UNIVERSITÀ DEGLI STUDI DI NAPOLI

FEDERICO II

Scuola Politecnica e delle Scienze di base

Area didattica Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Tecniche di Clustering basate sul Machine Learning

Tesi sperimentale di Laurea Triennale

Tutor Accademico
Prof. Roberto Prevete

Tutor Aziendale
Dr. Massimo Brescia

Candidato
Francesco Esposito
matr. N86/24

ANNO ACCADEMICO 2012/2013

INDICE

INDICE DELLE TABELLE

INDICE DELLE FIGURE

ABBREVIAZIONI ED ACRONIMI

A&A	Meaning
API	Application Programming Interface
BMU	Best Matching Unit
BP	Back Propagation
CC	Componenti Connesse
CHL	Competitive Hebbian Learning
CSV	Comma Separated Values
DAME	DAta Mining & Exploration
DAMEWARE	DAME Web Application Resource
DBMS	DataBase Management System
DB	Davies-Bouldin
DCS	Dynamic Cell Structure
DM	Data Mining
DMM	Data Mining Model
DRMS	Driver Management System
E-SOM	Evolving SOM
E-TREE	Evolving TREE
FCPS	Fundamental Clustering Problems Suite
GCS	Growing Cell Structures
GNG	Growing Neural Gas
GRID	Global Resource Information Database
GUI	Graphical User Interface
GPGPU	General Purpose Graphical Processing Unit
HW	Hardware
ICA	Index of Clustering Accuracy
ICC	Index of Clustering Completeness
ML	Machine Learning
MLP	Multi Layer Perceptron
MDS	Massive Data Sets
NN	Neural Network
OOP	Object Oriented Programming
REDB	Registry & DataBase
SOM	Self Organizing Map
SP	Spazio dei Parametri
SPE	Specifications
SRS	Software Requirements Specification
SVM	Support Vector Machine
SW	Software
TBC	To Be Completed
TBD	To Be Defined
TWL	Two Winners Linkage
UML	Unified Modeling Language
VO	Virtual Observatory
XML	eXtensible Markup Language
WTA	Winner Takes All
WTM	Winner Takse Most

Tabella 1 – Abbreviazioni ed acronimi

1 INTRODUZIONE

La teoria delle reti neurali si basa sui modelli computazionali, introdotti negli anni '40, da McCulloch & Pitts (1943), che riproducevano in maniera molto semplificata il funzionamento di un neurone.

In generale le reti neurali sono modelli auto-adattivi di calcolo, basati o sul concetto di apprendimento da esempi o su apprendimento non supervisionato.

Per apprendimento supervisionato si intende un tipo di addestramento della rete tramite coppie input/target che sono esempi conosciuti di risposta della rete, per uno specifico problema, in particolari punti dello spazio dei parametri del problema stesso (classificazione, approssimazione o regressione di funzioni).

In taluni casi non esiste però la possibilità di avere serie di dati relativi alla soluzione del problema, ma esistono invece dati da analizzare senza avere a priori un "esperto", un supervisore che possa dire quale sia la risposta giusta per specifici valori in ingresso. Un tipico problema di questo tipo è quello di cercare raggruppamenti di dati aventi caratteristiche similari, per cui associabili, all'interno di un gruppo disordinato di dati (clustering).

Questo problema può essere affrontato con una rete neurale auto-organizzante, cioè in grado di interagire con i dati, addestrando se stessa senza un supervisore che fornisca soluzioni in punti specifici nello spazio dei parametri. I principali utilizzi di queste reti sono, oltra al già citato clustering, il riconoscimento di immagini o segnali. E' da notare che sulla base di tali reti sono state anche realizzate molte applicazioni per ottimizzazione di processi industriali. La Figura 1 è una rappresentazione generale dell'apprendimento di tipo non supervisionato.

L'apprendimento dei modelli non supervisionati è solitamente legato alla necessità di visualizzazione dello spazio dei dati durante l'evoluzione dell'addestramento. I tool di visualizzazione tipicamente sono dedicati per:

- Diagrammi di stato per processi complessi di analisi dati;
- Applicazioni di data mining: grafici, tabelle statistiche, segmentazione di immagini etc.;

