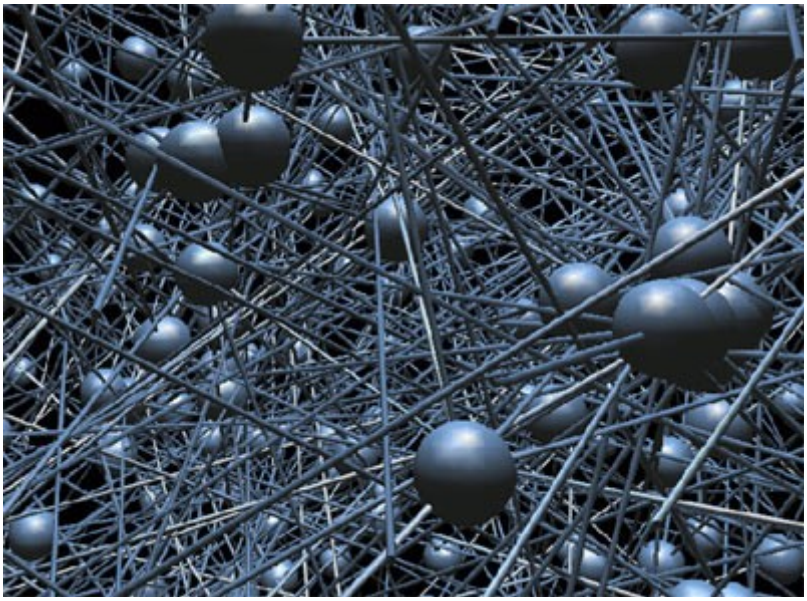
KB – Neural Data Mining con sorgenti Python

Presentazione

L'opera ha per obiettivo di presentare e di descrivere nei dettagli gli algoritmi di calcolo per l'estrazione della conoscenza nascosta nei dati utilizzando il linguaggio Python, linguaggio che consente di leggere e capire facilmente la natura e le caratteristiche delle regole di calcolo utilizzate, a differenza di quanto avviene per gli applicativi commerciali forniti solo nella forma di codice eseguibile che restano misteriosi ed inaccessibili a qualsiasi modifica.
Gli algoritmi di calcolo contenuti nell'opera, minutamente descritti, documentati e disponibili in formato sorgente Python, hanno lo scopo di estrarre la conoscenza nascosta nei dati siano essi di tipo testuale o di tipo numerico. Sono presentati anche diversi esempi di utilizzo, evidenziandone le caratteristiche, le modalità di esecuzione e fornendo i commenti ai risultati ottenuti.



L'applicativo KB è costituito da tre programmi di calcolo:

- KB_CAT: per l'estrazione della conoscenza dai dati e la catalogazione dei record in gruppi omogenei al loro interno
- KB_STA: per l'analisi statistica delle omogeneità dei gruppi fra di loro e nei gruppi al loro interno allo scopo di individuare i gruppi più significativi e le variabili più importanti che caratterizzano ogni gruppo
- KB_CLA: per la classificazione quasi istantanea di nuovi record nei gruppi di catalogazione prima determinati.

I programmi sono stati scritti nel linguaggio Python utilizzando preferibilmente i comandi, le istruzioni e le funzioni maggiormente comprensibili e simili a quelle di altri linguaggi (es. C, C++, PHP, Java, Ruby); comunque i programmi abbondano di

commenti e di spiegazioni.
L'applicativo KB per acquisire la conoscenza nascosta nei dati è il risultato di quasi cinque anni di studio, di programmazione e di prove anche con altri linguaggi (Clipper, Fortran, Ruby, C e C++).
Il costo dell'opera è modesto considerando l'importanza degli algoritmi di calcolo pubblicati e l'impegno profuso dall'autore nella loro realizzazione e nei successivi ripetuti ed accurati collaudi sia su dati di test sia su file reali contenenti molte migliaia di record; i file usati nelle applicazioni reali appartenevano a diversi settori di interesse (medicina, farmacologia, industria alimentare, sondaggi politici, prestazioni sportive e ricerche di mercato).
L'autore volutamente ha utilizzato forme molto semplici di comunicazione, evitando il ricorso a formule matematiche complicate, dopo aver constatato che i concetti utili potevano essere divulgati in modo semplice, ma non semplicistico.
Albert Einstein diceva: *Fate le cose nel modo più semplice possibile, ma senza semplificare*.
L'autore spera di dare un piccolo contributo teso a rinnovare nei giovani l'amore per la matematica, ma soprattutto spera nel ritorno del desiderio di realizzare programmi con il proprio computer, evitando di usarlo solo per consumarsi le dita navigando in *facebook* o scaricando musica e film da Internet.
Nella sua esperienza professionale di informatico prima e di dirigente industriale poi, ha ripetutamente realizzato programmi con contenuti matematici, statistici e di ricerca operativa che hanno contribuito notevolmente ai risultati economici e di buona gestione delle imprese che lo hanno visto protagonista significativo in importanti progetti.

La Business Intelligence e il cattivo uso della statistica

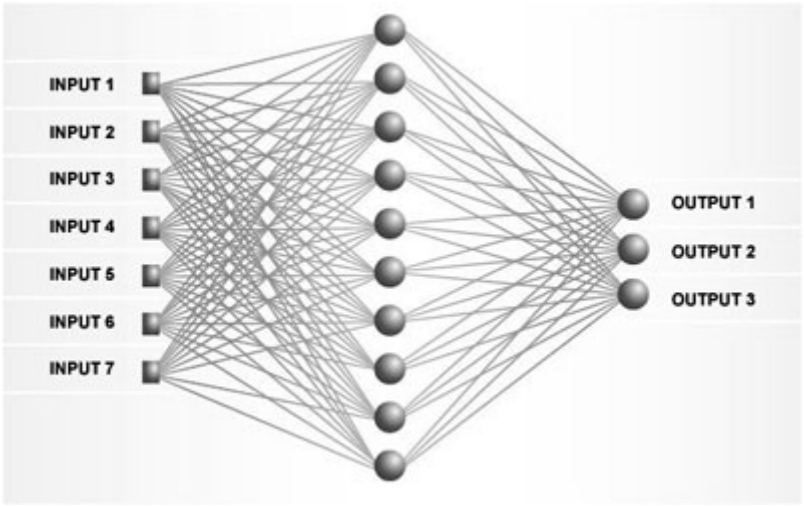
La statistica molto spesso sbaglia o, per meglio dire, sbagliano i suoi utilizzatori.
Sbagliano quando applicano gli strumenti statistici di aggregazione a frammenti informativi provenienti da oggetti o situazioni fra di loro del tutto differenti.
Prima frammentano, poi mescolano ed infine aggregano.
Per finire pretendono di sentenziare.
Così i ricercatori di tendenze politiche spezzettano le opinioni degli intervistati, mescolano le singole risposte, aggregano, incrociano ed infine sentenziano certezze che possono essere attribuite solo agli *intervistati virtuali* che loro hanno creato, soggetti non esistenti nella realtà e sicuramente non riconducibili ai singoli individui o a gruppi omogenei di intervistati.
In modo analogo la *Business Intelligence* rende disponibili degli strumenti di analisi dei dati in grado di tagliare i dati e poi di ricomporli in strutture multidimensionali nelle quali le peculiarità informative delle situazioni di partenza sono state distrutte.
Così con la *Business Intelligence* si mescolano aziende di diversi settori con fatturati non compatibili, con organici non comparabili, appartenenti a mercati dissimili dove si usano sistemi di pagamento diversi, abusando dell'arbitrio di cambiare di volta in volta le variabili di incrocio dei dati.
A quali soggetti (o situazioni) potrebbero essere applicate le decisioni che poi si prendono, avendo distrutto il patrimonio informativo globale dei soggetti (o situazioni) di partenza?

Per fare un esempio, se avessi un file di animali mammiferi nel quale fossero compresi anche uomini e primati, potrei ottenere come risultato che i mammiferi hanno mediamente circa tre zampe.
Dove trovo un mammifero che abbia mediamente tre zampe?
Per fare della vera statistica occorre conservare il più possibile intatto il patrimonio informativo dei dati di partenza dei soggetti o delle situazioni sotto esame.
Le tecniche derivate dalle reti neurali usano un approccio all'analisi dei dati del tutto rispettoso del patrimonio informativo dei dati di partenza.
Infatti non richiedono all'utente di definire le variabili da incrociare, impedendogli così di formulare incroci assurdi.
Richiedono unicamente di inserire il numero massimo dei gruppi che l'algoritmo dovrà creare.
Non distruggono il patrimonio informativo dei dati di partenza, ma elaborano sempre i dati dei soggetti (o situazioni) in rapporto ai dati degli altri soggetti (o situazioni).
Conservano tutte le informazioni attribuibili al soggetto o alla situazione in esame e creano le categorie di appartenenza dei soggetti (o situazioni) nelle quali i soggetti (o situazioni) saranno fra di loro simili.
Altre tecniche sono in grado di segnalare quali siano le variabili significative di aggregazione e quali siano i valori di aggregazione importanti per ogni gruppo creato.
Segnalano anche quali siano le variabili non influenti nella catalogazione.
Tecniche più sofisticate possono elaborare qualsiasi tipo di insieme di dati evidenziando se nel file sono presenti delle informazioni oppure sono presenti solo numeri o caratteri fra di loro non legati da relazioni interne.
Il modello deve seguire i dati e non viceversa (J. B. Benzecri).

