	2.4	3.3	1.0
G_01_00	versicolor61	5.0	2.0	3.5	1.0
G_01_00	versicolor65	5.6	2.9	3.6	1.3
G_01_00	versicolor68	5.8	2.7	4.1	1.0
G_01_00	versicolor80	5.7	2.6	3.5	1.0
G_01_00	versicolor82	5.5	2.4	3.7	1.0
G_01_00	versicolor89	5.6	3.0	4.1	1.3
G_01_00	versicolor94	5.0	2.3	3.3	1.0
G_01_00	versicolor96	5.7	3.0	4.2	1.2
G_01_00	versicolor99	5.1	2.5	3.0	1.1
G_01_01	versicolor62	5.9	3.0	4.2	1.5
G_01_01	versicolor64	6.1	2.9	4.7	1.4
G_01_01	versicolor67	5.6	3.0	4.5	1.5
G_01_01	versicolor71	5.9	3.2	4.8	1.8
G_01_01	versicolor79	6.0	2.9	4.5	1.5
G_01_01	versicolor86	6.0	3.4	4.5	1.6
G_01_01	versicolor92	6.1	3.0	4.6	1.4
G_01_02	versicolor78	6.7	3.0	5.0	1.7
G_01_02	virginica104	6.3	2.9	5.6	1.8
G_01_02	virginica111	6.5	3.2	5.1	2.0
G_01_02	virginica117	6.5	3.0	5.5	1.8
G_01_02	virginica130	7.2	3.0	5.8	1.6
G_01_02	virginica138	6.4	3.1	5.5	1.8
G_02_00	setosa1	5.1	3.5	1.4	0.2
G_02_00	setosa10	4.9	3.1	1.5	0.1
G_02_00	setosa11	5.4	3.7	1.5	0.2
G_02_00	setosa12	4.8	3.4	1.6	0.2
G_02_00	setosa13	4.8	3.0	1.4	0.1
G_02_00	setosa14	4.3	3.0	1.1	0.1
G_02_00	setosa15	5.8	4.0	1.2	0.2
G_02_00	setosa16	5.7	4.4	1.5	0.4
G_02_00	setosa17	5.4	3.9	1.3	0.4

Group	RecId	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
G_02_00	setosa18	5.1	3.5	1.4	0.3
G_02_00	setosa19	5.7	3.8	1.7	0.3
G_02_00	setosa2	4.9	3.0	1.4	0.2
G_02_00	setosa20	5.1	3.8	1.5	0.3
G_02_00	setosa21	5.4	3.4	1.7	0.2
G_02_00	setosa22	5.1	3.7	1.5	0.4
G_02_00	setosa23	4.6	3.6	1.0	0.2
G_02_00	setosa24	5.1	3.3	1.7	0.5
G_02_00	setosa25	4.8	3.4	1.9	0.2
G_02_00	setosa26	5.0	3.0	1.6	0.2
G_02_00	setosa27	5.0	3.4	1.6	0.4
G_02_00	setosa28	5.2	3.5	1.5	0.2
G_02_00	setosa29	5.2	3.4	1.4	0.2
G_02_00	setosa3	4.7	3.2	1.3	0.2
G_02_00	setosa30	4.7	3.2	1.6	0.2
G_02_00	setosa31	4.8	3.1	1.6	0.2
G_02_00	setosa32	5.4	3.4	1.5	0.4
G_02_00	setosa33	5.2	4.1	1.5	0.1
G_02_00	setosa34	5.5	4.2	1.4	0.2
G_02_00	setosa35	4.9	3.1	1.5	0.2
G_02_00	setosa36	5.0	3.2	1.2	0.2
G_02_00	setosa37	5.5	3.5	1.3	0.2
G_02_00	setosa38	4.9	3.6	1.4	0.1
G_02_00	setosa39	4.4	3.0	1.3	0.2
G_02_00	setosa4	4.6	3.1	1.5	0.2
G_02_00	setosa40	5.1	3.4	1.5	0.2
G_02_00	setosa41	5.0	3.5	1.3	0.3
G_02_00	setosa42	4.5	2.3	1.3	0.3
G_02_00	setosa43	4.4	3.2	1.3	0.2
G_02_00	setosa44	5.0	3.5	1.6	0.6
G_02_00	setosa45	5.1	3.8	1.9	0.4
G_02_00	setosa46	4.8	3.0	1.4	0.3
G_02_00	setosa47	5.1	3.8	1.6	0.2
G_02_00	setosa48	4.6	3.2	1.4	0.2
G_02_00	setosa49	5.3	3.7	1.5	0.2
G_02_00	setosa5	5.0	3.6	1.4	0.2
G_02_00	setosa50	5.0	3.3	1.4	0.2
G_02_00	setosa6	5.4	3.9	1.7	0.4
G_02_00	setosa7	4.6	3.4	1.4	0.3
G_02_00	setosa8	5.0	3.4	1.5	0.2
G_02_00	setosa9	4.4	2.9	1.4	0.2
G_02_02	versicolor51	7.0	3.2	4.7	1.4
G_02_02	versicolor52	6.4	3.2	4.5	1.5
G_02_02	versicolor53	6.9	3.1	4.9	1.5
G_02_02	versicolor55	6.5	2.8	4.6	1.5
G_02_02	versicolor57	6.3	3.3	4.7	1.6
G_02_02	versicolor59	6.6	2.9	4.6	1.3
G_02_02	versicolor66	6.7	3.1	4.4	1.4
G_02_02	versicolor74	6.1	2.8	4.7	1.2
G_02_02	versicolor75	6.4	2.9	4.3	1.3
G_02_02	versicolor76	6.6	3.0	4.4	1.4
G_02_02	versicolor77	6.8	2.8	4.8	1.4
G_02_02	versicolor87	6.7	3.1	4.7	1.5
G_02_02	versicolor98	6.2	2.9	4.3	1.3
G_02_02	virginica134	6.3	2.8	5.1	1.5

With the exception of records highlighted in yellow, the automatic cataloging has confirmed the botanists's opinion (inserted in the column *RecId*), reaching a high value of the *Knowledge Index* (0.9311).

Clinical trials on hepatitis B virus

In 2006 KB_CAT was used to process data from an important research on clinical trials regarding 1414 subjects with 33 variables / columns.
The research concerned the hepatitis B virus, the characteristics of typical carriers, of symptomless carriers, those with low viral repetition, the possible evolution of the virus in other pathologies, the identification *marker*, the diagnosis and the treatment.
The problem of identification consists in determining whether a person has the characteristics enough to associate him with a group of carriers of the virus.
The variables / columns that concern the generalities of the subjects, such as age, race, weight, height, the area of birth, and residency were omitted as they resulted to be of little importance in previous runs.
The weight and height of the subjects would be misleading if adopted separately. For this reason it was calculated the body mass index (*BMI*) that connects the two attributes, through the relationship

$$BMI = kg /(m)^2$$

In addition, it was also calculated an index, particularly significant that correlates weight, height, gender and age of the subject. The real index of weight, which also takes account of the muscles and the body of the subjects is in fact the percentage of fat mass (*FAT*). There are different formulas, very similar to each other, to calculate this index. In this case it was considered the formula of *Deurenberg* that, most of the other, takes into account the age of the subject:

$$FAT(\%)= (1,2 * BMI) + (0,23 * age) – (10,8 * gender) – 5,4$$
$$gender\ 1 = men,\ gender\ 2 = women$$

Regarding the variables related to the potus, indicating the number of glasses of wine / beer / spirits drunk daily and how long the subjects were drinking, were calculated the alcohol units as the product of the two values. Consequently, it was eliminated field indicating whether or not the subject was a teetotaler.

The input file, after the revision, it is shown in the following table:

Generalities	Age, Gender, BMI, FAT, Case
Potus	Total-UA (Total alcoholic Units)
Diagnosis	Diagnosis, Steatohepatitis, Steatosis
Therapy	PreInterferone, PreLamivudina , PreAdefovir , PrePegInterferone, PreEntecavir, PreTecnofovir, Interferone, Peginterferone, Lamivudina, Adefovir, Tecnofovir

Laboratory trials	AST, ALT, HBeAg, AntiHBe, AntiHBclgM, HBVDNAqualitative, HBVDNAquantitative, GentipoHBV, AntiDelta, HIV, AntiHCV, HCVRNAQualitative, GentipoHCV

The most significant characteristics of the more numerous groups are contained in the following table.

Group	M/F	FAT %	Case	UA	Diagnosis	Adefo-vir	AST/ALT	HBeAg	AntiHBclgM
1_01	M	11-43	Prevalent	73000	Chronic Hepatitis	No	high	Negative	Negative
1_03	M	1743	Prevalent	29200	Chronic Hepatitis	No	medium	Negative	Not researched
2_01	M	1636	Prevalent	45000	Carrier in non repeating phase	No	normal	Negative	Negative
2_08	F	2356	Prevalent	9000	Chronic Hepatitis and carrier in non repeating phase	No	normal	Negative	Negative
3_01	M	2837	Prevalent	282800	HCC and Cirrhosis	No	high	Negative	Negative
3_04	M	1434	Incidental	64000	Chronic Hepatitis and carrier in non repeating phase	No	high	Negative	Not researched
4_01	M	2838	Prevalent	73000	HCC and Cirrhosis	No	high	Negative	Positive / not researched
8_01	M	01/12/47	Prevalent	45600	Chronic Hepatitis	No	high	Positive	Negative
8_08	M	1737	Prevalent	73000	Cirrhosis	yes	medium	Negative	Negative

From a comprehensive analysis of the results obtained, not published in detail, the following conclusions have emerged:

- women drink less than men, and consequently suffer from hepatitis rather than from cirrhosis
- only men over the age of 50 years show a diagnosis of HCC, such persons do not have steatosis and the steatohepatitis positive does not take a very high level
- carriers of the virus in the *non repetitive phase* show normal values in AST and ALT laboratory exams
- HIV positive are almost exclusively men with chronic hepatitis
- the diagnosis of *cirrhosis* is only present in subjects who are over the age of 40 years
- the carriers in *non repetitive phase* have a percentage of *incident cases* greater than the other diagnoses
- subjects who have been diagnosed *HCC* are nearly always *prevalent cases*
- the highest values *ALT and AST* regard the subjects who have been diagnosed with *cirrhosis* or *HCC*
- nearly all the women in the *non repetitive phase* result teetotaler: this is not the case for the men
- the lowest values of FAT concerns men who have been diagnosed with chronic hepatitis and show values of units of alcohol reduced

Not good, but better! (2006 mail)

Dear Roberto,
it really does seem that your software is a winner.
As I anticipated, I have sent the conclusions of the E. to P. A. (hepatologist

and top-publisher, who is native of Milan but works in Palermo) also because he was one of the owners of the database which was used and especially because it was him, with his data who carried out the more detailed clinical/statistical analysis.
He phoned me this morning asking which doctor had contributed with analysing the database and writing the report. When I explained that neither you nor E. Are doctors and that the conclusions were drawn with your software, he didn't want to believe me.
All (and I mean ALL) the conclusions are correct and coincide with those taken from statistical analysis and from the clinical observations.
I explained that the software is in the process of being verified and that his collaboration would be useful in this.
As it would be a case of verifying that the conclusions of the software are parallel to those of the clinical/statistical analysis and in other databases and once this has been done, an informative/clinical publication could be released to verify your application at least in this field.
He would be pleased to collaborate with you in this way and will start by sending you one or two clinical databases which have already been analysed (from 800 to 8.000 cases) on which the small KB will be tested. A big pat on the back and congratulations!

KB_STA – the statistical analysis of the result of the cataloging

Generalities, aims and functions

The aim of KB_STA is to help researchers in the analysis of the results of processing.
KB_STA has proven to be indispensable when the input file is of a large size, either in the amount of records or the number of variables / columns.
A purely visual exam of records in each group would be difficult resulting in a lot of hard work highlighting the need to subject the groups to costly external analysis, complex and with questionable results.
KB_STA resolves the problem of the black box which is typical of algorithms of neural networks.
KB_STA:

- submit the file of CV groups to statistical analysis
- evaluates the degree of homogeneity of the groups within them
- evaluates the importance of the variables / columns in cataloging the records in the groups
- groups the records in each group for each variable / column in quartiles (if numeric) or frequency tables (if text values)
- if required, shows for each group and for each variable / column the original value of input records

Source of KB_STA (see attachment 2)

How to use

Having the kb_sta.py program and the input file to process already in the folder, run KB_STA by typing in the window:

python kb_sta.py

where with **python** you ask the ***kb_sta.py*** to be run (in python language)

The program begins the processing by asking in succession:

Catalogued Records File (_outsrt.txt) : vessels_M_g3_outsrt.txt

vessels_M_g3_outsrt.txt is the file in *txt format* containing the table of records / cases cataloged and arranged in *group_code* sequence.
The file *vessels_M_g3_outsrt.txt* is one of the results of the previous processing with the program KB_CAT.

Groups / CV File (_cv.txt) : vessels_M_g3_cv.txt

vessels_M_g3_cv.txt is the file in *txt* format containing the table of the CV of the groups.
The file *vessels_M_g3_cv.txt* is one of the results from the previous processing with the KB_CAT program.
It is important that this file and the previous one come from the same KB_CAT processing.

Report File (output) : vessels_M_g3_report.txt

vessels_M_g3_report.txt is the output file that will contain the statistical analysis of the results obtained from the previous program of cataloging.
It is useful, for clarity, that the name of the report file beginnings as the two previous, as just exemplified above, in the case of statistical analysis with different parameters, the names may change in the final part of the name (example _r1, _r2, _r3).

Display Input Records (Y / N) : n

Group Consistency (% from 0 to 100) : 0

Parameter to request the display of the groups with a percentage of homogeneity inside them not less than that indicated, it is advisable to carry out the initial processing with the parameter set equal to zero, which would show all groups, and then use a different parameter in relation to the results achieved.
Too high a value of this parameter may produce an empty list.

Variable Consistency (% from 0 to 100) : 0

Parameter to request the display of variables, within groups with a percentage of homogeneity of the variable is not less than that indicated, it is advisable to carry out the initial processing with the parameter set equal to zero, which would show all the variables of the groups, and then use a different parameter in relation to the results

obtained.
Too high a value of this parameter may produce an empty list.

Select groups containing records >= : 2

The parameter to request the visualization of the groups composed of at least *x* records. The groups formed by a single record are automatically homogeneous at 100% for all the variables / columns.

Select groups containing records <= : 1000

The parameter to request the display of groups composed of a number of records less than x. The parameter can be useful for examining the groups containing only a few records.

Summary / Detail report (S / D) : d

If the parameter has the value of s/S, the report will contain the values of homogeneity (consistency), the total number of records and the percentage of records cataloged in the group.
If the parameter has a value of d/D, the report will contain numerical values for each quartile, while for each text variable the report will contain the frequency distribution of the different text values.

Display Input Records (Y / N) : n

If the parameter has the value of *n/N* the input records belonging to the groups will not be visualized, on the contrary (*y/Y)* will be visualized.