L'apprendimento per induzione e le reti neurali

L'induzione è una modalità molto importante di apprendimento delle creature viventi.
Uno dei primi filosofi ad essere ricorso a questo concetto fu Aristotele il quale, attribuendo a Socrate il merito di averla scoperta, sosteneva che l'induzione fosse, appunto, "*il procedimento che dai particolari porta all'universale" (Top., I, 12, 105 a 11)*. Sempre secondo Aristotele non sono né i sensi per via induttiva, né la razionalità per via deduttiva, a dare di per sé garanzia di verità, bensì soltanto l'intuizione intellettuale: essa consente di cogliere l'essenza della realtà fornendo dei principi validi e universali, da cui il ragionamento sillogistico trarrà delle conclusioni coerenti con le premesse.
L'apprendimento, la vita e l'evoluzione sono fra di loro correlati.
Infatti la vita e' evoluzione ed evoluzione e' apprendimento di ciò che e' necessario alla sopravvivenza. Apprendimento e' la capacita' di elaborare le informazioni con intelligenza critica. Quindi l'elaborazione critica delle informazioni e' vita. (Roberto Bello).
Un semplice esempio può illustrare come si apprende per induzione.
Ipotizziamo di aver di fronte una persona che non abbia mai visto dei contenitori di uso comune come bicchieri, bottiglie, barattoli, tazze, vasi, scatole, fiaschi, boccali,

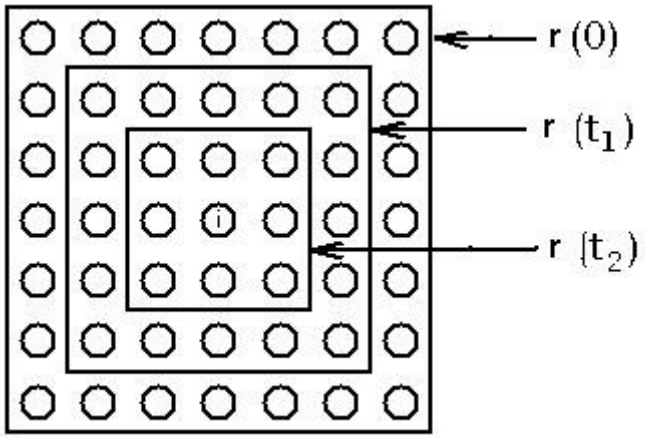
calici, *tetrapack* e via dicendo.
Senza alcun commento mostro in successione esempi reali di oggetti appartenenti alle categorie sopra descritte.
La persona può guardare, odorare, toccare e soppesare gli oggetti mostrati.
Dopo aver esaminato un sufficiente numero di oggetti, la persona facilmente sarà in grado di raggruppare gli oggetti in categorie contenenti gli oggetti fra di loro *globalmente* simili, privilegiando alcune caratteristiche rispetto ad altre ritenute ininfluenti perché non discriminanti.
Ad apprendimento avvenuto, io potrei presentare un altro oggetto a forma di bicchiere di altro colore, di altro materiale e di altro peso ottenendo comunque la collocazione dell'oggetto nella categoria dei bicchieri.
Sempre per induzione la persona in addestramento potrebbe fare due categorie dei bicchieri: quelli senza manico e quelli con manico (boccali).
L'apprendimento ha consentito alla persona di riconoscere gli aspetti dell'oggetto utili per passare dal particolare all'universale trascurandone gli aspetti non influenti.
Gli algoritmi basati sulle reti neurali, con particolare riferimento alle mappe di Kohonen (*SOM Self Organizing Map*), si basano sui principi appena illustrati nell'esempio.
Tale modello di Rete Neurale riflette in modo significativo i meccanismi biologici del sistema nervoso centrale; molti studi hanno infatti dimostrato che sulla superficie della corteccia celebrale esistono delle zone ben definite, ciascuna delle quali risponde ad una precisa funzione sensoria o motoria.
Ogni neurone si specializza a rispondere a determinati stimoli attraverso un’interazione continua con i neuroni confinanti.
Avremo quindi zone riservate all’udito, alla visione, all’attività motoria etc., e la demarcazione spaziale tra i diversi gruppi è tanto netta che si parla di formazione di bolle di attività.
Il modello di Rete Neurale presentato da Kohonen imita il comportamento sopra descritto.
L’architettura è abbastanza semplice; la rete è formata da una griglia rettangolare, detta anche *strato di Kohonen*, composta dai neuroni del livello di output, ciascuno dei quali occupa una precisa posizione ed è collegato a tutte le unità di ingresso.
I pesi delle connessioni tra il livello di input e quello di output sono aggiornati grazie al processo di apprendimento, mentre le connessioni tra i neuroni del livello di output presentano pesi che producono eccitazione tra i neuroni limitrofi ed inibizione tra i neuroni lontani.



Schema di una rete neurale

Le *reti SOM* sono applicate a molti problemi pratici; hanno la capacità di scoprire in modo autonomo proprietà importanti tra i dati di input e quindi sono particolarmente utili nei processi di *Data Mining*, soprattutto per problemi di catalogazione.
L'algoritmo di apprendimento delle reti di Kohonen parte dalla fase di inizializzazione dei pesi sinaptici, i quali devono assumere valori casuali nell'intervallo (0.0 – 0.99999) ed essere differenti per ogni neurone.
Successivamente sono presentati alla rete dei valori di input e l'algoritmo adottato permetterà alla rete di auto-organizzarsi e correggere i pesi dopo ogni presentazione degli input, fino a raggiungere uno stato di equilibrio.
Le reti di Kohonen sono anche definite *reti competitive* poiché si basano sul principio di competizione tra i neuroni per vincere e rimanere attivi; solo i pesi delle unità attive sono modificati. L'unità vincente *i** è quella che possiede il potenziale di attivazione maggiore; quanto più un'unità è attiva per un certo *pattern* di ingresso, tanto più il vettore dei pesi sinaptici è simile a tale *pattern*.
In base a questo assunto è possibile trovare l'unità vincente calcolando la distanza euclidea tra il vettore di input ed il relativo vettore di pesi sinaptici. A questo punto si seleziona il neurone *i** cui corrisponde la distanza minima.
Una volta determinato il neurone vincitore viene effettuato un aggiornamento automatico dei pesi del neurone stesso e di quelli che fanno parte del suo vicinato, in base ad una regola di tipo *hebbiano*.
In particolare si prende in considerazione una formula di modifica dei pesi che deriva dall'originale *regola di Hebb*; dato che quest'ultima farebbe crescere i pesi all'infinito viene introdotto un fattore di dimenticanza, spingendo i pesi verso i vettori di ingresso ai quali l'unità risponde maggiormente.
Si crea in questo modo una mappa relativa alle caratteristiche degli input dove unità limitrofe rispondono a determinati stimoli di ingresso grazie alla similarità dei pesi sinaptici.
A questo proposito è necessario introdurre anche il concetto di funzione di

vicinanza, che determina l'area di ampiezza *r* intorno ad *i** in cui le unità sono attive.
Minore è la dimensione del vicinato, minore è il numero di unità dello *strato di Kohonen* i cui pesi sono modificati significativamente, quindi maggiore è la capacità dei neuroni di differenziarsi e di acquisire dettagli ma anche la complessità del sistema di apprendimento.
Secondo Kohonen l'ampiezza della funzione di vicinanza deve essere fatta variare, scegliendola inizialmente in modo da coprire tutte le unità dello strato e facendola decrescere gradualmente.
In questo modo si passerà dall'apprendimento delle caratteristiche di ingresso di base all'apprendimento di dettagli e di specializzazione delle aree nel rispondere a determinati stimoli.



Rappresentazione della diminuzione graduale del vicinato

Conclusa la fase di addestramento la rete è in grado di fornire risposte in corrispondenza dei nuovi input presentati. La proprietà di generalizzazione deriva dal fatto che anche i neuroni vicini a quello selezionato sono modificati.
La rete dovrà quindi auto-organizzarsi in aree costituite da un ampio insieme di valori attorno all'input dal quale apprende; questo farà sì che se è presentato un input mai visto prima ma con caratteristiche simili, la rete sarà in grado di classificarlo nel modo appropriato.
Inoltre, rispetto agli algoritmi di tipo supervisionato, il processo di apprendimento *auto-organizzato* risulta efficiente anche se vengono utilizzati dati di input incompleti o contenenti errori, caratteristica che rende queste reti particolarmente adatte ad essere applicate nel processo di *Data Mining*.
Infatti l'algoritmo di Kohonen, al termine della fase di *addestramento non supervisionato*, produce una matrice tridimensionale che può essere utilizzata per la classificazione di nuovi record nei gruppi con caratteristiche di maggiore somiglianza.
Mentre la fase di addestramento può richiedere molto tempo di elaborazione, quella di classificazione di nuovi record nei gruppi con maggiore somiglianza è quasi istantanea, rendendo questa funzione utilmente applicabile a processi decisori e di

reazione in tempo reale (es. controllo di qualità in produzione, automazione di processi industriali, sistemi di controllo, monitoraggio del traffico dei messaggi in rete, etc.).
Gli algoritmi delle reti neurali hanno, come aspetto comune, l'incapacità di spiegare le caratteristiche dei gruppi ottenuti.
E' però possibile, utilizzando le informazioni contenute nella matrice di addestramento e ricorrendo ad altre tecniche statistiche, fornire le informazioni sulle caratteristiche di ogni gruppo aiutando il ricercatore nell'approfondire l'analisi dei risultati per meglio documentare le conclusioni della sua ricerca.
E' anche possibile determinare se la globalità di record utilizzati in fase di addestramento abbia dei contenuti di conoscenza o, al contrario, sia composta da dati fra di loro poco correlati e quindi non idonei ai fini della ricerca: infatti è possibile calcolare un indice globale di omogeneità dei gruppi nel loro insieme (Knowledge Index), informando il ricercatore sull'idoneità del file dei dati sottoposti ad elaborazione a conseguire gli obiettivi attesi.

Installazione di Python per utilizzare KB

Il linguaggio di programmazione Python è un linguaggio che può essere gratuitamente scaricato da Internet.
Python è compatibile con Windows, Linux/Unix, Mac OS X, OS/2, Amiga, Android, etc.
Python è distribuito con licenza Open-Source: il suo utilizzo è gratuito e libero anche per prodotti commerciali.
Il sito dal quale scaricare il linguaggio Python è *www.python.org/download*, scegliendo la versione adatta al proprio computer.
L'installazione di Python in ambiente Windows avviene mandando in esecuzione il file con estensione *msi* scaricato da Internet.
L'installazione di Python in ambiente Linux (e in Linux Ubuntu in particolare) avviene utilizzando il gestore dei pacchetti della distribuzione Linux, gestore che automaticamente si collega al sito contenitore (*repository*) ufficiale, *scaricando* quanto necessario per una successiva sicura, completa ed automatica installazione; le distribuzioni Linux normalmente contengono già preinstallato il linguaggio Python.
Qualunque sia il sistema operativo di installazione di Python, i programmi possono essere eseguiti solo in modalità *da riga di comando* aprendo:

- in Windows una finestra DOS (con *esegui*) nella cartella contenente il programma sorgente Python (esempio: programma.py), digitando poi *python programma.py*
- in Linux una finestra di terminale (*terminal*) nella cartella contenente il programma sorgente Python (esempio: programma.py), digitando poi python *programma.py*

KB - Generalità

L'applicativo KB è un sistema di estrazione della conoscenza contenuta nei dati basato su un algoritmo delle mappe di Kohonen rivisto e modificato dall'autore.
KB può elaborare qualsiasi tabella di dati numerici e/o testuali, tabella nella quale la prima riga della tabella è destinata alla descrizione delle colonne / variabili e la prima colonna della tabella è destinata ai codici (arbitrari) di identificazione dei record / casi.
In KB sono comprese delle funzionalità con l'obiettivo di:

- normalizzare i dati di tipo numerico rapportandoli alla *standard deviation* o al valore massimo della variabile / colonna a scelta dell'utilizzatore
- trasformare i dati alfanumerici in dati numerici opportunamente resi fra di loro equidistanti
- inserire funzioni statistiche in grado di valutare la bontà dei risultati delle catalogazioni per ogni singolo gruppo e globalmente
- scrivere diversi file di output contenenti:
  - i record / casi ordinati per codice di gruppo di catalogazione
  - le informazioni statistiche sintetiche sulle caratteristiche dei diversi gruppi in rapporto anche agli indici statistici riferibili all'intera popolazione dei record / casi
  - la matrice di addestramento finale avente l'errore minimo

Le reti neurali hanno il riconosciuto difetto di essere delle *scatole nere* nel senso che sono in grado di catalogare ma non *spiegano*:

- quali siano le caratteristiche di ogni gruppo
- quali siano le colonne / variabili che, per ogni gruppo, sono importanti nella catalogazione
- quali siano i gruppi maggiormente omogenei al loro interno
- se, nella sua globalità, la tabella di input contiene informazioni nel senso di relazioni fra le variabili oppure se la tabella è soltanto un insieme di numeri e di caratteri senza alcuna valenza di tipo induttivo.

Le pagine che seguono contengono i programmi sorgenti scritti in Python di KB_CAT (per la catalogazione), di KB_STA (per l'analisi dei risultati) e di KB_CLA (per la classificazione).
Essi possono essere convertiti in file in formato testo ricorrendo al *copia e incolla*; i programmi devono essere memorizzati rispettivamente con i nomi:

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

I nomi dei programmi possono anche essere diversi da kb_cat, kb_sta e kb_cla, purché l'estensione sia "*.py*" per consentire al linguaggio Python di riconoscere i programmi e di mandali in esecuzione.
Sono anche riprodotti alcuni dei file di prova e i risultati ottenuti rappresentati nella finestra DOS (Windows) o nella finestra del Terminale (Linux), risultati contenuti in file in formato testo.

Raccolta e predisposizione dei dati di input

L'utilizzo di programmi basati sugli algoritmi di Kohonen richiede che i dati siano preparati ed opportunamente normalizzati.
In primo luogo è importante scegliere accuratamente i dati che dovranno essere analizzati.
Le informazioni dalle quali l'utente intende estrarre conoscenza devono essere contenute in un file a forma tabellare avente le seguenti caratteristiche:

- il formato deve essere di tipo testo (txt, csv)
- i campi devono essere separati da tabulazioni (tab)
- la prima riga deve contenere le descrizioni delle colonne separate da tabulazioni (*tab*)
- la prima colonna è destinata ad identificare per ogni riga il codice identificativo di ogni record (es. codice cliente, nome prodotto, lotto di produzione, etc.)
- le informazioni da elaborare sono contenute nelle celle all'incrocio fra record / caso e variabile / colonna dalla seconda colonna all'ultima colonna e dalla seconda riga all'ultima riga
- tutti i valori di tutte le colonne e per tutte le righe devono essere separati da tabulazioni (*tab*)
- non possono esistere campi vuoti o contenenti solo spazi
- una colonna destinata a contenere dati numerici non può contenere dati di tipo testo

Per convertire in formato testo delle tabelle disponibili in formato *xls* (*Excel*) o *Libre-OpenOfficeCalc* (*ods*) basta ricorrere ai programmi in grado di leggere tali formati e convertirli in formato (*csv*), scegliendo come delimitatore di campo la tabulazione (*tab*) e *nulla* (*vuoto*) come delimitatore di testo.
Per la qualità dei risultati, vale sempre la famosa legge *spazzatura in entrata, spazzatura in uscita (garbage in, garbage out)*; è fondamentale raccogliere dati di buona qualità che consentano di descrivere e spiegare il fenomeno nel modo più completo possibile.
Occorre anche decidere quale parte dei dati utilizzare come file di input (numerosità del campione).
Le reti neurali danno il medesimo peso a tutte le variabili inserite; se una variabile oscilla in un intervallo (1000 - 10000) e l'altra in un intervallo (0 - 1), le variazioni della prima tendono a ridurre l'importanza della seconda, anche se quest'ultima potrebbe essere più significativa per la determinazione dei risultati della catalogazione.
Per questo esistono tecniche di trasformazione che rendono le variabili compatibili fra di loro, facendole ricadere all'interno di un certo intervallo (*range*).
Il programma KB_CAT può applicare diverse tecniche di normalizzazione dei dati numerici e dei dati di tipo testo.
I valori numerici possono essere normalizzati attraverso due metodi:

- Normalizzazione con il massimo: i nuovi valori della colonna sono ottenuti dividendo i valori originali per il valore massimo della colonna, in tal modo i nuovi valori varieranno fra zero e 1

- Standardizzazione: i nuovi valori sono ottenuti sottraendo al valore originale del dato la media della colonna e dividendo il risultato della differenza per la *standard deviation* della colonna.

Dalle colonne contenenti stringhe di caratteri si estraggono e si contano i valori delle stringhe diverse tra loro il loro numero è usato per determinare il passo di attribuzione del valore numerico compreso tra 0 e 1.
Il programma KB_CAT non prevede la trasformazione automatica di date e di orari.
Le date dovrebbero essere trasformate a cura dell'utilizzatore in variabili numeriche *pseudo continue* assegnando il valore 0 alla data più remota ed incrementando di una unità ogni data successiva, oppure esprimendo i 365 giorni dell'anno in millesimi, secondo la formula: *0,9999*giorno dell'anno/365*.
L'anno potrebbe invece essere indicato attraverso l'utilizzo di un'altra variabile.
Le coppie di variabili tra loro in relazione sono da esprimere preferibilmente come rapporto, attraverso un unico valore che offre informazioni più chiare ed immediate; in questo modo si calcolano delle variabili derivate a partire dalle variabili di input.
Ipotizziamo siano presenti due variabili: il peso e l'altezza della persona.
Considerate separatamente hanno poco significato, sarebbe meglio ottenere da esse il coefficiente di massa corporea che è sicuramente un buon indice sintetico di obesità (peso corporeo espresso in chilogrammi diviso l'altezza in metri al quadrato).
Un altro passo importante nell'elaborazione preliminare dei dati è quello di cercare di semplificare il problema che si vuole risolvere. A tal fine può essere utile ridimensionare lo spazio dei valori di input, composto da tutti i possibili legami tra i dati di input, spazio che cresce esponenzialmente al crescere dei dati.
Una tecnica spesso utilizzata per ridurre il numero delle variabili e migliorare la capacità di apprendimento delle reti neurali, è l'*analisi delle componenti principali*, che cerca di individuare un sotto-spazio a *m* dimensioni che sia il più significativo possibile rispetto allo spazio di input a *n* dimensioni.
Le *m* variabili finali sono dette *componenti principali* e sono combinazioni lineari delle *n* variabili di partenza.
Altri metodi impiegati per ridurre la dimensionalità del problema da risolvere sono l'eliminazione di variabili tra loro fortemente correlate e di dati poco importanti per l'obiettivo che si intende raggiungere.
Nel primo caso è utile considerare che non sempre la *correlazione* implica una relazione di causa / effetto, quindi l'eliminazione dei dati deve essere effettuata con particolare attenzione da parte dell'utente.
E' molto frequente riorganizzare il file dei dati di input da catalogare esaminando i risultati delle prime elaborazioni che spesso segnalano che certe variabili / colonne sono del tutto ininfluenti: la loro eliminazione nelle elaborazioni successive contribuisce spesso a migliorare la catalogazione essendosi attenuato il *rumore* causato dalle variabili / colonne inutili.
Nelle elaborazioni di dati riferiti a ricerche cliniche, è stato verificato che i dati anagrafici di sesso, nazionalità, residenza, istruzione, etc., non dando in quei casi alcun contributo alla catalogazione, potevano essere omessi migliorando la qualità del nuovo apprendimento.
Un aspetto molto importante da considerare è relativo alla numerosità dei record