KB_STA running

```
###############################################################################
# KB_STA KNOWLEDGE DISCOVERY IN DATA MINING (STATISTICAL PROGRAM)             #
# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                 #
# Language used: PYTHON                              .                        #
###############################################################################
Catalogued Records File (_outsrt.txt)       : vessels_M_g3_outsrt.txt
Groups / CV File (_cv.txt)                  : vessels_M_g3_cv.txt
Report File (output)                        : vessels_M_g3_report.txt
Group Consistency (% from 0 to 100)         : 0
Variable Consistency (% from 0 to 100)      : 0
Select groups containing records >=         : 2
Select groups containing records <=         : 1000
Summary / Detail report (S / D)             : d
Display Input Records (Y / N)               : n
Elapsed time (seconds)    :   0.3956
```

Analysis of the results of the cataloging of *vessels.txt*

Example for the group G_00_00

```
===========================================================================
G_00_00 Consistency 0.7140 %Consistency   79  Records  7    %Records 17.50
***  shape           Consistency         0.6910       %Consistency     3.22
Value  cylinder_cone   Frequency           3    Percentage   42.00
Value  ball_cone       Frequency           2    Percentage   28.00
Value  cylinder        Frequency           1    Percentage   14.00
Value  cut_cone        Frequency           1    Percentage   14.00
***  material        Consistency         0.7687       %Consistency     0.00
Value  glass           Frequency           5    Percentage   71.00
Value  terracotta      Frequency           1    Percentage   14.00
Value  metal           Frequency           1    Percentage   14.00
```

```
***  height          Consistency         0.4537       %Consistency    36.46
Mean   53.57 Min   30.00       Max    100.00     Step   17.50
First  Quartile (end)         47.50    Frequency %         57.14
Second Quartile (end)         65.00    Frequency %         14.29
Third  Quartile (end)         82.50    Frequency %         14.29
Fourth Quartile (end)        100.00    Frequency %         14.29
***  colour           Consistency        0.2673       %Consistency    62.56
Value  green             Frequency           5     Percentage    71.00
Value  grey              Frequency           1     Percentage    14.00
Value  brown             Frequency           1     Percentage    14.00
***  weight          Consistency         1.9116       %Consistency     0.00
Mean   2680.71 Min       120.00      Max     15000.00   Step   3720.00
First  Quartile (end)       3840.00    Frequency %         85.71
Fourth Quartile (end)      15000.00    Frequency %         14.29
***  haft            Consistency         0.0000       %Consistency   100.00
Value  no                Frequency           7     Percentage    100.00
***  plug            Consistency         0.9055       %Consistency     0.00
Value  cork              Frequency           3     Percentage    42.00
Value  no                Frequency           2     Percentage    28.00
Value  metal             Frequency           2     Percentage    28.00
===========================================================================
```

The group G_00_00 is composed predominantly of *glass* containers, with a *height* from 30 to 65 cm, *green* color, with a *weight* up to 3840 and without handle (*haft*) .

Analysis of the results of a political poll of 2007

The analysed case takes into consideration a political poll, carried out on 22nd and 23rd November 2007 by Prof. Paolo Natale of the State University of Milan, Department of Political Sciences. After an initial processing of the data which gave no evident results, the database was updated eliminating the fields which were evidently not relevant and grouping some variables.
Fields regarding the size of the area of residence, the judgement on the weight of democracy and politics were eliminated. The field regarding the province of residence was changed, by grouping together the provinces among the north, the centre and the south.
The new starting database contains 982 records relating to persons who have participated in the political poll by answering the questions that follow:

- gender (men / women)
- coalition of confidence regardless of the vote (*ES extreme left, left SS, CS center-left, center CC, CD center-right, right DD, ES extreme right, ** does not answer*)
- profession
- believer (yes/no)
- religion
- expectations for the economic situation for the next 6 months
- judgment on the current state of country's economy
- protection (who you can trust)
- security (perception)
- prediction of the winner of a possible short-term election
- opinion of the government's actions

- opinion of the opposition's actions
- interest in politics
- confidence in coalition
- short term vote
- party voted in 2006
- party you intend to vote in the next election
- PDL party
- age
- region
- qualifications
- attendance at religious functions

The processing with the program KB_CAT was made with 4 groups.
The results of the cataloging obtained by KB_CAT are been then processed by KB_STA.
From an analysis of the results obtained we can draw the following observations:

- supporters of the *left* believe, with the exception of those in the group G_04_02, that the eventual winner will be the *center right*, or do not respond to who will be the future winners (group G_02_04)
- supporters of the *center left / left* defend the government and considering his work on *average*
- supporters of the *centre left / left* on average give a positive opinion on the opposition
- group G_04_04 is formed of apolitical people, agnostic (or extremely reserved), people who prefer not to express an opinion, they are pensioners and *unemployed*, over 50 years of age
- does not exist, as in the past, a relationship between the profession, the economic condition and the trusted party
- the category of pensioners is divided between those who imagine a victory by the *centre right* (group G_02_03) and those who prefer not to reply (Group G_02_04 e G_04_04)
- age does not affect cataloging
- in all groups people say they do not want to give the vote to the PDL, even in groups in which the same people speculate that the winner will be the *center right* (G_01_04, G_02_03, G_04_01)

A large part of the observations expressed above were confirmation of the loss of ideological values linked to the *hard core* of belonging to a social class, age group, level of education, area of residency, etc., important characteristics in the past for political tendencies for the voters.
In 2007 the idea of *liquid* voters came into use, that is, the people who are no longer a *supporter* of a party or an alliance, but able to evaluate the results of government and opposition actions and decide whether and how to vote.
It 'obvious that if you define detailed profiles of voters you can then formulate specific election programs and not only based on ideologies almost meaningless.

KB_CLA – Classification of new records

Generalities, aims and functions

The KB_CAT program produces the file containing the training matrix (for example *vessels_M_g3_grid.txt*) which can be *immediately* used to classify records that are similar to records that have been previously cataloged.
The use of classification programs *on the fly* are very useful when you must act quickly taking into consideration the knowledge already acquired.
Classifications running in real time can be found, for example:

- in banking / insurance fields for the prevention of illegal activity
- in the business of mobile phones to identify customers in preparing to move to competition
- in the quality control of industrial processes and products
- in companies to avoid cases of insolvency with clients

Source of KB_CLA (attachment 3)

How to run

KB_CLA requires that the file of the new records / cases to classify has the same structure and a similar content as the file used in the previous KB_CAT processing.
For the same structure we mean that the file of the new records / cases must have the same number of variables / columns with an identical format of data (numerical / textual).
For similar content means that the file of new records / cases should contain records from samples of the same universe.
Acquired knowledge for the cataloging of animals, can not be used to classify new containers!

Input files = n_vessels.txt

Contents of the file n_vessels.txt

The records / cases to be classify are reported in the following table and are identified by the first character *N* in the description.

description	shape	material	height	colour	weight	haft	plug
n_glass	cut_cone	terracotta	6	transparent	22	No	no
n_bottle	cylinder_cone	glass	37	brown	120	No	metal
n_tea_cup	cut_cone	ceramic	7	white	28	Yes	no
n_cup	cut_cone	glass	22	transparent	36	Yes	no
n_coffee_cup	cut_cone	glass	6	transparent	19	Yes	no
n_perfume	cylinder	glass	7	transparent	12	No	plastic
n_trousse	cylinder	plastic	1	blue	6	No	yes
n_plant_pot	cut_cone	terracotta	40	brown	180	No	no
n_pasta_case	cylinder	glass	30	transparent	130	No	metal

Number of Groups (3 – 20) = 3

Normalization(Max, Std, None) = m

File Training Grid = vessels_M_g3_grid.txt

KB_CLA running

```
###############################################################################
# KB_CLA KNOWLEDGE DISCOVERY IN DATA MINING (CLASSIFY PROGRAM)                #
# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                 #
# Language used: PYTHON                                                       #
###############################################################################
InputFile                              : n_vessels.txt
Number of Groups (3 - 20)              : 3
Normalization(Max, Std, None)          : m
File Training Grid                     : vessels_M_g3_grid.txt
Output File Classify.original n_vessels_CM_g3_out.txt
Output File Classify.sort     n_vessels_CM_g3_outsrt.txt
Output File Summary sort      n_vessels_CM_g3_sort.txt
Output File Matrix Catal.     n_vessels_CM_g3_catal.txt
Output File Means, STD, CV.   n_vessels_CM_g3_medsd.txt
Output File CV of the Groups  n_vessels_CM_g3_cv.txt
Output File Training Grid     vessels_M_g3_grid.txt
Output File Run Parameters    n_vessels_CM_g3_log.txt
Elapsed time (seconds)    :   0.16115
```

Analysis of the results of the classification of n_vessels.txt

The new records, recognisable by the first letter *N*, have been inserted into the previous table obtained by KB_CAT.

Group	description	shape	material	height	colour	Weight	haft	plug
G_00_00	ancient_bottle	ball_cone	glass	40.0	green	150.0	no	cork
G_00_00	bottle_1	cylinder_cone	glass	40.0	green	120.0	no	cork
G_00_00	bottle_4	cylinder_cone	glass	35.0	green	125.0	no	metal
G_00_00	carboy	ball_cone	glass	80.0	green	15000.0	no	cork
G_00_00	magnum_bottle	cylinder_cone	glass	50.0	green	170.0	no	metal
G_00_00	plant_pot	cut_cone	terracotta	30.0	brown	200.0	no	no
G_00_00	umbrella_stand	cylinder	metal	100.0	grey	3000.0	no	no
G_00_00	n_bottle	cylinder_cone	glass	37.0	brown	120.0	no	metal
G_00_00	n_glass	cut_cone	terracotta	6.0	transparent	22.0	no	no
G_00_00	n_pasta_case	cylinder	glass	30.0	transparent	130.0	no	metal
G_00_00	n_plant_pot	cut_cone	terracotta	40.0	brown	180.0	no	no
G_00_01	pot_1	cylinder	metal	40.0	grey	500.0	two	yes
G_00_02	coffee_cup	cut_cone	ceramic	6.0	white	20.0	yes	no
G_00_02	cup_1	cut_cone	ceramic	10.0	white	30.0	yes	no
G_00_02	cup_2	cut_cone	glass	20.0	transparent	35.0	yes	no

Group	description	shape	material	height	colour	Weight	haft	plug
G_00_02	pot_2	cut_cone	metal	7.0	grey	200.0	yes	yes
G_00_02	n_coffee_cup	cut_cone	glass	6.0	transparent	19.0	yes	no
G_00_02	n_tea_cup	cut_cone	ceramic	7.0	white	28.0	yes	no
G_01_00	beer_jug	cut_cone	porcelain	18.0	severals	25.0	no	no
G_01_00	bottle_2	cylinder_cone	glass	40.0	transparent	125.0	no	cork
G_01_00	bottle_3	cylinder_cone	glass	45.0	opaque	125.0	no	plastic
G_01_00	glass_1	cut_cone	pewter	10.0	pewter	20.0	no	no
G_01_00	glass_3	cut_cone	terracotta	8.0	grey	20.0	no	no
G_01_00	tuna_can	cylinder	metal	10.0	severals	10.0	no	no
G_01_00	n_perfume	cylinder	glass	7.0	transparent	12.0	no	plastic
G_01_00	n_trousse	cylinder	plastic	1.0	blue	6.0	no	yes
G_01_02	n_cup	cut_cone	glass	22.0	transparent	36.0	yes	no
G_02_00	cd	parallelepiped	plastic	1.0	transparent	4.0	no	no
G_02_00	champagne_glass	cut_cone	crystal	17.0	transparent	17.0	no	no
G_02_00	dessert_glass	cut_cone	glass	17.0	transparent	17.0	no	no
G_02_00	glass_2	cut_cone	plastic	9.0	white	4.0	no	no
G_02_00	pasta_case	parallelepiped	glass	35.0	transparent	150.0	no	metal
G_02_00	perfume	parallelepiped	glass	7.0	transparent	15.0	no	plastic
G_02_00	tetrapack1	parallelepiped	mixed	40.0	severals	20.0	no	plastic
G_02_00	tetrapack2	parallelepiped	plastic	40.0	severals	21.0	no	plastic
G_02_00	tetrapack3	parallelepiped	millboard	40.0	severals	22.0	no	no
G_02_00	toothpaste	cylinder	plastic	15.0	severals	7.0	no	plastic
G_02_00	trousse	cylinder	plastic	1.0	silver	7.0	no	yes
G_02_00	tuna_tube	cylinder	plastic	15.0	severals	7.0	no	plastic
G_02_00	visage_cream	cylinder	metal	15.0	white	7.0	no	no
G_02_00	wine_glass	cut_cone	glass	15.0	transparent	15.0	no	no
G_02_01	pyrex	parallelepiped	glass	10.0	transparent	300.0	two	glass
G_02_02	cleaning_1	parall_cone	plastic	30.0	white	50.0	yes	plastic
G_02_02	cleaning_2	cylinder_cone	plastic	30.0	blue	60.0	yes	plastic
G_02_02	cleaning_3	cone	plastic	100.0	severals	110.0	yes	plastic
G_02_02	jug	cylinder	terracotta	25.0	white	40.0	yes	no
G_02_02	milk_cup	cut_cone	terracotta	15.0	blue	35.0	yes	no
G_02_02	tea_cup	cut_cone	terracotta	7.0	white	30.0	yes	no
G_02_02	watering_can	irregular	plastic	50.0	green	400.0	yes	no

The new records have been classified, almost completely, in the correct way except for the two records highlighted in pink colour.
The *n_glass* record has been classified in the group G_00_00 with the variables *colour* e *weight* with values that are not present in other records of the group.

Political opinions in Facebook (January 2013)

A sample of 1070 political opinions present in 14 different groups of discussion was examined: fb_casini, fb_fini, fb_bonino, fb_di_pietro, fb_corsera, fb_fanpage, fb_brambilla, fb_storace, fb_maroni, fb_bersani, fb_meloni, fb_grillo, fb_termometro_politico, fb_fattoquotidiano.
For every post are taken into consideration:

- the group of discussion
- the topic of the discussion (e.g. gay marriage and adoption, interview by TG3, Rai 3 news, the moral drift)
- the political leader involved (Bersani, Casini, Berlusconi, etc.)
- the attitude inferred from the judgment expressed by the author of the post using a 5-point scale (1 = an abusive opinion, 2 = a negative opinion, 3 = an indifferent opinion, 4 = a positive opinion, 5 = a laudatory judgment).

The research aims to explore the possible relationships existing between the groups, the arguments, the politicians and the opinions expressed. KB_CAT has cataloged the 1070 opinions in 25 groups, 15 of which contain a significant number of views.

Group 00 Records 132
groups: fb_corsera, fb_fanpage
politicians: Berlusconi, Dell'Utri, Bersani
topics: always-candidated, moral drift, euro 100000 (cheque for Veronica)
abusive opinion, negative opinion
comment: groups not aligned, and especially the fb_corsera group in which the insults abound

----

Group 04 Records 115
groups: fb_maroni, fb_storace, fb_meloni
politicians: Maroni, Storace, Meloni
topics: diary
positive, laudatory opinions
comment: groups aligned

Group 24 Records 85
groups: fb_fanpage, fb_grillo, fb_fattoquotidiano
politicians: Grillo, Ingroia
topics: various
positive opinion
comment: 2 of the groups not aligned but Grillo and Ingroia are new so they attract people

----

Group 02 Records 69
groups: fb_brambilla, fb_casini, fb_bersani
politicians: Brambilla, Casini, Bersani
topics: various
positive, laudatory opinions
comment: aligned groups, where the animal rights mission “pays”

Group 40 Records 69
groups: fb_termometro_politico
politicians: Berlusconi, Ingroia
topics: public services, succession, taxes
offensive, negative opinions
comment: not aligned, politicians and topics that are “hot”

Group 44 Records 66
groups: fb_corsera, fb_meloni, fb_fanpage
politicians: Pannella, Meloni, Vendola
topics: alliances, diary attack
offensive, negative opinions
comment: group fb_corsera not aligned and the unpopular alliance of Pannella with

Storace

Group 43 Records 66
groups: fb_fanpage
politician: Monti
topics: no_marr_adop_gay, monti_su_fb
offensive, negative opinions
comment; not aligned and disagreement on no_marr_adop_gay

Group 12 Records 58
groups: fb_bonino, fb_brambilla, fb_casini
politicians: Bonino, Brambilla, Casini
topics: president_republic, animalist_bill
positive opinion
comment: aligned groups

Group 11 Records 51
groups: fb_casini, fb_bersani
politicians: Casini, Bersani
topics: the 11 lies, interview with TG3 (national news), interview with TG5 (national news)
negative opinion
comment: criticism for aligned groups on non shared opinions

Group 23 Records 44
groups: fb_dipietro, fb_casini
politicians: Di Pietro, Casini, Grillo
topics: tv_adverts, last_word
positive, laudatory opinion
comment: aligned groups

Group 42 Records 43
groups: fb_fanpage, fb_corsera
politicians: Monti, Grillo
topics: piper, profile_fb, monti_exorcist
offensive opinion
comment: not aligned groups

Group 14 Records 40
groups: fb_fanpage, fb_grillo
politicians: Monti, Grillo
topics: no_marr_adop_gay, meeting_lecce
positive, laudatory opinion
comment: laudatory opinions on the two topics of Monti and Grillo
----
Group 21 Records 39
groups: fb_bonino, fb_casini
politicians: Bonino, Casini
topics: regional, pact_monti_bersani, president_republic

positive opinion
comment: aligned groups

Group 20 Records 33
groups: fb_fanpage
politician: Bersani
topics: pact_monti_bersani
offensive negative opinion
comment: not aligned group

Group 31 Records 32
groups: fb_di_pietro
politician: Di Pietro
topics: rivoluzione_civile, tv_advert
offensive negative opinion
comment: aligned group and disagreement on “rivoluzione_civile” (civil revolution)

Summary

There are close relationships between the typology of the groups, the politicians, the topics and the opinions.

In the groups "aligned":

- positive and laudatory opinions are plentiful
- any possible disagreements arise from sympathizers who do not share any political positions or from opponents who were immediately marginalised
- obscene language is rare and the syntactical and grammatical forms are proper

In the groups “not aligned”:

- prevails much dissent on consensus
- bad language is the norm and the syntactic and grammar forms are poor
- persons are aware not to suffer criticism
- open discussions on issues banned in the groups "aligned"

Know4Business (Cloud version in Google App Engine)

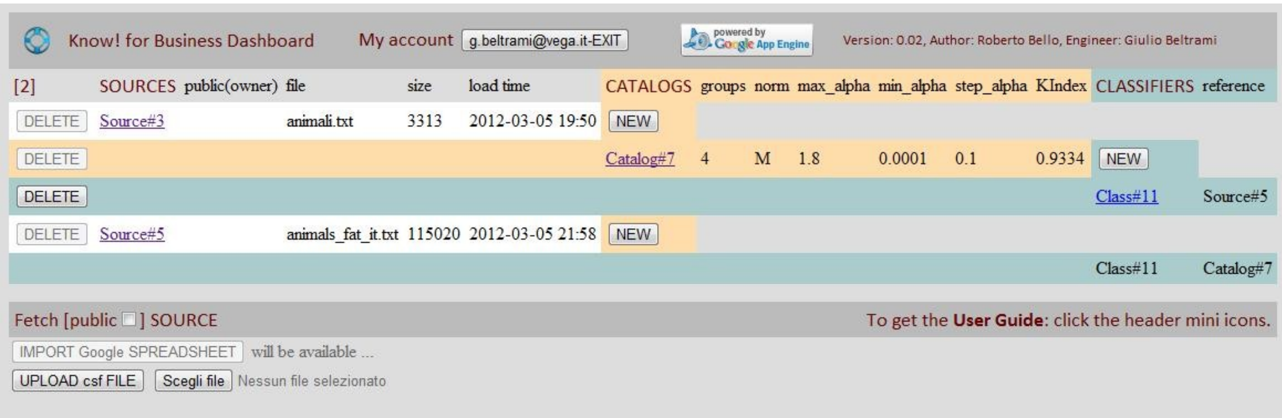
Giulio Beltrami, software engineer and expert in innovative architectures ICT of the social type, has transferred the KB in the field of Cloud Computing of the Google App Engine with the name of Know4Business.
Know4Business is usable in *pay to use* mode and is obtainable in Internet at the link http://know4business.appspot.com/.

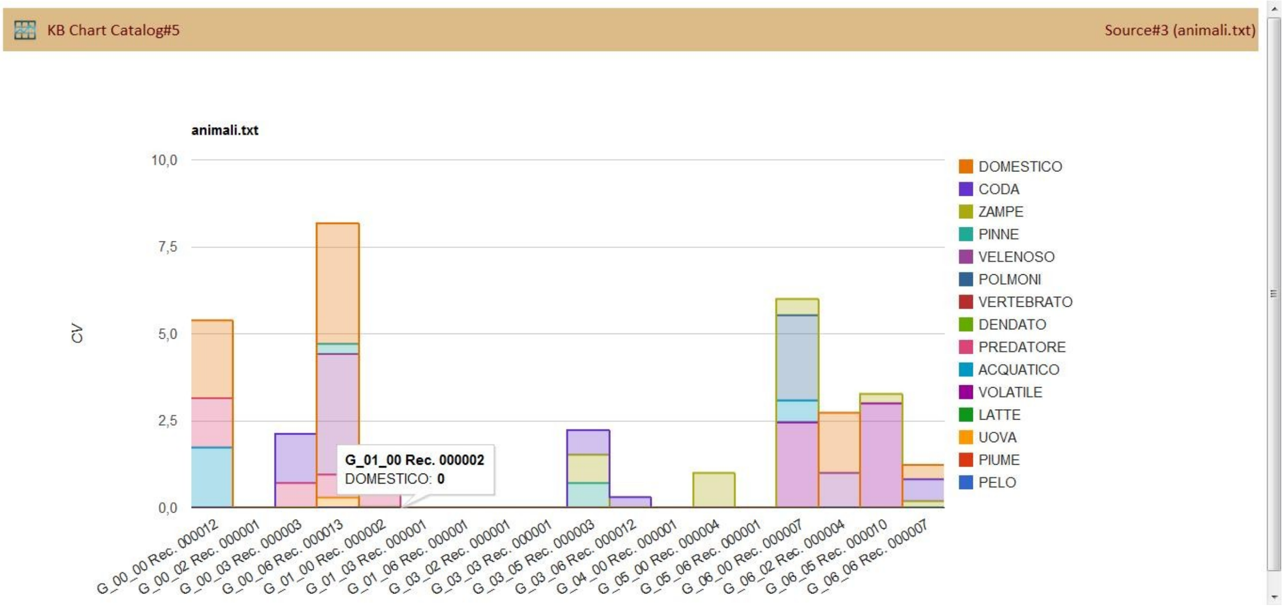
Know4Business adds to Google-Apps a powerful general-purpose discovering of the hidden knowledge, in your business data, enabled by a well-known neural-network self-learning data-mining algorithms.
Know4Business provides 5 tools, to fulfill the knowledge discovery:
•SOURCE for preparing (checking, normalizing, cleaning, filtering and encoding) the input data.

•CATALOGUE for discovering groups in the sample data, that are in some way or another "similar", without using known structures.
•STATISTICS to evaluate the success of the catalogue clustering.
•CLASSIFIER for generalizing known structures, to apply to new data.
•AGENCY to return some diagnosis and suggestions, about the user data management, checking the success of the classification. plus an interactive CONSOLE to help the data-analyst to:
•run the catalogue and the other tools
•Report and graph the results of the tools

Know4Business end-to-end process of knowledge discovery, provides a simple work-flow, with some useful feedback capabilities:
•**Classification** operates forward to the catalogue of the sample data.
•**Statistics** of the catalogue clustering can suggest some filtering rules, both on cardinality and dimensions, on the sample data.
•**Agency** can also influence some source filtering rules and/or put something to data management, depending on the circumstances. which enables a kind of data knowledge “auto-poiesis”, minimizing human intervention.

Know4Business - Main advantages:
•The ease of use, based upon a simple HTML 5 GUI, to use the tools and to look to the results.
•The clear implementation, based upon an object oriented paradigm and an authentic SaaS, for the cloud computing, architecture.