contenuti nel file di input alla catalogazione.
Molto spesso si ottengono migliori risultati con file di modeste dimensioni che sono in grado di meglio generalizzare e di produrre *matrici di addestramento* maggiormente predittive.
Al contrario un file contenente un numero considerevole di record potrebbe produrre un addestramento viziato da *overfitting* causando un *effetto fotografia* in grado solo di classificare nuovi record quasi identici a quelli usati nella fase di catalogazione.
As scientists at Cern have already discovered, it's more important to properly analyze the fraction of the data that is important ("of interest") than to process all the data. TomHCAnderson
In statistica si parla di *overfitting* (eccessivo adattamento) quando un modello statistico si adatta ai dati osservati (il campione) usando un numero eccessivo di parametri.
Un modello assurdo e sbagliato può adattarsi perfettamente se è abbastanza complesso per adeguarsi alla quantità dei dati disponibili.
Non è possibile segnalare a priori la numerosità ottimale dei record contenuti nei file da catalogare: troppo dipende dal numero delle variabili e dal contenuto informativo delle variabili per tutti i record presenti nel file.
Il migliore suggerimento è quello di effettuare distinte elaborazioni con il file originale e con altri file ottenuti con l'estrazione casuale di un numero inferiori di record.
Per ottenere un file di dimensioni ridotte mediante una estrazione casuale dei record presenti nel file originale si può utilizzare il piccolo programma KB_RND presente in appendice 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
```

Indicare in **InputFile** il file dal quale estrarre il file di output di dimensioni ridotte (**OutputFile**).
Indicare in **Out number of cells** il numero delle celle (righe x colonne) del file di output.
Altre volte è conveniente cancellare, dal file iniziale di input, i record che palesemente contengono informazioni contraddittorie, assurde o mancanti: così facendo si riduce la dimensione del file e se ne migliora la qualità diminuendone il

rumore.

Avvertenze generali per l'uso dei programmi di KB

E' importante che i file che sono in input e di output all'elaborazione dei tre programmi kb_cat, kb_sta, kb_cla non siano aperti in altre *finestre* per operazioni di lettura o di scrittura: in caso contrario kb_cat, kb_sta, kb_cla andrebbero in errore di esecuzione.
L'elaborazione dei tre programmi può essere interrotta premendo insieme il tasto *ctrl* e il tasto *c*.

KB_CAT Estrazione della conoscenza dai dati e catalogazione in gruppi omogenei

Generalità, obiettivi e funzioni

KB_CAT è il primo dei tre programmi da utilizzare ed è il più importante.
Esso ha lo scopo di analizzare qualsiasi file di testo strutturato in forma di tabella bidimensionale contenente dati numerici e/o dati di testo.
KB_CAT:

- controlla che la tabella bidimensionale da elaborare non contenga errori di formato
- normalizza i dati numerici e i dati di tipo testo
- dà corso alla fase di addestramento ricercando l'errore minimo che decresce nel corso dell'elaborazione fino a raggiungere il valore minimo del parametro *alpha* scelto dall'utente
- terminata l'elaborazione, il programma scrive i file di output contenenti i risultati del calcolo, risultati che potranno essere utilizzati anche dagli altri due programmi KB_STA e KB_CLA.

Sorgente di KB_CAT (vedere allegato 1)

Input file di prova (copia e incolla poi memorizzare con nome *vessels.txt*); campi separati da tabulazione

description	shape	material	height	color	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	trasparent	17	no	no
wine_glass	cut_cone	glass	15	trasparent	15	no	no

description	shape	material	height	color	weight	haft	plug
jug	cylinder	terracotta	25	white	40	yes	no
bottle_1	cylinder_cone	glass	40	green	120	no	cork
bottle_2	cylinder_cone	glass	40	trasparent	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	trasparent	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	trasparent	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
tuna_can	cylinder	metal	10	severals	10	no	no
tuna_tube	cylinder	plastic	15	severals	7	no	plastic
perfume	parallelepiped	glass	7	trasparent	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	trasparent	4	no	no
trousse	cylinder	plastic	1	silver	7	no	yes

description	shape	material	height	color	weight	haft	plug
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	trasparent	300	two	glass
plant_pot	cut_cone	terracotta	30	brown	200	no	no
pasta_case	parallelepiped	glass	35	trasparent	150	no	metal

Modalità di utilizzo

Essendo posizionati nella cartella contenente kb_cat.py e il file di input da elaborare, si manda in esecuzione KB_CAT digitando in finestra DOS Windows (o nel Terminal di Linux), il comando

python kb_cat.py

dove con **python** si chiede l'esecuzione (con il linguaggio python) del programma ***kb_cat.py.***

Il programma inizia l'elaborazione chiedendo in successione:

Input File = *vessels.txt*

vessels.txt è il file in formato txt contenente la tabella dei record / casi da catalogare, riportato sopra.
Nel caso si volesse attribuire maggiore importanza ad una variabile / colonna, basta duplicarne i valori, una o più volte, in variabili / colonne aggiuntive: se voglio rendere la variabile *shape* importante per il triplo del suo peso iniziale, creerò altre due variabili / colonne chiamandole ad esempio *shape1* e *shape2* con valori identici a quelli originali di *shape*.

Number of Groups (3 – 20) = 3

il valore 3 è la radice quadrata del numero massimo dei gruppi di catalogazione (in questo caso 9); poiché la matrice di addestramento ha la base di forma quadrata; il numero massimo dei gruppi di addestramento può solo essere il quadrato del valore digitato.
Non esistono regole utili per fissare il numero ottimale del parametro *Number of Groups*: si consiglia di provare inizialmente con valori bassi ed eventualmente poi fare altre elaborazioni con valori superiori se si siano ottenuti dei gruppi contenenti troppi record e, al contrario, diminuire il valore del parametro *Number of*

Groups se si siano ottenuti troppi gruppi contenenti pochi record.
Alle volte invece, il ricercatore è interessato ad analizzare in particolare proprio i gruppi poco numerosi ma con caratteristiche rare e singolari: in tal caso i gruppi contenenti pochi record sono da privilegiare.

Normalization (Max, Std, None) = m

il valore m (M) indica la richiesta che i dati numerici siano normalizzati rapportandoli tutti al valore massimo della colonna / variabile, quindi i nuovi valori della variabile / colonna varieranno da un minimo di 0 ad un massimo di 1.
La normalizzazione basata sul valore massimo (M) si è dimostrata la migliore.
Il valore S indica la richiesta che i dati numerici siano normalizzati sottraendo ad ogni valore in input la media della variabile / colonna e dividendo il risultato per la *standard deviation* della variabile / colonna.
E' sconsigliato inserire il valore *N* (None) soprattutto in presenza di variabili fra di loro molto dissimili con ampia distanza fra valore minimo e valore massimo (*range*).

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

KB_CAT, come tutti gli algoritmi delle reti neurali, esegue dei cicli di calcolo facendo delle iterazioni di apprendimento su tutti i record / casi in input.
In tali iterazioni gioca un ruolo fondamentale il parametro *alpha* dal suo valore iniziale di partenza (Start Value) al suo valore finale di arrivo (End Value) tenendo anche conto del valore dello *step* di decremento.
Può succedere di constatare una durata eccessiva dell'elaborazione avendo scelto un numero elevato di gruppi per un file contenente molti record e con valori iniziali e finali di *alpha* molto distanti e con *step* di decremento di *alpha* molto piccolo; in questi casi normalmente si verifica che l'errore minimo resta immutato per molte iterazioni.
Si consiglia di fermare l'elaborazione, premendo la combinazione dei due tasti ***ctrl*** e ***c,*** e di ripeterla con valori dei parametri più adatti.

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

Il parametro *alpha* è utilizzato da KB_CAT per affinare la catalogazione di record nei vari gruppi: un valore basso di *alpha* comporta una maggiore durata del calcolo con la possibilità di ottenere un errore minimo finale più basso ma anche un'ipotetica maggiore possibilità di *over fitting* (*effetto fotografia*).

Decreasing step of (from 0.1 to 0.001) = 0.001

Scegliere il valore del passo decrescente di alfa da applicare ad ogni ciclo.

Chiusura forzata dell'elaborazione

Nel caso si volesse forzare la fine dell'elaborazione in corso, basta premere la combinazione dei due tasti ***ctrl*** e ***c***.

Naturalmente i file scritti fino a quel momento non saranno validi.

KB_CAT produce i seguenti output:

Nella finestra DOS Windows (o del 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
```

Come si può notare, nel corso della elaborazione, l'errore minimo è passato da 3.616 (epoca 15) a 1.859 epoca 941).

Le epoche totali elaborate sono state 1000, quando il valore del parametro alpha ha raggiunto il valore minimo di 0.001.

Sono elencati anche i riferimenti ai file di output:

- Catalog.original = file di input *catalogato,* NON ordinato in sequenza

di gruppo e con valori originali (NON normalizzati)

- Catalog.sort = file di input *catalogato*, ORDINATO in sequenza di gruppo e con valori originali (NON normalizzati)
- Summary.sort = file di input *catalogato*, ORDINATO in sequenza di gruppo e con valori NORMALIZZATI
- Matrix Catal. = file con tre colonne (numero progressivo di record, codice di gruppo e codice di sottogruppo)
- Means, STD, CV = file con una colonna per ogni variabile e con tre righe (media aritmetica, *standard deviation* e coefficiente di variazione)
- CV of the Groups = file dei coefficienti di variazione dei gruppi e delle variabili / colonne con indicazione del numero dei record catalogati nei gruppi
- Training Grid = file contenente i valori della matrice di addestramento con errore minimo
- Run Parameters = file contenente i riferimenti ai file di input, parametri di calcolo e file di output.

KIndex (Knowledge Index) è un indice di KB che misura quanta conoscenza sia contenuta nei gruppi catalogati: nel caso KIndex raggiunga il valore massimo di 1, ogni gruppo sarebbe composto da record con valori costanti in tutte le variabili / colonne e renderebbe ogni gruppo del tutto distinto dagli altri gruppi.
KIndex è calcolato utilizzando i CV medi-ponderati delle variabili / colonne dei gruppi rapportandoli al CV delle variabili / colonne del file di input prima della catalogazione: vedere il programma sorgente KB_CAT per i dettagli del calcolo.
Nel caso in esame, il valore, non particolarmente alto di KIndex (0.8438), suggerisce di eseguire una nuova elaborazione aumentando, ad esempio, il numero dei gruppi da 3 a 4 ottenendo un sicuro miglioramento di KIndex.

File di Output/Catalog.original (vessels_*M_g3_out.txt*)

E' identico al file in input con l'aggiunta della colonna di attribuzione del codice del gruppo di appartenenza.
E' più interessante il file di Output/Catalog.sort che mostra i record catalogati nei singoli gruppi di appartenenza.

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

E' identico al file precedente ma i record / casi sono ordinati in sequenza di codice del gruppo di appartenenza attribuito.

Group	description	shape	material	height	color	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

Group	description	shape	material	height	color	weight	haft	plug
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	trasparent	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	trasparent	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	trasparent	4	no	no
G_02_00	champagne_glass	cut_cone	crystal	17	trasparent	17	no	no
G_02_00	dessert_glass	cut_cone	glass	17	trasparent	17	no	no
G_02_00	glass_2	cut_cone	plastic	9	white	4	no	no
G_02_00	pasta_case	parallelepiped	glass	35	trasparent	150	no	metal
G_02_00	perfume	parallelepiped	glass	7	trasparent	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	trasparent	15	no	no
G_02_01	pyrex	parallelepiped	glass	10	trasparent	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

Da una prima osservazione si nota come il programma KB_CAT di catalogazione sia stato in grado di catalogare i record in gruppi omogenei nel contenuto.
Bisogna osservare che il file *vessels.txt* era formato da solo 40 record fra di loro abbastanza diversi.
A titolo di esempio:

- il gruppo G_00_00 è caratterizzato da oggetti prevalentemente di *green*

color, e con *haft*

- il gruppo G_00_02 è formato prevalentemente da oggetti di *cut_cone shape*, con *haft* e senza *plug*
- il gruppo G_02_00 è caratterizzato da oggetti di *parallelepiped / cylinder / cut_cone shape e* senza *haft*
- il gruppo G_02_02 è composto da oggetti di *plastic* e di *terracotta* con *haft*

Se il file di input elaborato fosse stato formato da numerosi record con molte variabili / colonne, non sarebbe stato così facile trarre delle conclusioni sui risultati della catalogazione esaminando visivamente il file con formato identico a quello sopra riprodotto.
Il programma KB_STA è dedicato a risolvere il problema appena evidenziato.