APPENDIXES

Appendix 1 – KB_CAT source

```
# -*- coding: utf-8 -*-
###############################################################################
# KB_CAT KNOWLEDGE DISCOVERY IN DATA MINING (CATALOG PROGRAM)                 #
# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                 #
# Language used: PYTHON                                        .             #
###############################################################################
import os
import random
import copy
import datetime

def mean(x):         # mean
  n = len(x)
  mean = sum(x) / n
  return mean

def sd(x):           # standard deviattion
  n = len(x)
  mean = sum(x) / n
  sd = (sum((x-mean)**2 for x in x) / n) ** 0.5
  return sd

class ndim:                 # from 3D array to flat array
    def __init__(self,x,y,z,d):
        self.dimensions=[x,y,z]
        self.numdimensions=d
        self.gridsize=x*y*z
```

```
    def getcellindex(self, location):
        cindex = 0
        cdrop = self.gridsize
        for index in xrange(self.numdimensions):
            cdrop /= self.dimensions[index]
            cindex += cdrop * location[index]
        return cindex

    def getlocation(self, cellindex):
        res = []
        for size in reversed(self.dimensions):
            res.append(cellindex % size)
            cellindex /= size
        return res[::-1]

""" how to use ndim class
n=ndim(4,4,5,3)
print n.getcellindex((0,0,0))
print n.getcellindex((0,0,1))
print n.getcellindex((0,1,0))
print n.getcellindex((1,0,0))

print n.getlocation(20)
print n.getlocation(5)
print n.getlocation(1)
print n.getlocation(0)
"""

print("###############################################################################")
print("# KB_CAT KNOWLEDGE DISCOVERY IN DATA MINING (CATALOG PROGRAM)                 #")
print("# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                 #")
print("# Language used: PYTHON                                                       #")
print("###############################################################################")

# input and run parameters
error = 0

while True:
  arch_input = raw_input('InputFile                              : ')
  if not os.path.isfile(arch_input):
    print("Oops! File does not exist. Try again... or CTR/C to exit")
  else:
    break

while True:
```

```
  try:
    num_gruppi = int(raw_input('Number of Groups (3 - 20)              : '))
  except ValueError:
    print("Oops!  That was no valid number.  Try again...")
  else:
    if(num_gruppi < 3):
      print("Oops! Number of Groups too low. Try again...")
    else:
      if(num_gruppi > 20):
        print("Oops! Number of Groups too big. Try again...")
      else:
        break

while True:
  normaliz   = raw_input('Normalization(Max, Std, None)          : ')
  normaliz   = normaliz.upper()
  normaliz   = normaliz[0]
  if(normaliz <> 'M' and normaliz <> 'S' and normaliz <> 'N'):
    print("Oops! Input M, S or N. Try again...")
  else:
    break

while True:
  try:
    max_alpha   = float(raw_input('Start value of alpha (1.8 - 0.9)       : '))
  except ValueError:
    print("Oops!  That was no valid number.  Try again...")
  else:
    if(max_alpha > 1.8):
      print("Oops! Start value of alpha too big. Try again...")
    else:
      if(max_alpha < 0.9):
        print("Oops! Start value of alpha too low. Try again...")
      else:
        break

while True:
  try:
    min_alpha  = float(raw_input('End value of alpha (0.5 - 0.0001)      : '))
  except ValueError:
    print("Oops!  That was no valid number.  Try again...")
  else:
    if(min_alpha > 0.5):
      print("Oops! alpha too big. Try again...")
    else:
      if(min_alpha < 0.0001):
```

```
        print("Oops! alpha too low. Try again...")
      else:
        break

while True:
  try:
    step_alpha  = float(raw_input('Decreasing step of alpha (0.1 - 0.001) : '))
  except ValueError:
    print("Oops!  That was no valid number.  Try again...")
  else:
    if(step_alpha > 0.1):
      print("Oops! Decreasing step of alpha too big. Try again...")
    else:
      if(step_alpha < 0.001):
        print("Oops! Decreasing step of alpha too low. Try again...")
      else:
        break



file_input   = arch_input
gruppi_num   = num_gruppi
tipo_norm    = normaliz
alpha_min    = min_alpha
alpha_max    = max_alpha
alpha_step   = step_alpha

# outputs files
file_input   = arch_input
tipo_norm    = normaliz
gruppi_num   = num_gruppi
nome_input   = file_input.split(".")
arch_output  = nome_input[0] + "_" + tipo_norm + "_g" + str(gruppi_num) + "_out.txt"
arch_outsrt  = nome_input[0] + "_" + tipo_norm + "_g" + str(gruppi_num) + "_outsrt.txt"
arch_sort    = nome_input[0] + "_" + tipo_norm + "_g" + str(gruppi_num) + "_sort.txt"
arch_catal   = nome_input[0] + "_" + tipo_norm + "_g" + str(gruppi_num) + "_catal.txt"
arch_medsd   = nome_input[0] + "_" + tipo_norm + "_g" + str(gruppi_num) + "_medsd.txt"
arch_cv      = nome_input[0] + "_" + tipo_norm + "_g" + str(gruppi_num) + "_cv.txt"
arch_grid    = nome_input[0] + "_" + tipo_norm + "_g" + str(gruppi_num) + "_grid.txt"
arch_log     = nome_input[0] + "_" + tipo_norm + "_g" + str(gruppi_num) + "_log.txt"

# start time
t0 = datetime.datetime.now()

# read input file
arr_r    = []
arr_orig = []
```

```
arr_c    = []
mtchx    = []
mtchy    = []
txt_col  = []
xnomi    = []

# the numbers of variables / columns in all record must be the same
n_rows = 0
n_cols = 0
err_cols = 0
index = 0
for line in open(file_input).readlines():
  linea = line.split()
  if(index == 0):
    xnomi.append(linea)
    n_cols = len(linea)
  else:
    arr_r.append(linea)
    if(len(linea) != n_cols):
      err_cols = 1
      print("Different numbers of variables / columns in the record " + str(index)
        + " cols " + str(len(linea)))
  index += 1
if(err_cols == 1):
  print("File " + file_input + " contains errors. Exit ")
  quit()
index = 0
while index < len(arr_r):
  linea = arr_r[index]
  index_c = 0
  while index_c < len(linea):
    if linea[index_c].isdigit():
      linea[index_c] = float(linea[index_c])
    index_c += 1
  arr_r[index] = linea
  index += 1
arr_orig = copy.deepcopy(arr_r)        # original input file
testata_cat = copy.deepcopy(xnomi[0])  # original header row

# finding columns containing strings and columns containing numbers
testata = xnomi[0]
testata_orig = copy.deepcopy(xnomi[0])
n_cols = len(testata) - 1
n_rows = len(arr_r)
ind_c  = 1
err_type = 0
```

```
while ind_c < len(testata):
  ind_r    = 1
  tipo_num = 0
  tipo_txt = 0
  while ind_r < len(arr_r):
    arr_c = arr_r[ind_r]
    if isinstance(arr_c[ind_c],basestring):
      tipo_txt = 1
    else:
      tipo_num = 1
    ind_r += 1
  if tipo_num == 1 and tipo_txt == 1:
    print "The columns / variables " + testata[ind_c] + " contains both strings and 
numbers."
    print arr_c
    err_type = 1
  ind_c += 1
if err_type == 1:
  print "Oops! The columns / variables contains both strings and numbers. Exit. "
  quit()

index_c = 1
while index_c <= n_cols:
  txt_col = []
  index = 0
  while index < len(arr_r):
    arr_c = arr_r[index]
    if(isinstance(arr_c[index_c],str)):
      txt_col.append(arr_c[index_c])
    index += 1
  set_txt_col = set(txt_col)            # remove duplicates
  txt_col = list(set(set_txt_col))
  txt_col.sort()

  # from strings to numbers
  if(len(txt_col) > 0):
    if(len(txt_col) > 1):
      passo1 = 1.0 / (len(txt_col) - 1)
    else:
      passo1 = 0.0
    index = 0
    while index < len(arr_r):
      arr_c = arr_r[index]
      campo1 = arr_c[index_c]
      indice1 = txt_col.index(campo1)
      if(len(txt_col) == 1):  # same values in the column
```

```
        val_num1 = float(1)
      else:
        val_num1 = float(passo1 * indice1)
      arr_c[index_c] = val_num1 + 0.00000001   # to avoid zero values in means
                                               # (to prevent zero divide in CV)
      index += 1
  index_c += 1

# means, max & std
xmeans = []
xmaxs  = []
xmins  = []            ### aggiunto Roberto 4/03/2012
xsds   = []
xcv    = []
index_c = 0
while index_c <= n_cols:
  xmeans.append(0.0)
  xmaxs.append(-9999999999999999.9)
  xmins.append(9999999999999999.9)     ### aggiunto Roberto 4/03/2012
  xsds.append(0.0)
  xcv.append(0.0)
  index_c += 1

# means & max
index = 0
while index < n_rows:
  arr_c = arr_r[index]
  index_c = 1
  while index_c <= n_cols:
    xmeans[index_c] += arr_c[index_c]
    if(arr_c[index_c] > xmaxs[index_c]):
      xmaxs[index_c] = arr_c[index_c]
    index_c += 1
  index += 1
index_c = 1
while index_c <= n_cols:
  xmeans[index_c] = xmeans[index_c] / n_rows
  index_c += 1

# std
index = 0
while index < n_rows:
  arr_c = arr_r[index]
  index_c = 1
  while index_c <= n_cols:
    xsds[index_c] += (arr_c[index_c] - xmeans[index_c])**2
```

```
    index_c += 1
  index += 1
index_c = 1

while index_c <= n_cols:
  xsds[index_c] = (xsds[index_c] / (n_cols - 1)) ** 0.5
  index_c += 1

# Means, Max, Std, CV output file
medsd_file = open(arch_medsd, 'w')

# columns names
medsd_file.write('%s %s ' % ('Function' , "\t"))
index_c = 1
while index_c <= n_cols:
  medsd_file.write('%s %s ' % (testata[index_c], "\t"))
  index_c += 1
medsd_file.write('%s' % ('\n'))

# means
medsd_file.write('%s %s ' % ('Mean' , "\t"))
index_c = 1
while index_c <= n_cols:
  valore = str(xmeans[index_c])
  valore = valore[0:6]
  medsd_file.write('%s %s ' % (valore, "\t"))
  index_c += 1
medsd_file.write('%s' % ('\n'))

# max
medsd_file.write('%s %s ' % ('Max' , "\t"))
index_c = 1
while index_c <= n_cols:
  valore = str(xmaxs[index_c])
  valore = valore[0:6]
  medsd_file.write('%s %s ' % (valore, "\t"))
  index_c += 1
medsd_file.write('%s' % ('\n'))

# std
medsd_file.write('%s %s ' % ('Std' , "\t"))
index_c = 1
while index_c <= n_cols:
  valore = str(xsds[index_c])
  valore = valore[0:6]
  medsd_file.write('%s %s ' % (valore, "\t"))
```

```
  index_c += 1
medsd_file.write('%s' % ('\n'))

# CV
medsd_file.write('%s %s ' % ('CV' , "\t"))
index_c = 1
med_cv_gen = 0.0          # cv average of all columns / variables
while index_c <= n_cols:
  if xmeans[index_c] == 0:
    media1 = 0.000001
  else:
    media1 = xmeans[index_c]
  xcv[index_c] = xsds[index_c] / abs(media1)
  valore = str(xcv[index_c])
  med_cv_gen += xcv[index_c]
  valore = valore[0:6]
  medsd_file.write('%s %s ' % (valore, "\t"))
  index_c += 1
med_cv_gen = med_cv_gen / n_cols
str_med_cv_gen = str(med_cv_gen)
str_med_cv_gen = str_med_cv_gen[0:6]
medsd_file.write('%s' % ('\n'))
medsd_file.close()

# input standardization

# standardization on max

if tipo_norm == 'M':
  index = 0
  while index < n_rows:
    arr_c = arr_r[index]
    index_c = 1
    while index_c <= n_cols:    ## aggiornare anche kb_cla.py
      if xmaxs[index_c] == 0.0:
        xmaxs[index_c] = 0.00001
      arr_c[index_c] = arr_c[index_c] / xmaxs[index_c]
      index_c += 1
    index += 1

# standardization on std

if tipo_norm == 'S':
  index = 0
  while index < n_rows:
    arr_c = arr_r[index]
```

```
    index_c = 1
    while index_c <= n_cols:
      if xsds[index_c] == 0.0:
        xsds[index_c] = 0.00001
      arr_c[index_c] = (arr_c[index_c] - xmeans[index_c]) / xsds[index_c]
      if arr_c[index_c] < xmins[index_c]:     ### aggiunto Roberto 4/03/2012
        xmins[index_c] = arr_c[index_c]              ### aggiunto Roberto 4/03/2012
      index_c += 1
    index += 1
  # aggiungo xmins per eliminare i valori negativi (aggiunto da Roberto 4/03/2012)
  index = 0
  while index < n_rows:
    arr_c = arr_r[index]
    index_c = 1
    while index_c <= n_cols:
      arr_c[index_c] = arr_c[index_c] - xmins[index_c]
      print arr_c[index_c]
      index_c += 1
    index += 1
  # fine aggiunta da Roberto 4/03/2012

# start of kohonen algorithm

# min and max vectors

vmaxs = []
vmins = []

index_c = 0

while index_c <= n_cols:
  vmaxs.append(-10000000000000.0)
  vmins.append( 10000000000000.0)
  index_c += 1

# columns min & max
index = 0
while index < n_rows:
  arr_c = arr_r[index]
  index_c = 1
  while index_c <= n_cols:
    if arr_c[index_c] > vmaxs[index_c]:
      vmaxs[index_c] = arr_c[index_c]
    if arr_c[index_c] < vmins[index_c]:
      vmins[index_c] = arr_c[index_c]
    index_c += 1
```

```
  index += 1

# run parameters and temp arrays

n = n_rows
m = n_cols
nx = gruppi_num
ny = gruppi_num
ix = 950041                          # integer as random seed
nsteps = int(10000 * nx * ny)        # number of steps
nepoks = int(nsteps / n ** 0.5)      # number of epochs
unit_calc = int(n * m * nx * ny)     # running units
passo = int(5000 / n)                # step of visualization on monitor
rmax = nx - 1
rmin = 1.0

if passo < 1:
  passo = 1
grid = []                     # training grid
index = 0
while index < nx * ny * m:
  grid.append(0.0)
  index += 1
n=ndim(nx,ny,m,3)
random.seed(ix)                           # initial value of random seed to obtain the
same sequences in new runs
index = 0
while index < nx:
  index_c = 0
  while index_c < ny:
    index_k = 0
    while index_k < m:
      ig = n.getcellindex((index,index_c,index_k))
      grid[ig] = random.random()
      index_k += 1
    index_c += 1
  index += 1
gridp = copy.deepcopy(grid)     # initial previous grid = current grid
gridm = copy.deepcopy(grid)     # initial min grid = current grid

# for each record in each epoch
iter     = 0
discrea   = 1000000000000.0      # current error
discrep   = 0.0            # previous error
if nepoks < 20:
  nepoks  = 20                    # min epochs = 20
```

```
nepokx    = 0
min_epok  = 0                    # epoch with min error
min_err   = 1000000000.0         # min error
alpha     = float(alpha_max)     # initial value of alpha parameter
ir        = 0.0                 # initial value of ir parameter ir
ne        = 1

print " "
print 'Record ' + str(n_rows) + ' Columns ' + str(n_cols)

# main loop
try:
  while ne <= nepoks:
    if (ne % passo == 0):  # print running message when modulo division = zero
      min_err_txt = "%14.5f" % min_err    # format 8 integers and 3 decimals
      alpha_txt  = "%12.5f" % alpha       # format 6 integers and 5 decimals
      print ('Epoch ' + str(ne) + '   min err ' + min_err_txt + '   min epoch ' +
          str(min_epok - 1) + "   alpha " + alpha_txt)
    if min_err < 1000000000.0:
      nepokx += 1
    if min_err > discrea and discrep > discrea and discrea > 0.0:
      min_epok = ne               # current epoch (min)
      min_err = discrea
      # copy current grid to min grid
      gridm = copy.deepcopy(grid)
      min_err_txt = "%12.3f" % min_err    # format 8 integers and 3 decimals
      alpha_txt  = "%12.5f" % alpha       # format 6 integer and  5 decimals
      print ('**** Epoch ' + str(ne - 1) + '       WITH MIN ERROR ' + min_err_txt +
         "   alpha " + alpha_txt)

    # cheking the current value of alpha
    if alpha > alpha_min:
      discrea = discrep
      discrep = 0.0
      # copy current grid to previous grid
      gridp = copy.deepcopy(grid)

      # from the starting row to the ending row
      i = 0
      while i < n_rows:
        iter += 1
        # find the best grid coefficient
        ihit = 0
        jhit = 0
        dhit = 100000.0
        igx = 0
```

```
igy = 0
while igx < nx:
  igy = 0
  while igy < ny:
    d = 0.0
    neff = 0
    k = 0
    arr_c = arr_r[i]
    while k < m:   # update the sum of squared deviation of input
                   # value from the grid coefficient
      ig = n.getcellindex((igx,igy,k))
      d = d + (arr_c[k+1] - grid[ig]) ** 2
      k += 1
    d = d / float(m)
    #  d = d / m
    if d < dhit:
      dhit = d
      ihit = int(igx)
      jhit = int(igy)
    igy += 1
  igx += 1
# update iteration error
discrep = discrep + dhit
# now we have the coordinates of the best grid coefficient
ir = max(rmax * float(1001 - iter) / 1000.0 + 0.9999999999 , 1)
ir = int(ir)
# new alpha value to increase the radius of groups proximity
alpha = max(alpha_max * float(1 - ne * alpha_step) , alpha_min)
# update the grid coefficients applying alpha parameter
inn0 = int(ihit) - int(ir)
inn9 = int(ihit) + int(ir)
jnn0 = int(jhit) - int(ir)
jnn9 = int(jhit) + int(ir)
while inn0 <= inn9:
  jnn0 = int(jhit) - int(ir)
  while jnn0 <= jnn9:
    if not (inn0 < 0 or inn0 >= nx):
      if not (jnn0 < 0 or jnn0 >= ny):
        arr_c = arr_r[i]
        k = 0
        while k < m:
          ig = n.getcellindex((inn0,jnn0,k))
          grid[ig] += alpha * (arr_c[k+1] - grid[ig])
          k += 1
    jnn0 += 1
  inn0 += 1
```

```
        i += 1
      else:
        print
        print "Min alpha reached "
        print
        break
      ne += 1
except KeyboardInterrupt:
  print
  print "KeyboardInterrupt (Ctrl/C) "
  print
  pass

# computing results
# grid = grid min
grid = copy.deepcopy(gridm)

# write min grid file
arch_grid_file = open(arch_grid, 'w')
ii = 0
while ii < nx:
  j = 0
  while j < ny:
    k = 0
    while k < m:
      ig = n.getcellindex((ii,j,k))
      arch_grid_file.write('%6i %s %.6i %s %.6i %s %14.7f %s' % (ii,' ', j ,' ', k,' ', 
grid[ig], "\n"))
      k += 1
    j += 1
  ii += 1
arch_grid_file.close()

# catalog input by min grid
ii = 0
while ii < n_rows:
  ihit = 0
  jhit = 0
  dhit = 100000.0
  # from 1 to numbers of groups
  ir = 0
  while ir < nx:         # from 1 to numbers of groups
    jc = 0
    while jc < ny:       # from 1 to numbers of groups
      d = 0.0
      neff = 0
```

```
      k = 0
      while k < n_cols:  # update the sum of squared deviation of input
                         # value from the grid coefficient
        arr_c = arr_r[ii]
        ig = n.getcellindex((ir,jc,k))
        d = d + (arr_c[k+1] - grid[ig]) ** 2
        k += 1
      d = d / m
      if d < dhit:       # save the coordinates of the best coefficient
        dhit = d
        ihit = ir
        jhit = jc
      jc += 1
    ir += 1
  mtchx.append(ihit)
  mtchy.append(jhit)
  ii += 1

# write arch_catal file
arch_catal_file = open(arch_catal, 'w')
ii = 0
while ii < n_rows:
  arch_catal_file.write("%.6i %s %.6i %s %.6i %s" % (ii, ' ', mtchx[ii], ' ', mtchy[ii],
"\n"))
  ii += 1
arch_catal_file.close()

# matrix of statistics
arr_cv   = []             # CV array of the Groups and Total
arr_med  = []             # means array of the Groups
riga_cv  = []                   # CV row in arr_cv
arr_col  = []                   # group temporary array
arr_grsg = []                   # input data array (normalized)
arr_grsg_c = []               # copy of arr_grsg (for file out sort)

# input matrix sort in group sequence
ii = 0
ix = 0
while ii < n_rows:
  ix += 1
  gr1 = str(mtchx[ii])
  if mtchx[ii] < 10:
    gr1 = '0' + str(mtchx[ii])
  sg1 = str(mtchy[ii])
  if mtchy[ii] < 10:
    sg1 = '0' + str(mtchy[ii])
```

```
  riga_norm = arr_r[ii]
  im = 0
  riga_norm1 = []
  while im <= m:
    riga_norm1.append(str(riga_norm[im]))
    im += 1
  riga_norm2 = " ".join(riga_norm1)
  gr_sg_txt = "G_" + gr1 + "_" + sg1 + " " + str(ix) + " " + riga_norm2
  arr_grsg.append(gr_sg_txt)
  ii += 1
arr_grsg.sort()
ii = 0
while ii < n_rows:
  arr_grsg_c.append(arr_grsg[ii])
  ii += 1

# setup of arr_cv matrix
num_gr = 0
gruppo0 = ""
ir = 0
while ir < n_rows:
  grsg_key = arr_grsg_c[ir].split()
  if not grsg_key[0] == gruppo0:
    gruppo0 = grsg_key[0]
    num_gr +=1
    ic = 1
    riga1 = []
    riga1.append(grsg_key[0])
    while ic <= m + 2:          # adding new columns for row mean and  n° of records
      riga1.append(0.0)
      ic += 1
    arr_cv.append(riga1)        # cv row
  ir += 1
riga1 = []
riga1.append("*Means*")   # adding new row for cv mean
ic = 1
while ic <= m + 2:               # adding new column for row mean and n° of records
  riga1.append(0.0)
  ic += 1
arr_cv.append(riga1)

def found(x):
  ir = 0
  while ir < len(arr_cv):
    linea_cv = arr_cv[ir]
    key_cv = linea_cv[0]
```

```
    if key_cv == x:
      return ir
    ir += 1

ir  = 0
irx = len(arr_grsg_c)
ic  = 3
linea_cv = arr_cv[0]
icx = len(linea_cv)
val_col = []

while ic < icx:
  ir = 0
  gruppo  = ""
  val_col = []
  while ir < irx:
    linea = arr_grsg_c[ir].split()
    if linea[0] == gruppo or gruppo == "":
      gruppo = linea[0]
      val_col.append(float(linea[ic]))
    else:
      i_gruppo = found(gruppo)
      linea_cv = arr_cv[i_gruppo]
      media_v = abs(mean(val_col))
      if media_v == 0.0:
         media_v = 0.0000000001
      std_v = sd(val_col)
      cv_v  = std_v / media_v
      linea_cv[ic-2] = cv_v                     # cv value
      linea_cv[len(linea_cv)-1] = len(val_col)  # number of records
      val_col = []
      val_col.append(float(linea[ic]))
      gruppo = linea[0]
    ir += 1
  i_gruppo = found(gruppo)
  linea_cv = arr_cv[i_gruppo]
  media_v = abs(mean(val_col))
  if media_v == 0.0:
    media_v = 0.0000000001
  std_v = sd(val_col)
  cv_v  = std_v / media_v
  linea_cv[ic-2] = cv_v                         # cv value
  linea_cv[len(linea_cv)-1] = len(val_col)      # number of records
  ic += 1
ir  = 0
irx = len(arr_cv)
```

```
linea_cv = arr_cv[0]
icx = len(linea_cv) - 2
ic  = 1
num_rec1 = 0

while ir < irx:                                  # rows mean
  media_riga = 0.0
  ic = 1
  num_col1 = 0
  linea_cv = arr_cv[ir]
  while ic < icx:
    media_riga += float(linea_cv[ic])
    num_col1 += 1
    ic += 1
  linea_cv[icx] = media_riga / num_col1
  num_rec1 += linea_cv[icx + 1]
  ir += 1
ir  = 0
ic  = 1

while ic < icx:                  # weighted mean of columns
  media_col = 0.0
  ir = 0
  num_rec1 = 0
  while ir < irx - 1:
    linea_cv = arr_cv[ir]
    media_col = media_col + linea_cv[ic] * linea_cv[icx+1]  # linea_cv[icx+1] = number
of records
    num_rec1 = num_rec1 + linea_cv[icx+1]
    ir += 1
  linea_cv = arr_cv[irx - 1]
  linea_cv[ic] = media_col / num_rec1
  ic += 1

# updating mean of the row
linea_cv = arr_cv[irx - 1]
linea_means = linea_cv[1:icx]
media_riga  = mean(linea_means)
linea_cv[icx] = media_riga        # Total mean
linea_cv[icx + 1] = num_rec1      # n° of records
cv_media_gen_after = str(media_riga)
cv_media_gen_after = cv_media_gen_after[0:6]

# write cv  file
testata_cv = testata
testata_cv[0] = "*Groups*"
```

```
testata_cv.append("*Mean*")
testata_cv.append("N_recs")
arch_cv_file = open(arch_cv, 'w')
ic = 0
while ic <= icx + 1:
  arch_cv_file.write('%s %s ' % (testata_cv[ic], " "*(9-len(testata_cv[ic]))))
  ic += 1
arch_cv_file.write('%s' % ('\n'))
ir = 0
while ir < irx:
  ic = 0
  linea_cv = arr_cv[ir]
  while ic <= icx + 1:
    if ic == 0:
      arch_cv_file.write('%s %s ' % (linea_cv[0], "  "))
    else:
      if ic <= icx:
        arch_cv_file.write('%7.4f %s ' % (linea_cv[ic], "  "))
      else:
        arch_cv_file.write('%6i %s ' % (linea_cv[ic], "  "))
    ic += 1
  arch_cv_file.write('%s' % ("\n"))
  ir += 1
ic = 0

media_xcv = mean(xcv[1:icx])

while ic <= icx :   # print CV input (before catalogue)
  if ic == 0:
    arch_cv_file.write('%s %s ' % ("*CVinp*", "  "))
  else:
    if ic < icx:
      arch_cv_file.write('%7.4f %s ' % (xcv[ic], "  "))
    else:
      arch_cv_file.write('%7.4f %s ' % (media_xcv, "  "))
      arch_cv_file.write('%6i %s ' % (linea_cv[ic+1], "  "))
  ic += 1
arch_cv_file.write('%s' % ("\n"))
#=========istruzioni aggiunte Roberto Bello 29/02/2012======================
#know_index = str(1.0 - float(cv_media_gen_after) / float(str_med_cv_gen))
#know_index = know_index[0:6]
#arch_cv_file.write('%s %s %s' % ('*KIndex*   ', know_index, '\n'))
#=========fine istruzioni aggiunte da Roberto Bello 29/02/2012==============
arch_cv_file.close()

# writing out catalog file
```

```
testata_cat1 = []
testata_cat1.append("*Group*")
arch_output_file = open(arch_output, 'w')
ic= 0
while ic < icx:
  testata_cat1.append(testata_cat[ic])
  ic += 1
ic= 0
while ic < len(testata_cat1):
  arch_output_file.write('%s %s ' % (testata_cat1[ic], " "*(15-len(testata_cat1[ic]))))
  ic += 1
arch_output_file.write('%s ' % ("\n"))
index = 0
while index < len(arr_orig):
  riga_orig = arr_orig[index]
  ic = 0
  while ic < len(riga_orig):
    if not(isinstance(riga_orig[ic],str)):
      riga_orig[ic] = str(riga_orig[ic])
    ic += 1
  # place before 0 if gr / sg < 10
  gr1 = str(mtchx[index])
  if mtchx[index] < 10:
    gr1 = '0' + str(mtchx[index])
  sg1 = str(mtchy[index])
  if mtchy[index] < 10:
    sg1 = '0' + str(mtchy[index])
  arr_rig0 = "G_" + gr1 + "_" + sg1 + " "*8
  arch_output_file.write('%s ' % (arr_rig0))
  ic= 0
  while ic < len(riga_orig):
    arch_output_file.write('%s %s ' % (riga_orig[ic], " "*(15-len(riga_orig[ic]))))
    ic += 1
  arch_output_file.write('%s ' % ("\n"))
  index += 1
testata_cat1 = []
testata_cat1.append("*Group*")
testata_cat1.append("*RecNum*")
arch_sort_file = open(arch_sort, 'w')
ic= 0
while ic < icx:
  testata_cat1.append(testata_cat[ic])
  ic += 1
ic= 0
while ic < len(testata_cat1):
  arch_sort_file.write('%s %s ' % (testata_cat1[ic], " "*(15-len(testata_cat1[ic]))))
```

```
  ic += 1
arch_sort_file.write('%s ' % ("\n"))
index = 0
while index < len(arr_grsg_c):
  riga_grsg = arr_grsg_c[index].split()
  ic = 0
  while ic < len(riga_grsg):
    val_txt = riga_grsg[ic]
    val_txt = val_txt[0:13]
    arch_sort_file.write('%s %s ' % (val_txt, " "*(15-len(val_txt))))
    ic += 1
  if index < len(arr_grsg_c) - 1:
    arch_sort_file.write('%s ' % ("\n"))
  index += 1
arch_sort_file.close()

# writing out catalog and sorted file
arr_outsrt = []
index = 0
while index < len(arr_orig):
  riga_sort = []
  # place before 0 if gr / sg < 10
  gr1 = str(mtchx[index])
  if mtchx[index] < 10:
    gr1 = '0' + str(mtchx[index])
  sg1 = str(mtchy[index])
  if mtchy[index] < 10:
    sg1 = '0' + str(mtchy[index])
  riga_sort.append("G_" + gr1 + "_" + sg1)
  ic = 0
  riga_orig = arr_orig[index]
  while ic < len(riga_orig):
    val_riga = riga_orig[ic]
    riga_sort.append(val_riga)
    ic += 1
  arr_outsrt.append(riga_sort)
  index += 1

for line in arr_outsrt:
  line = "".join(line)

arr_outsrt.sort()

testata_srt = []
testata_srt.append("*Group*")
arch_outsrt_file = open(arch_outsrt, 'w')
```

```
ic= 0
while ic < icx:
  testata_srt.append(testata_orig[ic])
  ic += 1
ic= 0
while ic < len(testata_srt):
  arch_outsrt_file.write('%s %s' % (testata_srt[ic], " "*(15-len(testata_srt[ic]))))
  ic += 1
arch_outsrt_file.write('%s' % ("\n"))
index = 0
key_gruppo = ""
while index < len(arr_outsrt):
  riga_sort = arr_outsrt[index]
  index_c = 0
  while index_c < len(riga_sort):
    if index_c == 0:
      if riga_sort[0] != key_gruppo:
        # arch_outsrt_file.write('%s ' % ("\n"))
        key_gruppo = riga_sort[0]
    valore = riga_sort[index_c]
    arch_outsrt_file.write('%s %s' % (valore, " "*(15-len(valore))))
    index_c += 1
  if index < len(arr_grsg_c) - 1:
    arch_outsrt_file.write('%s' % ("\n"))
  index += 1
arch_outsrt_file.close()

print("################################################################################")
print("# KB_CAT KNOWLEDGE DISCOVERY IN DATA MINING (CATALOG PROGRAM)                 #")
print("# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                 #")
print("# Language used: PYTHON                                                       #")
print("################################################################################")

arch_log_file = open(arch_log, 'w')
arch_log_file.write("%s %s" %
("############################################################################", "\n"))
arch_log_file.write("%s %s" % ("# KB_CAT KNOWLEDGE DISCOVERY IN DATA MINING (CATALOG
PROGRAM)              #", "\n"))
arch_log_file.write("%s %s" % ("# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS
RESERVED)              #", "\n"))
arch_log_file.write("%s %s" % ("# Language used: PYTHON                     .
#", "\n"))
arch_log_file.write("%s %s" %
("############################################################################", "\n"))
arch_log_file.write("%s %s %s" % ("Input File                                   ->
", file_input, "\n"))
arch_log_file.write("%s %s %s" % ("Numer of Groups (3 - 20)                     ->
", str(gruppi_num), "\n"))
```

```
arch_log_file.write("%s %s %s" % ("Normalization (Max, Std, None)                ->
", tipo_norm, "\n"))
arch_log_file.write("%s %s %s" % ("Start Value of alpha (from 1.8 to 0.9)        ->
", str(alpha_max), "\n"))
arch_log_file.write("%s %s %s" % ("End Value of alpha (from 0.5 to 0.0001)       ->
", str(alpha_min), "\n"))
arch_log_file.write("%s %s %s" % ("Decreasing step of alpha (from 0.1 to 0.001)  ->
", str(alpha_step), "\n"))
arch_log_file.write("%s"      %
("=========================OUTPUT=========================================================
\n"))
arch_log_file.write("%s %s %s" % ("Output File Catalog.original     ", arch_output,
"\n"))
arch_log_file.write("%s %s %s" % ("Output File Catalog.sort         ", arch_outsrt,
"\n"))
arch_log_file.write("%s %s %s" % ("Output File Summary sort         ", arch_sort, "\n"))
arch_log_file.write("%s %s %s" % ("Output File Matrix Catal.        ", arch_catal,
"\n"))
arch_log_file.write("%s %s %s" % ("Output File Means, STD, CV.      ", arch_medsd,
"\n"))
arch_log_file.write("%s %s %s" % ("Output File CV of the Groups     ", arch_cv, "\n"))
arch_log_file.write("%s %s %s" % ("Output File Training Grid        ", arch_grid, "\n"))
arch_log_file.write("%s %s %s" % ("Output File Run Parameters       ", arch_log, "\n"))
#=========istruzioni aggiunte Roberto Bello 29/02/2012=====================
know_index = str(1.0 - float(cv_media_gen_after) / float(str_med_cv_gen))
know_index = know_index[0:6]
arch_log_file.write('%s %s %s' % ('*KIndex*   ', know_index, '\n'))
#=========fine istruzioni aggiunte da Roberto Bello 29/02/2012==============

min_err_txt = "%12.3f" % min_err      # format 8 integer and 3 decimals
alpha_txt  = "%12.5f" % alpha         # format 6 integer and 5 decimals
alpha_min_txt = "%12.5f" % alpha_min  # format 6 integer and 5 decimals

print
if min_err == 1000000000.000:
  print("Oops! No result. Try again with new alpha parameters")
print
print ("EPOCH " + str(min_epok -1) + "   WITH MIN ERROR " + min_err_txt +
  " starting alpha " + alpha_min_txt + "   ending alpha " + alpha_txt +
  " Iterations " + str(iter) + " Total Epochs " + str(ne - 1))
print
print 'Output File Catalog.original ' + arch_output
print 'Output File Catalog.sort     ' + arch_outsrt
print 'Output File Summary sort     ' + arch_sort
print 'Output File Matrix Catal.    ' + arch_catal
print 'Output File Means, STD, CV.  ' + arch_medsd
print 'Output File CV of the Groups ' + arch_cv
print 'Output File Training Grid    ' + arch_grid
print 'Output File Run Parameters   ' + arch_log
```

```
print 'CV before Catalog            ' + str_med_cv_gen
print 'CV after Catalog             ' + cv_media_gen_after
know_index = str(1.0 - float(cv_media_gen_after) / float(str_med_cv_gen))
know_index = know_index[0:6]
print 'Knowledge Index              ' + know_index
print

# Elapsed time
t1 = datetime.datetime.now()
elapsed_time = t1 - t0
print "Elapsed time (seconds)   :   " + str(elapsed_time.seconds)
print
```