File di Output/Medie, Std, CV (*vessels_M_g3_medsd.txt*)

Il file contiene le Medie, i Massimi, gli Std e i CV dei valori normalizzati di tutta la *popolazione*.
Valori bassi dei CV (coefficienti di variazione) indicano che i valori della variabile / colonna sono poco dispersi.

shape	**material**	**height**	**color**	**weight**	**haft**	**plug**
Mean1 0.4892	Mean2 0.5000	Mean3 28.075	Mean4 0.6222	Mean5 530.32	Mean6 0.3000	Mean7 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

File di Output/CV (*vessels_M_g3_cv.txt*)

Groups	**shape**	**material**	**height**	**color**	**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

Groups	shape	material	height	color	weight	haft	plug	Means	N_recs
Total	1.51	1.63	2.16	1.32	11.51	3.82	1.11	3.29	40

Il file contiene le informazioni atte a misurare la qualità della catalogazione.
Il valore contenuto in **ogni cella** rappresenta l'importanza dei valori della variabile / colonna nel gruppo della riga considerata: più il valore è prossimo allo zero e maggiormente la colonna / variabile è significativa nella catalogazione.
Nel caso il valore sia uguale a zero, la variabile / colonna in quel gruppo avrà valori identici: esempio tutti i gruppi hanno valori identici nella variabile *haft.*
I valori contenuti nelle **celle della penultima colonna** (*Means*) indicano quanto i gruppi siano omogenei al loro interno considerando tutte le variabili / colonne: più il valore è prossimo allo zero e maggiore sarà la somiglianza dei record / casi fra di loro all'interno del gruppo in esame.
Sono omogenei al loro interno i gruppi G_00_02 e G_01_00, mentre non lo è il gruppo G_00_00 per la significativa variabilità delle variabili *weight* e *plug*.
E' opportuno anche confrontare i valori contenuti in ogni riga / colonna con il valore corrispondente contenuto nelle due ultime righe: **Means** e *Total* (riferito all'intera popolazione non ancora catalogata).

File di Output/Grid di addestramento (*vessels_M_g3_grid.txt*)

Il file contiene i valori della matrice tridimensionale di addestramento con errore minimo; questa matrice è utilizzata dal programma KB_CLA adibito alla classificazione di nuovi record / casi che possono essere riconosciuti e classificati sulla base di quanto in precedenza appreso dal programma 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

Group	SubGroup	Variable/Column	Values
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
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

Group	SubGroup	Variable/Column	Values
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

L'analisi statistica dei risultati della catalogazione

I file contenenti i risultati dell'elaborazione di KB_CAT sono stati sottoposti ad analisi statistica eseguendo il programma KB_STA, descritto nel seguito usando i parametri sotto indicati.

```
#############################################################################
# KB_STA KNOWLEDGE DISCOVERY IN DATA MINING (STATISTICAL PROGRAM)           #
# by ROBERTO BELLO (COPYRIGHT MARCH 2011 ALL RIGHTS RESERVED)               #
# Language used: PYTHON                                                     #
#############################################################################
INPUT - Cataloged 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
***  color    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
***  color  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 trasparent
G_00_02     ID record    pot_2           Value grey
Value  white           Frequency          2    Percentage   50.00
Value  trasparent      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
***  color    Consistency          0.2997       %Consistency            35.77
G_01_00     ID record    beer_jug        Value severals
G_01_00     ID record    bottle_2        Value trasparent
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  trasparent      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
```

```
***  color    Consistency          0.1507       %Consistency             71.57
G_02_00     ID record    cd               Value trasparent
G_02_00     ID record    champagne_glass  Value trasparent
G_02_00     ID record    dessert_glass    Value trasparent
G_02_00     ID record    glass_2          Value white
G_02_00     ID record    pasta_case       Value trasparent
G_02_00     ID record    perfume          Value trasparent
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 trasparent
Value  trasparent      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
```

```
***  color   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
***  color       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
```

Altro file di input a KB_CAT (*animals.txt*)

Il file *animals.txt* è formato da 84 record con 15 variabili / colonne (Fur, Feather,

Eggs, Milk, Flying, Aquatic, Predatory, Teeth, Vertebrate, Polmones, Poisonous, Flippers, Legs, Tail, Domestic).

ANIMAL	FUR	FEATHER	EGGS	MILK	FLYING	AQUATIC	PREDATORY	TEETH	VERTEBR.	POLMONES	POISONOUS	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
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

ANIMAL	FUR	FEATHER	EGGS	MILK	FLYING	AQUATIC	PREDATORY	TEETH	VERTEBR.	POLMONES	POISONOUS	FLIP.	LEGS	TAIL	DOM.
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
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

Elaborazione del file animals.txt con KB_CAT

Il file *animals.txt* è stato elaborato con i seguenti parametri di calcolo:

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
L'elaborazione è terminata con Errore Minimo di 5.255 all'epoca 971 producendo i risultati nei file:

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

Il file Output/Catalog.sort dei catalogati ordinati per gruppo, ottenuto da *animals.txt*

Group	ANIMAL	FUR	FEATH.	EGGS	MILK	FLYING	AQUAT.	PRED.	TEETH	VERT.	POLM.	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
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

Group	ANIMAL	FUR	FEATH.	EGGS	MILK	FLYING	AQUAT.	PRED.	TEETH	VERT.	POLM.	POIS.	FLIP.	LEGS	TAIL	DOM.
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
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

Group	ANIMAL	FUR	FEATH.	EGGS	MILK	FLYING	AQUAT.	PRED.	TEETH	VERT.	POLM.	POIS.	FLIP.	LEGS	TAIL	DOM.
G_03_03	SWAN	0	1	1	0	1	1	0	0	1	1	0	0	2	1	0

Il file Output/CV dei catalogati ordinati per gruppo, ottenuto da *animals.txt*

Groups	FUR	FEATHER	EGGS	MILK	FLYING	AQUATIC	PRED.	TEETH	VERT.	POLM.	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 questo esempio si nota come sia notevole la differenza del CV riferito all'intera popolazione (3,57) rispetto alla media dei CV dei gruppi (0,41).
La catalogazione ha prodotto un miglioramento del 88,5%.
Ulteriore conferma deriva dalla presenza di molti valori posti a zero nelle celle della tabella, valori che indicano che in quella variabile / colonna del gruppo esiste un unico dato costantemente ripetuto: la variabile / colonna (per quel valore costante) è certamente importante per la catalogazione dei record in quel gruppo.

Verificare un'ipotesi di catalogazione soggettiva non automatica

Alle volte può risultare utile verificare se una catalogazione manuale e soggettiva trovi conferma nella catalogazione automatica di KB_CAT.
Nell'esempio sotto descritto alle variabili / colonne già presenti è stata aggiunta la variabile / colonna D1 nella quale sono stati indicati dei codici di catalogazione della specie dell'animale; la variabile / colonna è stata poi duplicata in altre due colonne (D2 e D3).
La modifica al file originale ha due diverse conseguenze:

- rafforza l'importanza del codice di catalogazione rispetto a tutte le altre variabili / colonne
- la nuova struttura del file consente a KB_CAT di elaborare i dati in una modalità simile a quella degli algoritmi tipici delle reti supervisionate con le diverse variabili / colonne di input e una variabile / colonna obiettivo (nel nostro caso D1, D2, D3 considerate globalmente)

File di input a KB_CAT (*animali_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
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

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

L'elaborazione è avvenuta con i seguenti parametri:

```
#############################################################################
# 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
Numer 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
```

I risultati ottenuti elaborando animali_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
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

Group	ANIMAL	FUR	FEATHER	EGGS	MILK	XXX	D1	D2	D3
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

La catalogazione manuale e soggettiva risulta confermata salvo per i record dei gruppi G_00_00, G_00_01, G_01_02 e G_03_02.

Le domande che il ricercatore potrebbe porsi in casi analoghi, ma molto più complessi ed importanti nelle realtà operative delle imprese e delle organizzazioni, potrebbero essere:

- esistono errori nella rilevazione dei dati?
- esistono errori nell'introduzione dei dati?
- sono in atto delle mutazioni all'interno del gruppo?
- esistono delle difettosità di produzione / lavorazione?
- lo schema di catalogazione manuale è carente?
- è necessario introdurre altre variabili / colonne?

Analisi dei risultati della catalogazione del file degli Iris verso quella botanica

L'affidabilità degli algoritmi delle reti neurali è spesso valutata dalla coincidenza fra la catalogazione automatica del file *iris.txt* (150 records) con quella effettuata dai botanici.
Con KB_CAT è stata effettuata una catalogazione in 3 gruppi ottenendo i risultati che seguono.

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
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

Group	RecId	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
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
G_02_00	setosa18	5.1	3.5	1.4	0.3

Group	RecId	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
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

```
*Group* RecId          Sepal_Length Sepal_Width Petal_Length Petal_Width
G_02_02 virginica134   6.3          2.8         5.1          1.5
```

Con l'eccezione dei record evidenziati in giallo, la catalogazione automatica ha confermato quella dei botanici (inserita nella colonna RecId) raggiungendo un alto valore del Knowledge Index (0.9311).

Sperimentazione clinica riguardante il virus dell'epatite B

KB_CAT nel 2006 è stato utilizzato per elaborare i dati di un'importante ricerca di sperimentazione clinica riguardante 1414 soggetti per 33 variabili / colonne.
La ricerca riguardava il virus dell'epatite B, le caratteristiche dei portatori tipici, dei portatori sani, quelli a bassa replicazione virale, le possibili evoluzioni del virus in altre patologie, i *marker* di identificazione, le diagnosi e le terapie.
Il problema dell'identificazione consisteva nel determinare se una persona aveva caratteristiche tali da farla associare a un gruppo di persone portatrici del virus.
Le variabili / colonne che riguardavano le generalità dei soggetti, come l'età, la razza, il peso, l'altezza, le regioni di nascita e di residenza furono omesse perché risultate poco significative in precedenti elaborazioni.
Il peso e l'altezza dei soggetti risultavano poco indicativi se adottati separatamente. Per questo motivo fu calcolato l'indice di massa corporea, *BMI*, che mette in relazione i due attributi, attraverso il rapporto

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

Inoltre fu calcolato anche un altro indice, particolarmente significativo, che metteva in relazione peso, altezza, genere ed età del soggetto. Il vero indice di peso forma, che tiene conto anche della muscolatura e della corporatura dei soggetti è infatti la percentuale di massa grassa, *FAT*. Esistono diverse formule, molto simili tra loro, per calcolare tale indice. In questo caso fu presa in considerazione la formula di Deurenberg, che più delle altre tiene conto dell'età del soggetto:

$$FAT(\%)= (1,2 * BMI) + (0,23 * età) – (10,8 * genere) – 5,4$$
$$genere\ 1 = uomo,\ genere\ 2 = donna$$

Per quanto riguarda le variabili relative al *potus*, che indicavano il numero di bicchieri di vino/birra/superalcolici bevuti giornalmente e da quanto tempo i soggetti stavano bevendo, sono state calcolate le unità alcoliche, come prodotto tra i due valori. Di conseguenza fu eliminato il campo che indicava se il soggetto era astemio o meno.