Appendix 2 – KB_STA source

```
# -*- coding: utf-8 -*-
###############################################################################
# KB_STA KNOWLEDGE DISCOVERY IN DATA MINING (STATISTICAL PROGRAM)             #
# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                 #
# Language used: PYTHON                               .                       #
###############################################################################
import os
import random
import copy
import datetime

def fp_conversion(value):     # from string containing number to float
    try:
        return float(value)
    except ValueError:
        return (value)

def count(s, e):          # frequencies count
      return len([x for x in s if (x == e)])

print("###############################################################################")
print("# KB_STA KNOWLEDGE DISCOVERY IN DATA MINING (STATISTICAL PROGRAM)             #")
print("# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                 #")
print("# Language used: PYTHON                               .                       #")
print("###############################################################################")

# input / output files and run parameters
error = 0

while True:
  file_input = raw_input('Cataloged Records File (_outsrt.txt)       : ')
```

```
  if not os.path.isfile(file_input):
    print("Oops! File does not exist. Try again... or CTR/C to exit")
  else:
    break

while True:
  file_gruppi = raw_input('Groups / CV File (_cv.txt)                : ')
  if not os.path.isfile(file_gruppi):
    print("Oops! File does not exist. Try again... or CTR/C to exit")
  else:
    break

while True:
  file_rappor = raw_input('Report File (output)                      : ')
  if os.path.isfile(file_rappor):
    print("Oops! File exist. Try again... or CTR/C to exit")
  else:
    break

while True:
  try:
    omog_perc = int(raw_input("Group Consistency (% from 0 to 100)       : "))
  except ValueError:
    print("Oops!  That was no valid number.  Try again...")
  else:
    if(omog_perc < 0):
      print("Oops! Group Consistency too low. Try again...")
    else:
      if(omog_perc > 100):
        print("Oops! Group Consistency too big. Try again...")
      else:
        break

while True:
  try:
    omog_vari = int(raw_input("Variable Consistency (% from 0 to 100)    : "))
  except ValueError:
    print("Oops!  That was no valid number.  Try again...")
  else:
    if(omog_vari < 0):
      print("Oops! Variable Consistency too low. Try again...")
    else:
      if(omog_vari > 100):
        print("Oops! Variable Consistency too big. Try again...")
      else:
        break
```

```
while True:
  try:
    rec_min = int(raw_input("Select groups containing records >=        : "))
  except ValueError:
    print("Oops!  That was no valid number.  Try again...")
  else:
    if(rec_min < 1):
      print("Oops! Number of records too low. Try again...")
    else:
      break

while True:
  try:
    rec_max = int(raw_input("Select groups containing records <=        : "))
  except ValueError:
    print("Oops!  That was no valid number.  Try again...")
  else:
    if(rec_max < 1):
      print("Oops! Number of records too low. Try again...")
    if (rec_max < rec_min):
      print("Oops! Number of records must be >= " + str(rec_min) + " Try again...")
    else:
      break

while True:
  est_rapp   = raw_input("Summary / Detail report (S / D)            : ")
  est_rapp   = est_rapp.upper()
  est_rapp   = est_rapp[0]
  if(est_rapp <> 'S' and est_rapp <> 'D'):
    print("Oops! Input S, D. Try again...")
  else:
    break

inp_rapp = "N"
if est_rapp == "D" or est_rapp == "d":
  while True:
    inp_rapp   = raw_input("Display Input Records (Y / N)              : ")
    inp_rapp   = inp_rapp.upper()
    inp_rapp   = inp_rapp[0]
    if(inp_rapp <> 'Y' and inp_rapp <> 'N'):
      print("Oops! Input Y, N. Try again...")
    else:
      break

# start time
```

```
t0 = datetime.datetime.now()

# initial setup
arr_r    = []              # input rows
arr_c    = []              # row list of arr_c
xnomi    = []              # headings row
len_var  = []       # max string lenght of variable

# the numbers of variables / columns in all record must be the same
n_rows = 0
n_cols = 0
err_cols = 0
index = 0
file_log = file_input + "_log.txt"
for line in open(file_input).readlines():
  linea = line.split()
  if(index == 0):
    xnomi.append(linea)
    n_cols = len(linea)
  else:
    arr_r.append(linea)
    if(len(linea) != n_cols):
      err_cols = 1
      print("Different numbers of variables / columns in the record " + str(index)
        + " cols " + str(len(linea)))
  index += 1
if(err_cols == 1):
  print("File " + file_input + " contains errors. Exit ")
  quit()
index = 0
while index < len(arr_r):
  linea = arr_r[index]
  index_c = 0
  while index_c < len(linea):         # converting strings containing numbers to float
    linea[index_c] = fp_conversion(linea[index_c])
    index_c += 1
  arr_r[index] = linea
  index += 1
testata = xnomi[0]
n_cols = len(testata) - 1
n_rows = len(arr_r)
index = 0
while index < len(testata):            #      finding max string len (for the report)
  len_var.append(len(testata[index]))
  index += 1
index = 0
```

```
while index < len(arr_r):
  linea = arr_r[index]
  index_c = 0
  while index_c < len(linea):
    if isinstance(linea[index_c],basestring):   # text
      len_campo = len(linea[index_c])
    else:                              # number
      len_campo = len(str(linea[index_c]))
    if len_campo > len_var[index_c]:
      len_var[index_c] = len_campo
    index_c += 1
  index += 1
max_len = max(len_var)
arr_cv = []
testata_cv = []
index = 0

# reading Groups / CV file
for line in open(file_gruppi).readlines():
  linea = line.split()
  if(index == 0):
    n_cols = len(linea)
    testata_cv.append(linea)
  else:
    arr_cv.append(linea)
    if(len(linea) != n_cols):
      err_cols = 1
      print("Different numbers of variables / columns in the record " + str(index)
        + " cols " + str(len(linea)))
  index += 1

if(err_cols == 1):
  print("File " + file_gruppi + " contains errors. Exit ")
  quit()

for line in arr_cv:
  index_c = 0
  linea = line
  while index_c < len(linea):    # converting strings containing numbers to float
    linea[index_c] = fp_conversion(linea[index_c])
    index_c += 1

ind_fine = len(arr_cv)            # last value of arr_cv
ind_fine = ind_fine - 1
arr_c = arr_cv[ind_fine]          # row of totals
tot_omogen = float(arr_c[-2])     # consistency totals
```

```
tot_record = float(arr_c[-1])    # total of records


arch_rappor = open(file_rappor, 'w')
index = 0
ind_fine = len(arr_cv)
ind_fine = ind_fine - 2          # row of CV Totals
testata_cv = testata_cv[0]
testata_cv = testata_cv[:-2]      # removing the last two columns
arch_rappor.write("%s %s %s %s %s" % ("KB_STA - Statistical Analysis from: ",
file_input, " and from: ", file_gruppi, "\n"))
arch_rappor.write("%s %s %s %s %s %s %s %s %s" % (("Min Perc. of group Consistency: ",
str(omog_perc), " Min Perc. of variable Consistency: ",
  str(omog_vari), "\nMin Number of records: " , str(rec_min), " Max Number of records: "
, str(rec_max),"\n")))
arch_rappor.write("%s " % ("by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS
RESERVED) \n"))


while index < len(arr_cv):
    arr_c = arr_cv[index]
    intero =  int(arr_c[-1])       # totals of records
    perc_omogen = 0.0
    if not tot_omogen == 0:
      perc_omogen = (arr_c[-2] * 100.0) / tot_omogen
    perc_omog = 100 - int(perc_omogen)
    if perc_omog < 0.0:
      perc_omog = 0.0
    perc_rec  = intero * 100.0 / tot_record
    if (perc_omog >= omog_perc and rec_min <= intero and rec_max >= intero) or arr_c[0]
== "*Means*":
      arch_rappor.write("%s " %
("===================================================================================
===\n"))
      arch_rappor.write(("%s %s %.4f %s %3s %s %5s %s %7.2f %s " % (arr_c[0], "
Consistency ", arr_c[-2], "   %Consistency ",
        str(perc_omog), "   Records ", str(intero), "   %Records ", perc_rec, "\n")))
      ind_c = 0
      cod_gruppo = arr_c[0]
      while ind_c < len(arr_c) - 3:
        omogen_perc = 0.0
        if float(arr_c[ind_c +1]) == 0.0:
          arr_c[ind_c] = "0.00000"
        if not arr_c[-2] <= 0.0:
          omogen_perc = 100.0 - arr_c[ind_c +1] * 100.0 / arr_c[-2] # CV of group
variabile divided by CV of the group
        else:
          omogen_perc = 100.0
        if omogen_perc <= 0.0:
          omogen_perc = 0.0
        if omogen_perc >= omog_vari and (est_rapp == "d" or est_rapp == "D"): #
```

```
consistency value >= min parameter
          arch_rappor.write("%s %s %s %10.4f %s %10.2f %s" % (("*** ", testata_cv[ind_c
+ 1] +
           " " * (max_len - len(testata_cv[ind_c + 1])) , "Consistency\t",
           float(arr_c[ind_c + 1]),"\t%Consistency\t",omogen_perc, "\n")))

          # computing variables frequencies and quartiles

          # 1) variables frequencies
          ind_sort = 0
          arr_temp = []     # variable array of records included in the group
          ind_temp = 0
          while ind_sort < len(arr_r): # list of variable values in the group
            linea = arr_r[ind_sort]
            if linea[0].strip() == cod_gruppo:
              arr_temp.append(linea[ind_c + 2])
              if (est_rapp == "d" or est_rapp) == "D" and (inp_rapp == "y" or inp_rapp
== "Y"):
                arch_rappor.write(("%s %s %s %s %s %s" % (linea[0], "\tID record\t",
str(linea[1]) +
                " " * (max_len - len(str(linea[1]))), "Value", linea[ind_c + 2], "\n")))
              ind_temp += 1
            ind_sort += 1

          # 2) converting strings containing numbers to float
          ind_temp = 0
          tipo_num = 0
          tipo_txt = 0
          while ind_temp < len(arr_temp):  # texts or numbers
            arr_temp[ind_temp] = fp_conversion(arr_temp[ind_temp])
            if isinstance(arr_temp[ind_temp],basestring):
               tipo_txt = 1
            else:
               tipo_num = 1
            ind_temp += 1
          if tipo_num == 1 and tipo_txt == 1:
            print "The columns / variable " + testata[ind_c] + " contains both strings
and numbers. Exit. "
            quit()
          if tipo_num == 0:    # the variable is a text
            arr_temp.sort()

            # 3) computing frequencies
            key1 = ""
            key_freq = []          # keys and frequencies
           arr_t_index = 0
            while arr_t_index < len(arr_temp):
```

```
            if arr_temp[arr_t_index] <> key1:
              kf_valore = []
              kf_valore.append(arr_temp.count(arr_temp[arr_t_index]))
              kf_valore.append(arr_temp[arr_t_index])
              key_freq.append(kf_valore)
              key1 = arr_temp[arr_t_index]
            arr_t_index += 1

          key_freq.sort()       # frequencies ascending sort
          key_freq.reverse()   # frequencies descending sort

          ris_out_index = 0
          while ris_out_index < len(key_freq) and (est_rapp == "d" or est_rapp ==
"D"):
            kf_valore = key_freq[ris_out_index]
            arch_rappor.write("%s %s %s %7i %s %.2f %s" % (("Value\t", kf_valore[1] +
           " " * (max_len - len(kf_valore[1])),
              "Frequency\t", kf_valore[0],"\tPercentage\t",
kf_valore[0]*100/len(arr_temp), "\n")))
            ris_out_index += 1
        if tipo_txt == 0:       # the variabile is a number
          # computing means
          if len(arr_temp) > 0:
            mean_arr = sum(arr_temp)/len(arr_temp)
          # computing the step of quartiles
          arr_temp.sort()
          if len(arr_temp) > 0:
            minimo = arr_temp[0]
            massimo = arr_temp[len(arr_temp) - 1]
            passo = (float(massimo) - float(minimo)) / 4.0
            q1 = minimo + passo
            q2 = q1 + passo
            q3 = q2 + passo
            q4 = q3 + passo
            fr1 = 0.0     # first quartile
            fr2 = 0.0     # second quartile
            fr3 = 0.0     # third quartile
            fr4 = 0.0     # fourth quartile
            arr_index = 0
            while arr_index < len(arr_temp):
              if arr_temp[arr_index] <= q1:
                fr1 += 1
              elif arr_temp[arr_index] <= q2:
                fr2 += 1
              elif arr_temp[arr_index] <= q3:
                fr3 += 1
```

```
                else:
                  fr4 += 1
                arr_index += 1
              records = len(arr_temp)
              p1 = fr1 * 100 / records
              p2 = fr2 * 100 / records
              p3 = fr3 * 100 / records
              p4 = fr4 * 100 / records
              if (est_rapp == "d" or est_rapp == "D"):
                arch_rappor.write("%s %.2f %s %.2f %s %.2f %s %.2f %s" % ("Mean\t",
mean_arr,"Min\t", minimo, "\tMax\t", massimo,"\tStep\t", passo,    "\n"))
                if p1 > 0.0:
                  arch_rappor.write(("%s %10.2f %s %7.2f %s" % ("First  Quartile (end)
", q1,
                    "  Frequency %\t", p1, "\n")))
                if p2 > 0.0:
                  arch_rappor.write(("%s %10.2f %s %7.2f %s" % ("Second Quartile (end)
", q2,
                    "  Frequency %\t", p2, "\n")))
                if p3 > 0.0:
                  arch_rappor.write(("%s %10.2f %s %7.2f %s" % ("Third  Quartile (end)
", q3,
                    "  Frequency %\t", p3, "\n")))
                if p4 > 0.0 :
                  arch_rappor.write(("%s %10.2f %s %7.2f %s" % ("Fourth Quartile (end)
", q4,
                    "  Frequency %\t", p4, "\n")))
        ind_c += 1
    index += 1
arch_rappor.close()


arch_log = open(file_log, 'w')
arch_log.write("%s %s" %
("#################################################################################",
"\n"))
arch_log.write("%s %s" % ("# KB_STA KNOWLEDGE DISCOVERY IN DATA MINING (STATISTICAL
PROGRAM)               #", "\n"))
arch_log.write("%s %s" % ("# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)
#", "\n"))
arch_log.write("%s %s" % ("# Language used: PYTHON
#", "\n"))
arch_log.write("%s %s" %
("#################################################################################",
"\n"))
arch_log.write("%s %s %s" % ("INPUT - Cataloged Records File (_outsrt.txt)
-> ", file_input, "\n"))
arch_log.write("%s %s %s" % ("INPUT - Groups / CV File (_cv.txt)
-> ", file_gruppi, "\n"))
arch_log.write("%s %s %s" % ("Group Consistency (% from 0 to 100)
-> ", str(omog_perc), "\n"))
arch_log.write("%s %s %s" % ("Variable Consistency (% from 0 to 100)
```

```
-> ", str(omog_vari), "\n"))
arch_log.write("%s %s %s" % ("Select groups containing records >=
-> ", str(rec_min), "\n"))
arch_log.write("%s %s %s" % ("Select groups containing records <=
-> ", str(rec_max), "\n"))
arch_log.write("%s %s %s" % ("Summary / Detail report (S / D)
-> ", est_rapp, "\n"))
arch_log.write("%s %s %s" % ("Display Input Records (Y / N)
-> ", inp_rapp, "\n"))
arch_log.write("%s %s" %
("========================OUTPUT==============================================",
"\n"))
arch_log.write("%s %s %s" % ("Report File
-> ", file_rappor, "\n"))
arch_log.close()

# Elapsed time
t1 = datetime.datetime.now()
elapsed_time = t1 - t0
print "Elapsed time (seconds)   :   " + str(elapsed_time.seconds) + "." +
str(elapsed_time.microseconds)
print
```