Il file di input, dopo la rielaborazione, presentava per ogni caso gli attributi indicati nella seguente tabella:

Generalità	Età, Genere, BMI, FAT, Caso
Potus	UA_totali (Unità alcoliche totali)
Diagnosi	Diagnosi, Steatoepatite, Steatosi

Terapie	PreInterferone, PreLamivudina , PreAdefovir , PrePegInterferone, PreEntecavir, PreTecnofovir, Interferone, Peginterferone, Lamivudina, Adefovir, Tecnofovir
Esami di Laboratorio	AST, ALT, HBeAg, AntiHBe, AntiHBclgM, HBVDNAqualitativo, HBVDNAquantitativo, GentipoHBV, AntiDelta, HIV, AntiHCV, HCVRNAQualitativo, GentipoHCV

Le caratteristiche maggiormente significative dei gruppi più numerosi sono contenute nella seguente tabella.

Gruppo	M/F	FAT %	Caso	UA	Diagnosi	Adefo-vir	AST/ALT	HBeAg	AntiHBclgM
1_01	M	11-43	Prevalente	73000	Epatite Cronica	No	alti	Negativo	Negativo
1_03	M	1743	Prevalente	29200	Epatite Cronica	No	medi	Negativo	Non ricercato
2_01	M	1636	Prevalente	45000	Portatore in fase non replicativa	No	normali	Negativo	Negativo
2_08	F	2356	Prevalente	9000	Epatite Cronica e portatore in fase non replicativa	No	normali	Negativo	Negativo
3_01	M	2837	Prevalente	282800	HCC e Cirrosi	No	alti	Negativo	Negativo
3_04	M	1434	Incidente	64000	Epatite Cronica e portatore in fase non replicativa	No	alti	Negativo	Non ricercato
4_01	M	2838	Prevalente	73000	HCC e Cirrosi	No	alti	Negativo	Positivo / non ricercato
8_01	M	01/12/47	Prevalente	45600	Epatite Cronica	No	alti	Positivo	Negativo
8_08	M	1737	Prevalente	73000	Cirrosi	Si	medi	Negativo	Negativo

Da un'analisi completa dei risultati ottenuti, non pubblicabili nel dettaglio, sono emerse le conclusioni:

- le donne bevono meno degli uomini, di conseguenza soffrono di epatite cronica piuttosto che di cirrosi
- si presentano con diagnosi *HCC* solo uomini di età superiore ai 50 anni; tali soggetti non presentano *steatoepatite* positiva e la *statosi* non assume mai valori molto alti
- i portatori di virus in fase *non replicativa* riportano dei valori normali agli esami di laboratorio *AST e ALT*
- risultano *HIV positivi* quasi esclusivamente uomini con epatite cronica
- la diagnosi *cirrosi* è presente solo per soggetti di età superiore ai 40 anni
- i portatori in fase *non replicativa* hanno una percentuale di *casi incidenti* maggiore rispetto alle altre diagnosi
- i soggetti ai quali è stata diagnosticata l'*HCC* sono quasi interamente *casi prevalenti*
- i valori più alti di *ALT e AST* riguardano i soggetti ai quali sono state diagnosticate *cirrosi* o *HCC*
- quasi tutte le donne portatrici in fase *non replicativa* risultano astemie; non vale lo stesso per gli uomini
- i valori più bassi di *FAT* riguardano uomini ai quali è stata diagnosticata

epatite cronica e che presentano valori di unità alcoliche ridotti

Non bene, di più! (mail del 2006)

Caro Roberto,
pare proprio che il tuo software abbia fatto goal.
Come anticipato, ho spedito le conclusioni di E. a P. A. (epatologo e top-publisher, milanese ma operante in quel di Palermo) anche per il fatto che egli era tra i proprietari del database utilizzato e, soprattutto, che è stato colui che, su quei dati, ha effettuato le analisi clinico/statistiche più approfondite.
Mi ha chiamato stamane chiedendo quale medico avesse contribuito ad analizzare il database e a scrivere la relazione.
Quando gli ho spiegato che né tu né E. siete medici e che le conclusioni sono state tratte dal tuo software non ci voleva credere.
Tutte (leggasi TUTTE) le conclusioni sono corrette e coincidono con quelle tratte dalle analisi statistiche e da sottili considerazioni cliniche.
Ho allora speso due parole per spiegargli che il software è in fase di validazione e che sarebbe stata utile una sua collaborazione per questo.
Trattasi, in pratica, di verificare che le conclusioni del software siano parallele a quelle delle analisi clinico/statistiche anche in altri database e, fatto questo, si potrà stendere una pubblicazione a carattere Informatico/clinico che validerà, almeno in questo importante campo, il tuo applicativo.
Egli avrebbe piacere a collaborare con te in questo senso ed inizierebbe inviandoti uno o due database clinici già analizzati (da 800 a 8.000 casi) sui quali cimentare il piccolo.
Una grossa pacca sulla spalla e... complimenti!

KB_STA - L'analisi statistica dei risultati della catalogazione

Generalità, obiettivi e funzioni

KB_STA ha lo scopo di aiutare il ricercatore nell'analisi dei risultati dell'elaborazione.
KB_STA dimostra di essere indispensabile quando il file di input è di notevoli dimensioni sia come numero di record sia come numero di variabili / colonne.
Una valutazione solo visiva dei record contenuti in ogni gruppo sarebbe difficile o laboriosa e renderebbe evidente la necessità di sottoporre il file dei gruppi ad elaborazioni esterne gravose, complicate e di dubbio risultato.
KB_STA risolve il problema della *scatola nera* (*black box*) tipico degli algoritmi delle reti neurali.
KB_STA:

- sottopone ad analisi statistica il file dei CV dei gruppi
- valuta il grado di omogeneità dei gruppi al loro interno

- valuta l'importanza delle variabili / colonne nella catalogazione dei record nei gruppi
- raggruppa i record di ogni gruppo per ogni variabile / colonna in quartili (se valori numerici) o in tabelle di frequenza (se valori testuali)
- se richiesto, lista per ogni gruppo e per ogni variabile / colonna i valori originali di input dei record

Sorgente di KB_STA (vedere allegato 2)

Modalità di utilizzo

Essendo posizionati nella cartella contenente il programma kb_sta.py e i file di input da elaborare, si manda in esecuzione KB_STA digitando in finestra DOS Windows (o nel Terminal di Linux), il comando

python kb_sta.py

dove con **python** chiedo l'esecuzione (con il linguaggio python) del programma ***kb_sta.py***

Il programma inizia l'elaborazione chiedendo in successione:

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

vessels_M_g3_outsrt.txt è il file in formato *txt* contenente la tabella dei record / casi catalogati ed ordinati in sequenza di codice di gruppo.
Il file *vessels_M_g3_outsrt.txt* è uno dei risultati della precedente elaborazione con il programma KB_CAT.

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

vessels_M_g3_cv.txt è il file in formato *txt* contenente la tabella dei coefficienti di variazione dei gruppi.
Il file *vessels_M_g3_cv.txt* è uno dei risultati della precedente elaborazione con il programma KB_CAT.
E' importante che questo file e il precedente provengano dalla stessa elaborazione di KB_CAT.

Report File (output) : vessels_M_g3_report.txt

vessels_M_g3_report.txt è il file di output che conterrà l'analisi statistica dei risultati ottenuti dal precedente programma di catalogazione.
E' utile, per chiarezza, che il nome del file del rapporto inizi come i due precedenti, come appena sopra esemplificato; nel caso di analisi statistiche con diversi parametri, i nomi potrebbero variare nella parte terminale del nome (esempio _r1, _r2, _r3).

Group Consistency (% from 0 to 100) : 0

Parametro per richiedere la visualizzazione dei gruppi con percentuale di

omogeneità al loro interno non inferiore a quella indicata; è consigliabile effettuare la prima elaborazione con il parametro posto uguale a zero, che mostrerebbe tutti i gruppi, per poi usare un diverso parametro in relazione ai risultati ottenuti.
Un valore troppo alto di questo parametro potrebbe produrre una lista vuota.

Variable Consistency (% from 0 to 100) : 0

Parametro per richiedere la visualizzazione delle variabili, all'interno dei gruppi visualizzati, con percentuale di omogeneità della variabile non inferiore a quella indicata; è consigliabile effettuare la prima elaborazione con il parametro posto uguale a zero, che mostrerebbe tutte le variabili dei gruppi visualizzati, per poi usare un diverso parametro in relazione ai risultati ottenuti.
Un valore troppo alto di questo parametro potrebbe produrre una lista vuota di variabili.

Select groups containing records >= : 2

Parametro per richiedere la visualizzazione dei gruppi composti da almeno *x* record.
I gruppi formati da un solo record sono automaticamente omogenei al 100% al loro interno e per tutte le variabili / colonne.

Select groups containing records <= : 1000

Parametro per richiedere la visualizzazione dei gruppi composti da un numero di record inferiore ad x. Il parametro può essere utile per esaminare solo i gruppi contenenti pochi record.

Summary / Detail report (S / D) : d

Se il parametro ha il valore di s*/S*, il rapporto conterrà per ogni gruppo i valori di omogeneità (*consistency*), la numerosità assoluta e percentuale dei record catalogati nel gruppo.
Se il parametro ha il valore di d*/D*, il rapporto conterrà per ogni variabile numerica i valori dei quartili, mentre per ogni variabile di testo il rapporto conterrà la distribuzione delle frequenze dei diversi valori testuali.

Display Input Records (Y / N) : n

Se il parametro ha il valore di *n/N* non saranno visualizzati i record di input appartenenti ai gruppi, in caso contrario (*y/Y)* saranno visualizzati.

Esecuzione di KB_STA

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

```
Cataloged 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
```

Analisi dei risultati della catalogazione di *vessels.txt*

Esempio per il gruppo 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
***  color           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
================================================================================
```

Il gruppo G_00_00 è composto in prevalenza da contenitori di vetro (glass), con un'altezza (height) da 30 a 65 cm, di colore verde (green), con un peso (weight) fino a 3840 e senza manico (haft).

Analisi dei risultati di una catalogazione di un sondaggio politico del 2007

Il caso analizzato prende in considerazione un sondaggio politico, effettuato nei giorni 22 e 23 novembre 2007 a cura del Prof. Paolo Natale dell'Università Statale di Milano, facoltà di Scienze Politiche. Dopo una prima elaborazione dei dati senza evidenti risultati, il database è stato preparato eliminando i campi palesemente non

significativi e accorpando alcune variabili.
Sono stati eliminati i campi riguardanti l'ampiezza del comune di residenza, il giudizio sul peso della democrazia e della politica. Il campo riguardante la regione di residenza è stato modificato, raggruppando le regioni tra nord, centro e sud.
Il nuovo database di partenza contiene 982 record relativi alle persone alle quali è stato chiesto di sottoporsi alle domande del sondaggio contenente i campi, di seguito elencati:

- genere (uomo / donna)
- coalizione di fiducia indipendentemente dal voto (*ES estrema sinistra, SS sinistra, CS centro sinistra, CC centro, CD centro destra, DD destra, ES estrema destra,** non risponde*)
- professione
- credente (sì/no)
- religione
- aspettative sulla situazione economica dei prossimi 6 mesi
- giudizio sullo stato attuale dell'economia del paese
- tutela (su chi fare affidamento)
- sicurezza (percezione)
- previsione del vincitore di un'eventuale elezione a breve
- giudizio sull'operato del governo
- giudizio sull'operato dell'opposizione
- interesse politico
- coalizione di fiducia
- voto a breve
- partito votato nel 2006
- partito che si intende votare alle prossime elezioni
- Partito delle Libertà
- età
- regione
- titolo di studio
- frequenza alle funzioni religiose

Per l'elaborazione, con il programma di KB_CAT, sono stati richiesti 4 gruppi.
I risultati della catalogazione ottenuti da KB_CAT sono stati sottoposti ad elaborazione da parte di KB_STA.
Da un'analisi dei risultati ottenuti emergono le seguenti considerazioni:

- i sostenitori della sinistra ritengono, con l'eccezione di quelli del gruppo G_04_02, che il centro destra sarà il futuro vincitore, oppure non rispondono su chi saranno i futuri vincitori (gruppo G_02_04)
- i sostenitori del centro sinistra / sinistra difendono il governo ritenendo il suo operato *in media*
- i sostenitori del centro sinistra / sinistra danno un giudizio mediamente positivo dell'opposizione

- il gruppo G_04_04 è formato da persone apolitiche, agnostiche (o estremamente riservate) che preferiscono non esprimersi; sono pensionati e *non occupati* di età superiore ai 50 anni
- non esiste, come in passato, una relazione tra la professione dei soggetti, la condizione economica e la coalizione di fiducia
- la categoria dei pensionati è divisa tra chi ipotizza la vittoria del centro destra (gruppo G_02_03) e chi preferisce non rispondere (Gruppo G_02_04 e G_04_04)
- l'età non influisce nella catalogazione
- in tutti i gruppi le persone dichiarano di non volere dare il voto al PDL, anche nei gruppi nei quali le stesse persone ipotizzano che il vincitore sarà il centro destra (G_01_04, G_02_03, G_04_01)

Gran parte delle considerazioni sopra esposte sono state una conferma della perdita di valore delle ideologie legate allo *zoccolo duro* dell'appartenenza alla classe sociale, all'età, al grado di istruzione, al territorio di residenza, etc., caratteristiche in passato significative per l'orientamento politico degli elettori. Nel 2007 si cominciò a fare riferimento al concetto di *elettore liquido* quale soggetto non più *tifoso* di un partito o di uno schieramento ma capace di valutare i risultati delle azioni di governo e dell'opposizione e su tali valutazioni decidere se e come votare.
E' evidente che definire profili prevalenti degli elettori permette di formulare programmi elettorali specifici e non solamente ideologici, comunicandoli poi nei modi adeguati ai gruppi individuati.

KB_CLA – Classificazione di nuovi record

Generalità, obiettivi e funzioni

Il programma KB_CAT produce il file contenente la matrice di addestramento (esempio *vessels_M_g3_grid.txt*) che può essere utilizzata per classificare *al volo* nuovi record che abbiano delle caratteristiche analoghe a quelle dei record utilizzati nella precedente catalogazione.
Applicazioni di programmi di classificazione *al volo* sono molto utili quando si deve agire rapidamente tenendo in considerazione la conoscenza già acquisita e disponibile.
Applicazioni di classificazioni in *tempo reale* in applicazioni funzionanti si riscontrano, ad esempio:

- in ambito bancario / assicurativo per la prevenzione delle frodi
- nella telefonia mobile per individuare i clienti in procinto di passare alla concorrenza
- nel controllo dei processi produttivi e della qualità dei prodotti
- nelle aziende per prevenire casi di insolvenza dei clienti.

Sorgente di KB_CLA (allegato 3)

Modalità di utilizzo

KB_CLA richiede che il file dei nuovi record / casi da classificare abbia la stessa struttura e simile contenuto del file utilizzato nella precedente elaborazione di catalogazione.
Per stessa struttura si intende che il file dei nuovi record / casi deve avere lo stesso numero di variabili / colonne con identici formati dei dati (numerici / testo).
Per simile contenuto si intende che il file dei nuovi record / casi deve contenere record provenienti da campioni dello stesso universo / ambito.
Una conoscenza acquisita per la catalogazione degli animali, non può essere utilizzata per classificare dei nuovi contenitori!

Essendo posizionati nella cartella contenente il programma kb_cla.py e i file di input da elaborare, si manda in esecuzione KB_CLA digitando in finestra DOS Windows (o nel Terminal di Linux), il comando

python kb_cla.py

dove con **python** chiedo l'esecuzione (con il linguaggio python) del programma ***kb_cla.py***

Il programma inizia l'elaborazione chiedendo in successione:

File di Input = n_vessels.txt

Contenuto del file n_vessels.txt

I record / casi da classificare sono riportate nella tabella seguente e sono identificati dal primo carattere *N* nella descrizione.

description	shape	material	height	color	weight	haft	plug
n_glass	cut_cone	terracotta	6	trasparent	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	trasparent	36	yes	no
n_coffee_cup	cut_cone	glass	6	trasparent	19	yes	no
n_perfume	cylinder	glass	7	trasparent	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	trasparent	130	no	metal

Number of Groups (3 – 20) = 3

Indicare lo stesso numero dei gruppi utilizzato nel precedente programma KB_CAT.

Normalization(Max, Std, None) = m

Indicare lo stesso tipo di normalizzazione utilizzato nel precedente programma KB_CAT.

File Training Grid = vessels_M_g3_grid.txt

Indicare il nome del file di output ottenuto dal precedente programma KB_CAT.

Esecuzione di KB_CLA