Appendix 3 – KB_CLA source

```
# -*- coding: utf-8 -*-
###############################################################################
# KB_CLA KNOWLEDGE DISCOVERY IN DATA MINING (CLASSIFY PROGRAM)               #
# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                #
# Language used: PYTHON                              .                       #
###############################################################################
import os
import random
import copy
import datetime
def mean(x):        # mean
  n = len(x)
  mean = sum(x) / n
  return mean

def sd(x):          # standard deviattion
  n = len(x)
  mean = sum(x) / n
  sd = (sum((x-mean)**2 for x in x) / n) ** 0.5
  return sd

class ndim:             # from 3D array to flat array
    def __init__(self,x,y,z,d):
        self.dimensions=[x,y,z]
```

```
        self.numdimensions=d
        self.gridsize=x*y*z

    def getcellindex(self, location):
        cindex = 0
        cdrop = self.gridsize
        for index in xrange(self.numdimensions):
            cdrop /= self.dimensions[index]
            cindex += cdrop * location[index]
        return cindex

    def getlocation(self, cellindex):
        res = []
        for size in reversed(self.dimensions):
            res.append(cellindex % size)
            cellindex /= size
        return res[::-1]

""" how to use ndim class
n=ndim(4,4,5,3)
print n.getcellindex((0,0,0))
print n.getcellindex((0,0,1))
print n.getcellindex((0,1,0))
print n.getcellindex((1,0,0))

print n.getlocation(20)
print n.getlocation(5)
print n.getlocation(1)
print n.getlocation(0)
"""

print("###############################################################################")
print("# KB_CLA KNOWLEDGE DISCOVERY IN DATA MINING (CLASSIFY PROGRAM)                #")
print("# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                 #")
print("# Language used: PYTHON                                                       #")
print("###############################################################################")

# input and run parameters
error = 0

while True:
  arch_input = raw_input('InputFile                              : ')
  if not os.path.isfile(arch_input):
    print("Oops! File does not exist. Try again... or CTR/C to exit")
  else:
    break
```

```
while True:
  try:
    num_gruppi = int(raw_input('Number of Groups (3 - 20)              : '))
  except ValueError:
    print("Oops!  That was no valid number.  Try again...")
  else:
    if(num_gruppi < 3):
      print("Oops! Number of Groups too low. Try again...")
    else:
      if(num_gruppi > 20):
        print("Oops! Number of Groups too big. Try again...")
      else:
        break

while True:
  normaliz   = raw_input('Normalization(Max, Std, None)          : ')
  normaliz   = normaliz.upper()
  normaliz   = normaliz[0]
  if(normaliz <> 'M' and normaliz <> 'S' and normaliz <> 'N'):
    print("Oops! Input M, S or N. Try again...")
  else:
    break

while True:
  arch_grid = raw_input('File Training Grid                      : ')
  if not os.path.isfile(arch_grid):
    print("Oops! File does not exist. Try again... or CTR/C to exit")
  else:
    break

file_input   = arch_input
gruppi_num   = num_gruppi
tipo_norm    = normaliz

# outputs files
file_input   = arch_input
tipo_norm    = normaliz
gruppi_num   = num_gruppi
nome_input   = file_input.split(".")
arch_output  = nome_input[0] + "_" + "C" + tipo_norm + "_g" + str(gruppi_num) +
"_out.txt"
arch_outsrt  = nome_input[0] + "_" + "C" + tipo_norm + "_g" + str(gruppi_num) +
"_outsrt.txt"
arch_sort    = nome_input[0] + "_" + "C" + tipo_norm + "_g" + str(gruppi_num) +
"_sort.txt"
arch_catal   = nome_input[0] + "_" + "C" + tipo_norm + "_g" + str(gruppi_num) +
```

```
"_catal.txt"
arch_medsd   = nome_input[0] + "_" + "C" + tipo_norm + "_g" + str(gruppi_num) +
"_medsd.txt"
arch_cv      = nome_input[0] + "_" + "C" + tipo_norm + "_g" + str(gruppi_num) +
"_cv.txt"
arch_log     = nome_input[0] + "_" + "C" + tipo_norm + "_g" + str(gruppi_num) +
"_log.txt"

# start time
t0 = datetime.datetime.now()

# read input file
arr_r    = []
arr_orig = []
arr_c    = []
mtchx    = []
mtchy    = []
txt_col  = []
xnomi    = []

# the numbers of variables / columns in all record must be the same
n_rows = 0
n_cols = 0
err_cols = 0
index = 0
for line in open(file_input).readlines():
  linea = line.split()
  if(index == 0):
    xnomi.append(linea)
    n_cols = len(linea)
  else:
    arr_r.append(linea)
    if(len(linea) != n_cols):
      err_cols = 1
      print("Different numbers of variables / columns in the record " + str(index)
        + " cols " + str(len(linea)))
  index += 1
if(err_cols == 1):
  print("File " + file_input + " contains errors. Exit ")
  quit()
index = 0
while index < len(arr_r):
  linea = arr_r[index]
  index_c = 0
  while index_c < len(linea):
    if linea[index_c].isdigit():
      linea[index_c] = float(linea[index_c])
```

```
    index_c += 1
  arr_r[index] = linea
  index += 1
arr_orig = copy.deepcopy(arr_r)        # original input file
testata_cat = copy.deepcopy(xnomi[0])  # original header row

# finding columns containing strings and columns containing numbers
testata = xnomi[0]
testata_orig = copy.deepcopy(xnomi[0])
n_cols = len(testata) - 1
n_rows = len(arr_r)
ind_c  = 1
err_type = 0
while ind_c < len(testata):
  ind_r    = 1
  tipo_num = 0
  tipo_txt = 0
  while ind_r < len(arr_r):
    arr_c = arr_r[ind_r]
    if isinstance(arr_c[ind_c],basestring):
      tipo_txt = 1
    else:
      tipo_num = 1
    ind_r += 1
  if tipo_num == 1 and tipo_txt == 1:
    print "The columns / variables " + testata[ind_c] + " contains both strings and
numbers."
    err_type = 1
  ind_c += 1
if err_type == 1:
  print "Oops! The columns / variables contains both strings and numbers. Exit. "
  quit()

index_c = 1
while index_c <= n_cols:
  txt_col = []
  index = 0
  while index < len(arr_r):
    arr_c = arr_r[index]
    if(isinstance(arr_c[index_c],str)):
      txt_col.append(arr_c[index_c])
    index += 1
  set_txt_col = set(txt_col)            # remove duplicates
  txt_col = list(set(set_txt_col))
  txt_col.sort()
```

```
  # from strings to numbers
  if(len(txt_col) > 0):
    if(len(txt_col) > 1):
      passo1 = 1.0 / (len(txt_col) - 1)
    else:
      passo1 = 0.0
    index = 0
    while index < len(arr_r):
      arr_c = arr_r[index]
      campo1 = arr_c[index_c]
      indice1 = txt_col.index(campo1)
      if(len(txt_col) == 1):  # same values in the column
        val_num1 = float(1)
      else:
        val_num1 = float(passo1 * indice1)
      arr_c[index_c] = val_num1 + 0.00000001   # to avoid zero values in means
                                               # (to prevent zero divide in CV)
      index += 1
  index_c += 1

# means, max & std
xmeans = []
xmaxs  = []
xmins  = []            ### aggiunto Roberto 4/03/2012
xsds   = []
xcv    = []
index_c = 0
while index_c <= n_cols:
  xmeans.append(0.0)
  xmaxs.append(-9999999999999999.9)
  xmins.append(9999999999999999.9)     ### aggiunto Roberto 4/03/2012
  xsds.append(0.0)
  xcv.append(0.0)
  index_c += 1

# means & max
index = 0
while index < n_rows:
  arr_c = arr_r[index]
  index_c = 1
  while index_c <= n_cols:
    xmeans[index_c] += arr_c[index_c]
    if(arr_c[index_c] > xmaxs[index_c]):
      xmaxs[index_c] = arr_c[index_c]
    index_c += 1
  index += 1
```

```
index_c = 1
while index_c <= n_cols:
  xmeans[index_c] = xmeans[index_c] / n_rows
  index_c += 1

# std
index = 0
while index < n_rows:
  arr_c = arr_r[index]
  index_c = 1
  while index_c <= n_cols:
    xsds[index_c] += (arr_c[index_c] - xmeans[index_c])**2
    index_c += 1
  index += 1
index_c = 1

while index_c <= n_cols:
  xsds[index_c] = (xsds[index_c] / (n_cols - 1)) ** 0.5
  index_c += 1

# Means, Max, Std, CV output file
medsd_file = open(arch_medsd, 'w')

# columns names
index_c = 1
while index_c <= n_cols:
  medsd_file.write('%s %s %s ' % ('Col' + str(index_c), testata[index_c], "\t"))
  index_c += 1
medsd_file.write('%s' % ('\n'))

# means
index_c = 1
while index_c <= n_cols:
  valore = str(xmeans[index_c])
  valore = valore[0:6]
  medsd_file.write('%s %s %s ' % ('Mean' + str(index_c), valore, "\t"))
  index_c += 1
medsd_file.write('%s' % ('\n'))

# max
index_c = 1
while index_c <= n_cols:
  valore = str(xmaxs[index_c])
  valore = valore[0:6]
  medsd_file.write('%s %s %s ' % ('Max' + str(index_c), valore, "\t"))
  index_c += 1
```

```
medsd_file.write('%s' % ('\n'))

# std
index_c = 1
while index_c <= n_cols:
  valore = str(xsds[index_c])
  valore = valore[0:6]
  medsd_file.write('%s %s %s ' % ('Std' + str(index_c), valore, "\t"))
  index_c += 1
medsd_file.write('%s' % ('\n'))
# CV
index_c = 1
med_cv_gen = 0.0          # cv average of all columns / variables
while index_c <= n_cols:
  if xmeans[index_c] == 0:
    media1 = 0.000001
  else:
    media1 = xmeans[index_c]
  xcv[index_c] = xsds[index_c] / abs(media1)
  valore = str(xcv[index_c])
  med_cv_gen += xcv[index_c]
  valore = valore[0:6]
  medsd_file.write('%s %s %s ' % ('CV_' + str(index_c), valore, "\t"))
  index_c += 1
med_cv_gen = med_cv_gen / n_cols
str_med_cv_gen = str(med_cv_gen)
str_med_cv_gen = str_med_cv_gen[0:6]
medsd_file.write('%s' % ('\n'))
medsd_file.close()

# input standardization

# standardization on max

if tipo_norm == 'M':
  index = 0
  while index < n_rows:
    arr_c = arr_r[index]
    index_c = 1
    while index_c <= n_cols:
      if xmaxs[index_c] == 0.0:
        xmaxs[index_c] = 0.00001
      arr_c[index_c] = arr_c[index_c] / xmaxs[index_c]
      index_c += 1
    index += 1
```

```
# standardization on std

if tipo_norm == 'S':
  index = 0
  while index < n_rows:
    arr_c = arr_r[index]
    index_c = 1
    while index_c <= n_cols:
      if xsds[index_c] == 0.0:
        xsds[index_c] = 0.00001
      arr_c[index_c] = (arr_c[index_c] - xmeans[index_c]) / xsds[index_c]
      if arr_c[index_c] < xmins[index_c]:     ### aggiunto Roberto 4/03/2012
        xmins[index_c] = arr_c[index_c]             ### aggiunto Roberto 4/03/2012
      index_c += 1
    index += 1
  # aggiungo xmins per eliminare i valori negativi (aggiunto da Roberto 4/03/2012)
  index = 0
  while index < n_rows:
    arr_c = arr_r[index]
    index_c = 1
    while index_c <= n_cols:
      arr_c[index_c] = arr_c[index_c] - xmins[index_c]
      index_c += 1
    index += 1
  # fine aggiunta da Roberto 4/03/2012

# start of kohonen algorithm
n = len(arr_r) - 1
m = len(arr_c) - 1
nx = gruppi_num
ny = gruppi_num
rmax = nx
rmin = 1.0
grid = []                      # training grid
index = 0
while index < nx * ny * m:
  grid.append(0.0)
  index += 1
n=ndim(nx,ny,m,3)

# carico la Grid di addestramento da arch_grid
for line in open(arch_grid).readlines():
  linea   = line.split()
  index   = int(linea[0])
  index_c = int(linea[1])
  index_k = int(linea[2])
```

```
  valore  = float(linea[3])
  ig = n.getcellindex((index,index_c,index_k))
  grid[ig] = valore

# from the starting row to the ending row
i = 0
while i < n_rows:
  # find the best grid coefficient
  ihit = 0
  jhit = 0
  dhit = 100000.0
  igx = 0
  igy = 0
  while igx < nx:
    igy = 0
    while igy < ny:
      d = 0.0
      neff = 0
      k = 0
      arr_c = arr_r[i]
      while k < m:   # update the sum of squared deviation of input
                     # value from the grid coefficient
        ig = n.getcellindex((igx,igy,k))
        d = d + (arr_c[k+1] - grid[ig]) ** 2
        k += 1
      d = d / float(m)
      #  d = d / m
      if d < dhit:
        dhit = d
        ihit = int(igx)
        jhit = int(igy)
      igy += 1
    igx += 1
  i += 1

# computing results

# catalog input by min grid
ii = 0
while ii < n_rows:
  ihit = 0
  jhit = 0
  dhit = 100000.0
  # from 1 to numbers of groups
  ir = 0
  while ir < nx:         # from 1 to numbers of groups
```

```
    jc = 0
    while jc < ny:       # from 1 to numbers of groups
      d = 0.0
      neff = 0
      k = 0
      while k < n_cols:  # update the sum of squared deviation of input
                         # value from the grid coefficient
        arr_c = arr_r[ii]
        ig = n.getcellindex((ir,jc,k))
        d = d + (arr_c[k+1] - grid[ig]) ** 2
        k += 1
      d = d / m
      if d < dhit:       # save the coordinates of the best coefficient
        dhit = d
        ihit = ir
        jhit = jc
      jc += 1
    ir += 1
  mtchx.append(ihit)
  mtchy.append(jhit)
  ii += 1

# write arch_catal file
arch_catal_file = open(arch_catal, 'w')
ii = 0
while ii < n_rows:
  arch_catal_file.write("%.6i %s %.6i %s %.6i %s" % (ii, ' ', mtchx[ii], ' ', mtchy[ii],
"\n"))
  ii += 1
arch_catal_file.close()

# matrix of statistics
arr_cv   = []            # CV array of the Groups and Total
arr_med  = []            # means array of the Groups
riga_cv  = []                  # CV row in arr_cv
arr_col  = []                  # group temporary array
arr_grsg = []                  # input data array (normalized)
arr_grsg_c = []              # copy of arr_grsg (for file out sort)

# input matrix sort in group sequence
ii = 0
ix = 0
while ii < n_rows:
  ix += 1
  gr1 = str(mtchx[ii])
  if mtchx[ii] < 10:
```

```
    gr1 = '0' + str(mtchx[ii])
  sg1 = str(mtchy[ii])
  if mtchy[ii] < 10:
    sg1 = '0' + str(mtchy[ii])
  riga_norm = arr_r[ii]
  im = 0
  riga_norm1 = []
  while im <= m:
    riga_norm1.append(str(riga_norm[im]))
    im += 1
  riga_norm2 = " ".join(riga_norm1)
  gr_sg_txt = "G_" + gr1 + "_" + sg1 + " " + str(ix) + " " + riga_norm2
  arr_grsg.append(gr_sg_txt)
  ii += 1
arr_grsg.sort()
ii = 0
while ii < n_rows:
  arr_grsg_c.append(arr_grsg[ii])
  ii += 1

# setup of arr_cv matrix
num_gr = 0
gruppo0 = ""
ir = 0
while ir < n_rows:
  grsg_key = arr_grsg_c[ir].split()
  if not grsg_key[0] == gruppo0:
    gruppo0 = grsg_key[0]
    num_gr +=1
    ic = 1
    riga1 = []
    riga1.append(grsg_key[0])
    while ic <= m + 2:          # adding new columns for row mean and  n° of records
      riga1.append(0.0)
      ic += 1
    arr_cv.append(riga1)        # cv row
  ir += 1
riga1 = []
riga1.append("*Means*")   # adding new row for cv mean
ic = 1
while ic <= m + 2:                # adding new column for row mean and n° of records
  riga1.append(0.0)
  ic += 1
arr_cv.append(riga1)

def found(x):
```

```
  ir = 0
  while ir < len(arr_cv):
    linea_cv = arr_cv[ir]
    key_cv = linea_cv[0]
    if key_cv == x:
      return ir
    ir += 1

ir  = 0
irx = len(arr_grsg_c)
ic  = 3
linea_cv = arr_cv[0]
icx = len(linea_cv)
val_col = []

while ic < icx:
  ir = 0
  gruppo  = ""
  val_col = []
  while ir < irx:
    linea = arr_grsg_c[ir].split()
    if linea[0] == gruppo or gruppo == "":
      gruppo = linea[0]
      val_col.append(float(linea[ic]))
    else:
      i_gruppo = found(gruppo)
      linea_cv = arr_cv[i_gruppo]
      media_v = abs(mean(val_col))
      if media_v == 0.0:
         media_v = 0.0000000001
      std_v = sd(val_col)
      cv_v  = std_v / media_v
      linea_cv[ic-2] = cv_v                         # cv value
      linea_cv[len(linea_cv)-1] = len(val_col)    # number of records
      val_col = []
      val_col.append(float(linea[ic]))
      gruppo = linea[0]
    ir += 1
  i_gruppo = found(gruppo)
  linea_cv = arr_cv[i_gruppo]
  media_v = abs(mean(val_col))
  if media_v == 0.0:
    media_v = 0.0000000001
  std_v = sd(val_col)
  cv_v  = std_v / media_v
  linea_cv[ic-2] = cv_v                             # cv value
```

```
  linea_cv[len(linea_cv)-1] = len(val_col)       # number of records
  ic += 1
ir  = 0
irx = len(arr_cv)
linea_cv = arr_cv[0]
icx = len(linea_cv) - 2
ic  = 1
num_rec1 = 0

while ir < irx:                                  # rows mean
  media_riga = 0.0
  ic = 1
  num_col1 = 0
  linea_cv = arr_cv[ir]
  while ic < icx:
    media_riga += float(linea_cv[ic])
    num_col1 += 1
    ic += 1
  linea_cv[icx] = media_riga / num_col1
  num_rec1 += linea_cv[icx + 1]
  ir += 1
ir  = 0
ic  = 1

while ic < icx:                 # weighted mean of columns
  media_col = 0.0
  ir = 0
  num_rec1 = 0
  while ir < irx - 1:
    linea_cv = arr_cv[ir]
    media_col = media_col + linea_cv[ic] * linea_cv[icx+1]  # linea_cv[icx+1] = number
of records
    num_rec1 = num_rec1 + linea_cv[icx+1]
    ir += 1
  linea_cv = arr_cv[irx - 1]
  linea_cv[ic] = media_col / num_rec1
  ic += 1

# updating mean of the row
linea_cv = arr_cv[irx - 1]
linea_means = linea_cv[1:icx]
media_riga  = mean(linea_means)
linea_cv[icx] = media_riga        # Total mean
linea_cv[icx + 1] = num_rec1      # n° of records
cv_media_gen_after = str(media_riga)
cv_media_gen_after = cv_media_gen_after[0:6]
```

```
# write cv file

testata_cv = testata
testata_cv[0] = "*Groups*"
testata_cv.append("*Mean*")
testata_cv.append("N_recs")
arch_cv_file = open(arch_cv, 'w')
ic = 0
while ic <= icx + 1:
  arch_cv_file.write('%s %s ' % (testata_cv[ic], " "*(9-len(testata_cv[ic]))))
  ic += 1
arch_cv_file.write('%s' % ('\n'))
ir = 0
while ir < irx:
  ic = 0
  linea_cv = arr_cv[ir]
  while ic <= icx + 1:
    if ic == 0:
      arch_cv_file.write('%s %s ' % (linea_cv[0], "  "))
    else:
      if ic <= icx:
        arch_cv_file.write('%7.4f %s ' % (linea_cv[ic], "  "))
      else:
        arch_cv_file.write('%6i %s ' % (linea_cv[ic], "  "))
    ic += 1
  arch_cv_file.write('%s' % ("\n"))
  ir += 1
ic = 0

media_xcv = mean(xcv[1:icx])

while ic <= icx :   # print CV input (before catalogue)
  if ic == 0:
    arch_cv_file.write('%s %s ' % ("*CVinp*", "  "))
  else:
    if ic < icx:
      arch_cv_file.write('%7.4f %s ' % (xcv[ic], "  "))
    else:
      arch_cv_file.write('%7.4f %s ' % (media_xcv, "  "))
      arch_cv_file.write('%6i %s ' % (linea_cv[ic+1], "  "))
  ic += 1
arch_cv_file.write('%s' % ("\n"))

#=========istruzioni aggiunte Roberto Bello 29/02/2012======================
#know_index = str(1.0 - float(cv_media_gen_after) / float(str_med_cv_gen))
```

```
#know_index = know_index[0:6]
#arch_cv_file.write('%s %s %s' % ('*KIndex*   ', know_index, '\n'))
#=========fine istruzioni aggiunte da Roberto Bello 29/02/2012==============
arch_cv_file.close()

# writing out catalog file
testata_cat1 = []
testata_cat1.append("*Group*")
arch_output_file = open(arch_output, 'w')
ic= 0
while ic < icx:
  testata_cat1.append(testata_cat[ic])
  ic += 1
ic= 0
while ic < len(testata_cat1):
  arch_output_file.write('%s %s ' % (testata_cat1[ic], " "*(15-len(testata_cat1[ic]))))
  ic += 1
arch_output_file.write('%s ' % ("\n"))
index = 0
while index < len(arr_orig):
  riga_orig = arr_orig[index]
  ic = 0
  while ic < len(riga_orig):
    if not(isinstance(riga_orig[ic],str)):
      riga_orig[ic] = str(riga_orig[ic])
    ic += 1
  # place before 0 if gr / sg < 10
  gr1 = str(mtchx[index])
  if mtchx[index] < 10:
    gr1 = '0' + str(mtchx[index])
  sg1 = str(mtchy[index])
  if mtchy[index] < 10:
    sg1 = '0' + str(mtchy[index])
  arr_rig0 = "G_" + gr1 + "_" + sg1 + " "*8
  arch_output_file.write('%s ' % (arr_rig0))
  ic= 0
  while ic < len(riga_orig):
    arch_output_file.write('%s %s ' % (riga_orig[ic], " "*(15-len(riga_orig[ic]))))
    ic += 1
  arch_output_file.write('%s ' % ("\n"))
  index += 1
testata_cat1 = []
testata_cat1.append("*Group*")
testata_cat1.append("*RecNum*")
arch_sort_file = open(arch_sort, 'w')
ic= 0
```

```
while ic < icx:
  testata_cat1.append(testata_cat[ic])
  ic += 1
ic= 0
while ic < len(testata_cat1):
  arch_sort_file.write('%s %s ' % (testata_cat1[ic], " "*(15-len(testata_cat1[ic]))))
  ic += 1
arch_sort_file.write('%s ' % ("\n"))
index = 0
while index < len(arr_grsg_c):
  riga_grsg = arr_grsg_c[index].split()
  ic = 0
  while ic < len(riga_grsg):
    val_txt = riga_grsg[ic]
    val_txt = val_txt[0:13]
    arch_sort_file.write('%s %s ' % (val_txt, " "*(15-len(val_txt))))
    ic += 1
  if index < len(arr_grsg_c) - 1:
    arch_sort_file.write('%s ' % ("\n"))
  index += 1
arch_sort_file.close()

# writing out catalog and sorted file
arr_outsrt = []
index = 0
while index < len(arr_orig):
  riga_sort = []
  # place before 0 if gr / sg < 10
  gr1 = str(mtchx[index])
  if mtchx[index] < 10:
    gr1 = '0' + str(mtchx[index])
  sg1 = str(mtchy[index])
  if mtchy[index] < 10:
    sg1 = '0' + str(mtchy[index])
  riga_sort.append("G_" + gr1 + "_" + sg1)
  ic = 0
  riga_orig = arr_orig[index]
  while ic < len(riga_orig):
    val_riga = riga_orig[ic]
    riga_sort.append(val_riga)
    ic += 1
  arr_outsrt.append(riga_sort)
  index += 1

for line in arr_outsrt:
  line = "".join(line)
```

```
arr_outsrt.sort()

testata_srt = []
testata_srt.append("*Group*")
arch_outsrt_file = open(arch_outsrt, 'w')
ic= 0
while ic < icx:
  testata_srt.append(testata_orig[ic])
  ic += 1
ic= 0
while ic < len(testata_srt):
  arch_outsrt_file.write('%s %s' % (testata_srt[ic], " "*(15-len(testata_srt[ic]))))
  ic += 1
arch_outsrt_file.write('%s' % ("\n"))
index = 0
key_gruppo = ""
while index < len(arr_outsrt):
  riga_sort = arr_outsrt[index]
  index_c = 0
  while index_c < len(riga_sort):
    if index_c == 0:
      if riga_sort[0] != key_gruppo:
        # arch_outsrt_file.write('%s ' % ("\n"))
        key_gruppo = riga_sort[0]
    valore = riga_sort[index_c]
    arch_outsrt_file.write('%s %s' % (valore, " "*(15-len(valore))))
    index_c += 1
  if index < len(arr_grsg_c) - 1:
    arch_outsrt_file.write('%s' % ("\n"))
  index += 1
arch_outsrt_file.close()

print("###############################################################################")
print("# KB_CLA KNOWLEDGE DISCOVERY IN DATA MINING (CLASSIFY PROGRAM)                #")
print("# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                 #")
print("# Language used: PYTHON                                                       #")
print("###############################################################################")

arch_log_file = open(arch_log, 'w')
arch_log_file.write("%s %s" %
("############################################################################", "\n"))
arch_log_file.write("%s %s" % ("# KB_CLA KNOWLEDGE DISCOVERY IN DATA MINING (CLASSIFY
PROGRAM)             #", "\n"))
arch_log_file.write("%s %s" % ("# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS
RESERVED)                #", "\n"))
arch_log_file.write("%s %s" % ("# Language used: PYTHON                          .
```

```
#", "\n"))
arch_log_file.write("%s %s" %
("##############################################################################", "\n"))
arch_log_file.write("%s %s %s" % ("Input File                                                ->
", file_input, "\n"))
arch_log_file.write("%s %s %s" % ("Numer of Groups (3 - 20)                                  ->
", str(gruppi_num), "\n"))
arch_log_file.write("%s %s %s" % ("Normalization (Max, Std, None)                            ->
", tipo_norm, "\n"))
arch_log_file.write("%s %s %s" % ("File Training Grid                                        ->
", arch_grid, "\n"))
arch_log_file.write("%s"       %
("=====================================OUTPUT==============================================================================
\n"))
arch_log_file.write("%s %s %s" % ("Output File Classify.original    ", arch_output,
"\n"))
arch_log_file.write("%s %s %s" % ("Output File Classify.sort        ", arch_outsrt,
"\n"))
arch_log_file.write("%s %s %s" % ("Output File Summary sort         ", arch_sort, "\n"))
arch_log_file.write("%s %s %s" % ("Output File Matrix Catal.        ", arch_catal,
"\n"))
arch_log_file.write("%s %s %s" % ("Output File Means, STD, CV.      ", arch_medsd,
"\n"))
arch_log_file.write("%s %s %s" % ("Output File CV of the Groups     ", arch_cv, "\n"))
arch_log_file.write("%s %s %s" % ("Output File Training Grid        ", arch_grid, "\n"))
arch_log_file.write("%s %s %s" % ("Output File Run Parameters       ", arch_log, "\n"))
#=========istruzioni aggiunte Roberto Bello 29/02/2012======================
know_index = str(1.0 - float(cv_media_gen_after) / float(str_med_cv_gen))
know_index = know_index[0:6]
arch_log_file.write('%s %s %s' % ('*KIndex*   ', know_index, '\n'))
#=========fine istruzioni aggiunte da Roberto Bello 29/02/2012==============

print
print 'Output File Classify.original ' + arch_output
print 'Output File Classify.sort     ' + arch_outsrt
print 'Output File Summary sort      ' + arch_sort
print 'Output File Matrix Catal.     ' + arch_catal
print 'Output File Means, STD, CV.   ' + arch_medsd
print 'Output File CV of the Groups  ' + arch_cv
print 'Output File Training Grid     ' + arch_grid
print 'Output File Run Parameters    ' + arch_log
print 'CV after Catalog             ' + cv_media_gen_after
know_index = str(1.0 - float(cv_media_gen_after) / float(str_med_cv_gen))
know_index = know_index[0:6]
print 'Knowledge Index              ' + know_index
print

# Elapsed time
t1 = datetime.datetime.now()
elapsed_time = t1 - t0
```

```
print "Elapsed time (seconds)   :   " + str(elapsed_time.seconds) + "." +
str(elapsed_time.microseconds)
print
```

Appendix 4 – KB_RND source

```
# -*- coding: utf-8 -*-
import os
import random
import copy
import datetime

print("###############################################################################")
print("# KB_RND KNOWLEDGE DISCOVERY IN DATA MINING (RANDOM FILE SIZE REDUCE)         #")
print("# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)                 #")
print("# Language used: PYTHON                                                       #")
print("###############################################################################")

# input and run parameters
error = 0

while True:
  arch_input = raw_input('InputFile                             : ')
  if not os.path.isfile(arch_input):
    print("Oops! File does not exist. Try again... or CTR/C to exit")
  else:
    break

while True:
  arch_output = raw_input('OutputFile                            : ')
  if os.path.isfile(arch_output):
    print("Oops! File does exist. Try again... or CTR/C to exit")
  else:
    break

while True:
  try:
    num_cells_out = int(raw_input('Out number of cells (<= 90000)         : '))
  except ValueError:
    print("Oops!  That was no valid number.  Try again...")
  else:
    if(num_cells_out > 90000):
      print("Oops! Number of Cells too big. Try again...")
    else:
      break

# start time
```

```
t0 = datetime.datetime.now()

# read input file
arr_r    = []
arr_rnd  = []
arr_out  = []
xnomi    = []
index    = 0

for line in open(arch_input).readlines():
  linea = line.split()
  if(index == 0):
    xnomi.append(linea)
  else:
    arr_r.append(linea)
  index += 1
rec_input = index - 1
num_cols = len(linea)
num_records_out = int(num_cells_out / num_cols)
print "Nun. Records Input " + str(rec_input)

if rec_input < num_records_out:
  num_records_out = rec_input

# random values sequence

ix = 950041                         # integer as random seed
random.seed(ix)                            # initial value of random seed to obtain the
same sequences in new runs
index = 0
while index < num_records_out:
  val_rnd = int(random.random()*rec_input)
  doppio = 0
  for index_rnd, item in enumerate(arr_rnd): # check for duplicates
    if item == val_rnd:
      doppio = 1
  if doppio == 0:
    arr_rnd.append(val_rnd)
    index += 1

# arr_out writing

index = 0

arr_out.append(xnomi[0])            # header
while index < len(arr_rnd):
```

```
  key1 = arr_rnd[index]
  arr_out.append(arr_r[key1])              # from source to random output
  index += 1

# write arch_out_file
arch_out_file = open(arch_output, 'w')
index = 0
while index < len(arr_out):
  line = arr_out[index]
  ncol = 0
  while ncol < len(line):
    field = line[ncol].strip()
    if ncol < len(line) - 1:
      arch_out_file.write('%s%s' % (field, "\t"))
    else:
      arch_out_file.write('%s%s' % (field, "\n"))
    ncol += 1
  index += 1
arch_out_file.close()

# Elapsed time
t1 = datetime.datetime.now()
elapsed_time = t1 - t0
print "Elapsed time (microseconds)            :   " + str(elapsed_time.microseconds)
print
```

KB – Guarantee and copyright

The author guarantees that the work is without viruses and malicious codes also considering:

- the text is in pdf format
- the python programs are in *txt* format and do not contain malicious code, as easily verifiable by a simple reading of their
- the test files are in a *txt* format
- the language for the processing of the programs (python) is of Open Source type and is universally recognised as reliable and safe.

As regards the copyright, the author does not intend to renounce his legal rights on the algorithms and on the method of computing and analysis contained in the KB programs.

Roberto Bello

Graduate in Economics and Commerce with specialization in Operations Research
Data Scientist, expert in Knowledge Mining, in Data Mining and in Open Source
ICT Strategist of the ClubTI of Milan (www.clubtimilano.net)
Researcher of the AISF (www.accademiascienzeforensi.it)
Expert (CTP) and ex CTU (Technical Office Consultant) of the Court of Milan

Author of professional publications available on www.lulu.com/spotlight/robertobb
Founding associate of AIPI (Italian Professional Computer Association)
In the past CIO of Plasmon, of Wrangler in Italy and consultant for the most important Italian food companies
Linkedin: it.linkedin.com/pub/roberto-bello/4/1a5/677

Table of contents