```
################################################################################
# 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
```

Analisi dei risultati della classificazione di n_vessels.txt

I nuovi record, riconoscibili dal primo carattere *N*, sono stati inseriti nella precedente tabella ottenuta da KB_CAT.

Group	description	shape	material	height	color	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	trasparent	22.0	no	no

Group	description	shape	material	height	color	weight	haft	plug
G_00_00	n_pasta_case	cylinder	glass	30.0	trasparent	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	trasparent	35.0	yes	no
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	trasparent	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	trasparent	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	trasparent	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	trasparent	36.0	yes	no
G_02_00	cd	parallelepiped	plastic	1.0	trasparent	4.0	no	no
G_02_00	champagne_glass	cut_cone	crystal	17.0	trasparent	17.0	no	no
G_02_00	dessert_glass	cut_cone	glass	17.0	trasparent	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	trasparent	150.0	no	metal
G_02_00	perfume	parallelepiped	glass	7.0	trasparent	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	trasparent	15.0	no	no
G_02_01	pyrex	parallelepiped	glass	10.0	trasparent	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

I nuovi record sono stati classificati, nella quasi totalità, in modo corretto con l'eccezione dei due record evidenziati in colore rosa.
Il record *n_glass* è stato classificato nel gruppo G_00_00 con le variabili *color* e *weight* aventi valori non presenti negli altri record del gruppo.

Opinioni politiche in Facebook (ricerca del gennaio 2013)

Si è esaminato un campione di 1070 opinioni politiche presenti in 14 diversi gruppi di discussione: 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.

Di ogni opinione si è considerato l'argomento della discussione (es. matrimonio e adozione dei gay, intervista TG3, deriva morale, etc.), il politico interessato (Bersani, Casini, Berlusconi, etc.) valutando l'atteggiamento desumibile dal giudizio espresso dall'autore del post utilizzando una scala di 5 valori: 1 = giudizio ingiurioso, 2 = giudizio negativo, 3 = giudizio indifferente, 4 = giudizio positivo, 5 = giudizio elogiativo.
La ricerca aveva l'obiettivo di scoprire le eventuali relazioni intercorrenti fra i gruppi di discussione, gli argomenti, i politici e i giudizi espressi.
KB_CAT ha catalogato le 1070 opinioni in 25 gruppi dei quali 15 contengono un numero significativo di opinioni.

Gruppo 00 Records 132
gruppi fb_corsera, fb_fanpage
politici Berlusconi, Dell'Utri, Bersani
argomenti candidato-sempre, deriva morale, euro100000 (assegno a Veronica)
giudizio ingiurioso, negativo
commento gruppi non “schierati” e sopratutto nel gruppo fb_corsera abbondano i giudizi oltraggiosi
---
Gruppo 04 Records 115
gruppi fb_maroni, fb_storace, fb_meloni
politici Maroni, Storace, Meloni
argomenti diario
giudizio positivo, elogiativo
commento sono gruppi “schierati”

Gruppo 24 Records 85
gruppi fb_fanpage, fb_grillo, fb_fattoquotidiano
politici Grillo, Ingroia
argomenti diversi
giudizio positivo
commento 2 gruppi non “schierati” ma Grillo e Ingroia sono le novità che attirano
---
Gruppo 02 Records 69
gruppi fb_brambilla, fb_casini, fb_bersani
politici Brambilla, Casini, Bersani
argomenti diversi
giudizio positivo, elogiativo
commento sono gruppi “schierati”, inoltre la missione animalista “premia”

Gruppo 40 Records 69
gruppi fb_termometro_politico
politici Berlusconi, Ingroia
argomenti servizio pubblico, successo, tasse
giudizio oltraggioso, negativo

commento gruppo non “schierato”, politici e argomenti che scottano
---
Gruppo 44 Records 66
gruppi fb_corsera, fb_meloni, fb_fanpage
politici Pannella, Meloni, Vendola
argomenti alleanza, diario, attacco
giudizio oltraggioso, negativo
commento gruppo fb_corsera non “schierato” e alleanza di Pannella con Storace non gradita
---
Gruppo 43 Records 66
gruppi fb_fanpage
politici Monti
argomenti no_matr_adoz_gay, monti_su_fb
giudizio oltraggioso, negativo
commento gruppo fb_fanpage non “schierato” e dissenso su no_matr_adoz_gay
---
Gruppo 12 Records 58
gruppi fb_bonino, fb_brambilla, fb_casini
politici Bonino, Brambilla, Casini
argomenti presidente_repubblica, manifesto_animalista
giudizio positivo
commento gruppi “schierati”

Gruppo 11 Records 51
gruppi fb_casini, fb_bersani
politici Casini, Bersani
argomenti le11bugie, intervistaTG3, intervistaTG5
giudizio negativo
commento critiche a gruppi “schierati” su opinioni non condivise
---
Gruppo 23 Records 44
gruppi fb_dipietro, fb_casini
politici Di Pietro, Casini, Grillo
argomenti spot_tv, ultima_parola
giudizio positivo, elogiativo
commento gruppi “schierati”

Gruppo 42 Records 43
gruppi fb_fanpage, fb_corsera
politici Monti, Grillo
argomenti pifferaio, profilo_fb, monti_esorcista
giudizio oltraggioso
commento gruppi non “schierati”

Gruppo 14 Records 40

gruppi fb_fanpage, fb_grillo
politici Monti, Grillo
argomenti no_matr_adoz_gay, raduno_lecce
giudizio positivo, elogiativo
commento giudizi elogiativi sui due argomenti di Monti e di Grillo
---
Gruppo 21 Records 39
gruppi fb_bonino, fb_casini
politici Bonino, Casini
argomenti regionali, patto_monti_bersani, presidente_repubblica
giudizio positivo
commento gruppi “schierati”

Gruppo 20 Records 33
gruppi fb_fanpage
politici Bersani
argomenti patto_monti_bersani
giudizio oltraggioso negativo
commento gruppo non “schierato”

Gruppo 31 Records 32
gruppi fb_di_pietro
politici Di Pietro
argomenti rivoluzione_civile, spot_tv
giudizio oltraggioso negativo
commento gruppo “schierato” e dissenso su rivoluzione_civile

Sintesi

Esistono strette relazioni fra tipologia dei gruppi, leader politici, argomenti e giudizi.

Nei gruppi “schierati”:

- abbondano i giudizi positivi ed elogiativi
- gli eventuali dissensi provengono da simpatizzanti che non condividono alcune posizioni politiche o da avversari immediatamente emarginati nelle discussioni
- il turpiloquio è raro e la forma sintattica e grammaticale è accettabile

Nei gruppi non “schierati”:

- il dissenso prevale di molto sul consenso
- il turpiloquio è la norma e la forma sintattica e grammaticale è scadente
- si *respira* la consapevolezza di trovarsi fra persone tolleranti
- si aprono discussioni su argomenti che i gruppi “schierati” vorrebbero nascondere

Know4Business (versione Cloud in Google App Engine)

Giulio Beltrami, software engineer ed esperto in innovative architetture ICT di tipo sociale, ha trasferito KB nell'ambiente Cloud Computing di Google App Engine con la denominazione **Know4Business**.
Know4Business è fruibile in modalità *pay per use* ed è raggiungibile in Internet al link http://know4business.appspot.com/.

Know4Business adds to Google-Apps a powerful general-pourpose discovering of the hidden knowledge, in your business data, enabled by a well-known neural-network self-learning data-mining algorithm.
Know4Business provides 5 tools, to fulfill the knowledge discovery:

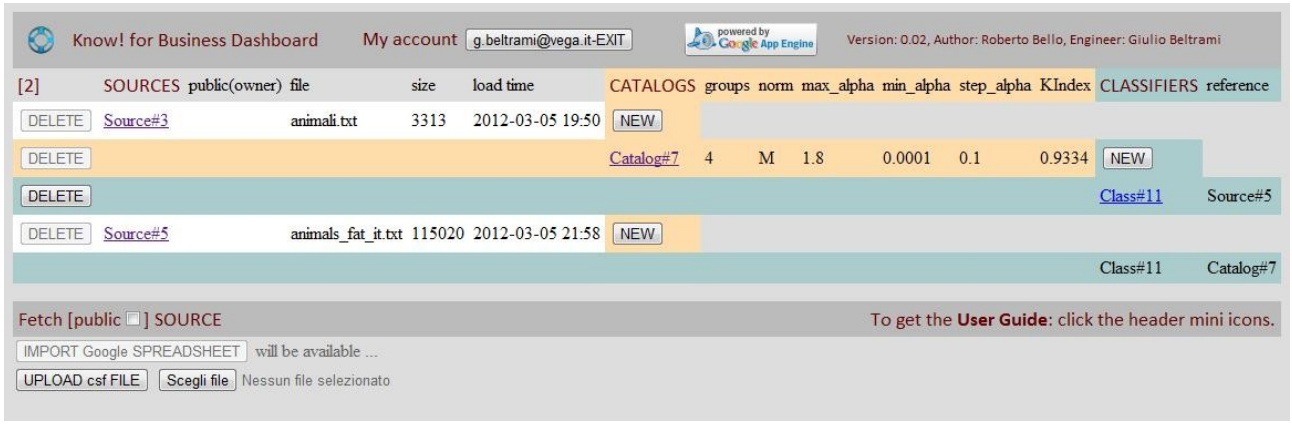
- SOURCE to preparing (checking, normalizing, cleaning, filtering and encoding) the input data.
- CATALOGUE to discovering groups in the sample data, that are in some way or another "similar", without using known structures.
- STATISTICS to evaluate the goodness of the catalogue clustering.
- CLASSIFIER to generalizing known structure, to apply to new data.
- AGENCY to return some diagnosis and suggestions, about the user data management, checking the goodness of the classification. plus a interactive CONSOLE to help the data-analyst to:
    - Driving the catalogue and the other tools
    - Reporting and graphing the results of the tools

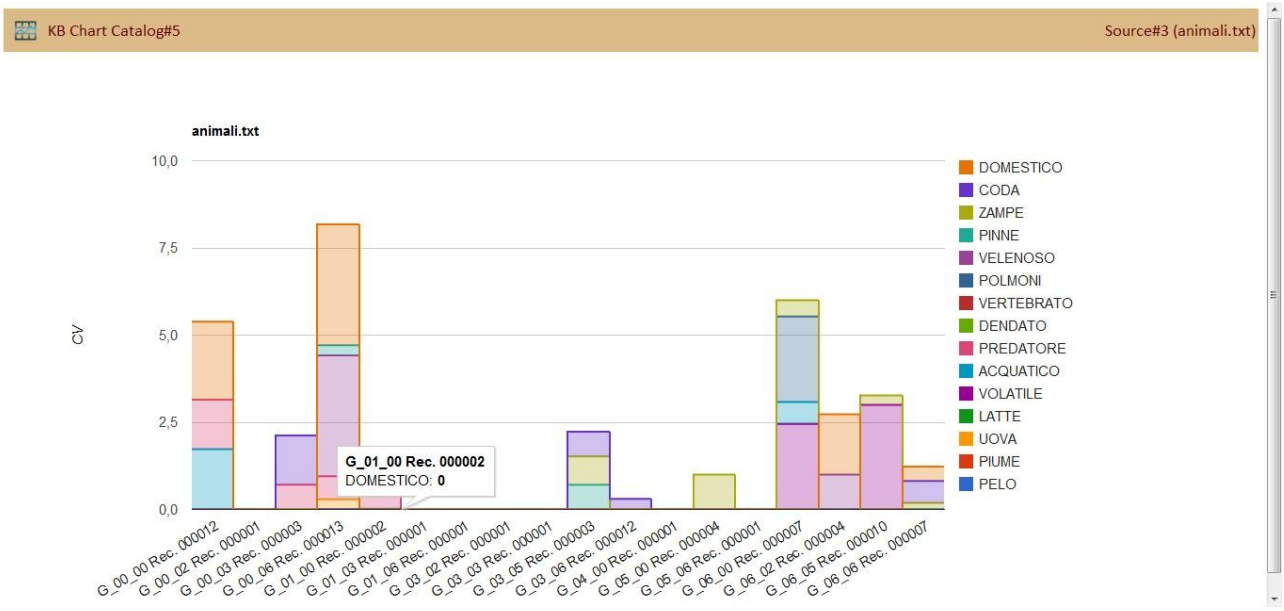
Know4Business end-to-end process of knowledge discovery, provides a simple work-flow, with some useful feedback capabilities:

- **Classification** operates forward to the catalogue of the sample data.
- **Statistics** about the catalog 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. witch enables a kind of data knowledge “auto-poiesis”, minimizing the 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.





APPENDICI

Appendice 1 – Sorgente KB_CAT

```
# -*- 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

def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        return False

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 is_number(arr_c[ind_c]):
      tipo_num = 1
    else:
      tipo_txt = 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
```

Appendice 2 – Sorgente KB_STA

```
# -*- 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
```

Appendice 3 – Sorgente KB_CLA

```
# -*- 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
```

Appendice 4 – Sorgente KB_RND

```
# -*- 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 – Garanzie, sicurezza e copyright

L'autore garantisce l'opera esente da virus e da codici maligni anche in considerazione che:

- il testo è in formato pdf
- i programmi python sono in formato testo e non contengono codice maligno, come facilmente verificabile da una loro semplice lettura
- il linguaggio per l'elaborazione dei programmi (python) è di tipo Open Source ed è universalmente riconosciuto come affidabile e sicuro.

Riguardo al copyright, l'autore non intende rinunciare ai suoi legittimi diritti sugli algoritmi e sulle modalità di calcolo e di analisi dei dati contenuti nei programmi componenti KB.

Roberto Bello

Laureato in Economia e Commercio con specializzazione in Operations Research
Data Scientist, esperto in Knowledge Mining, in Data Mining e nei linguaggi di programmazione Open Source
ICT Strategist del ClubTI di Milano (www.clubtimilano.net)
Ricercatore dell'AISF (www.accademiascienzeforensi.it)
Perito (CTP) ed ex CTU (Consulente Tecnico di Ufficio) del Tribunale di Milano
Autore di pubblicazioni professionali disponibili in www.lulu.com/spotlight/robertobb
Socio fondatore dell'AIPI (Associazione Italiana Professionale di Informatica)
In passato CIO della Plasmon, della Wrangler in Italia e consulente delle più importanti aziende alimentari italiane
Linkedin: it.linkedin.com/pub/roberto-bello/4/1a5/677

KB - Neural Data Mining con sorgenti Python