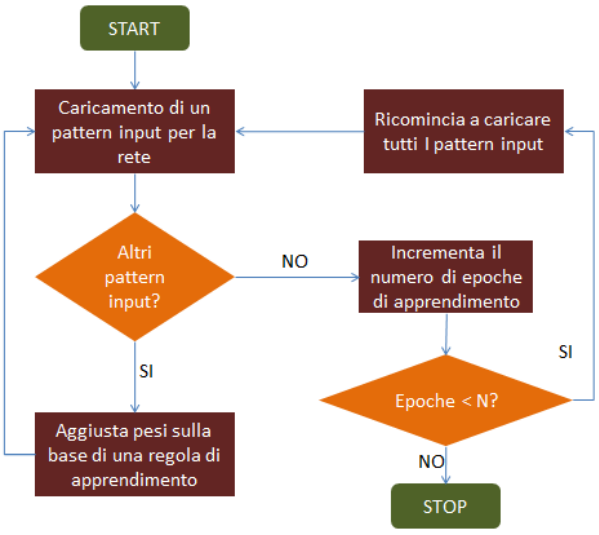


Figura 1 – Schema di una generica architettura *non supervisionata* di una rete neurale

In particolare, i modelli elaborati e sviluppati in questo lavoro sono utilizzati per l'esplorazione di grandi archivi di dati derivanti da osservazioni/simulazioni astronomiche, generalmente sotto forma di immagini 2D (*CCD stacked frames*), 3D (Voxel o Datacube) o di tabelle di dati (cataloghi di oggetti estratti astrometricamente attraverso la riduzione di immagini/spettri). Ciò sebbene la loro implementazione permetta di affrontare qualunque problema del mondo reale.

I modelli descritti in questo lavoro saranno utilizzati all'interno di un progetto più ampio che prende il nome di DAMEWARE (DAME Web Application Resource). Tale progetto riguarda una web application dedicata ad esperimenti di data mining mediante tecniche di machine learning. Il presente lavoro intende quindi estendere la dotazione di modelli non supervisionati di clustering e riduzione di dimensionalità, nonché di modelli supervisionati di classificazione, in seno al progetto, rendendoli fruibili alla comunità esterna attraverso un semplice browser web.

Il principale oggetto dell'utilizzo di tali modelli riguarda in particolare il clustering di dati tabulari ed immagini ad alta risoluzione, implicitamente legato anche alla riduzione della dimensionalità dello spazio dei parametri (*features extraction*), in modo da escludere dal dataset di partenza i parametri non influenti in termini di entropia informativa per la risoluzione dello specifico problema. Al fine di rendere i modelli innestabili all'interno della suddetta suite, è stata necessaria l'implementazione software ex novo di tutti i modelli, in modo da rispettare gli specifici requisiti imposti dalla piattaforma di destinazione.

Una delle migliori formulazioni di apprendimento non supervisionato è senza dubbio l'algoritmo di apprendimento proposto da Teuvo Kohonen (2001) nell'ambito delle reti auto-organizzanti (Self Organizing Maps o SOM), da lui ideate. Tale modello si basa sull'utilizzo di una griglia di nodi, a due o tre dimensioni, che costituisce lo strato di output della rete, o strato di Kohonen.

Affinché questo modello possa però essere utilizzato come effettivo metodo di clustering, è necessaria una fase di post-processing in cui i nodi della mappa di Kohonen vengano raggruppati in cluster in base a criteri di similarità. Come vedremo in seguito, questo risultato può essere ottenuto tramite l'ausilio di differenti tecniche, definite di post-processing, ovvero innestate all'interno di un metodo gerarchico di clustering, in cui variare l'ultimo stadio con differenti metodi, fra cui uno completamente nuovo e proposto in questo lavoro.

Inoltre, il modello SOM rappresenta il punto di partenza per diversi filoni evolutivi di modelli di clustering, la maggior parte dei quali si propone di superare uno dei più grossi limiti del modello di Kohonen, ovvero la struttura rigida della griglia dello strato di output. Tra le diverse evoluzioni della SOM ad architettura incrementale, quelle più interessanti, per i motivi che vedremo in seguito, risultano essere l'Evolving SOM e l'Evolving TREE, per i quali il presente lavoro ha progettato e sviluppato i rispettivi algoritmi.

Lo scopo di questo lavoro consiste dunque nell'estendere la Suite DAMEWARE[1], fornendogli:

- un modello unsupervised per clustering/dimensional reduction: Self Organizing Maps (SOM);
- una libreria di modelli di clustering da utilizzare come raffinamento (post-processing) della SOM, ispirandosi a metodi esistenti in letteratura: SOM + K-means, SOM+UmatCC (U-matrix a Componenti Connesse)
- un modello di raffinamento del clustering della SOM completamente nuovo: SOM + TWL (Two Winners Linkage);
- un modello unsupervised per clustering/dimensional reduction, evoluto dal paradigma SOM: Evolving SOM (E-SOM);
- un modello supervised per classificazione, evoluto dal paradigma SOM: Evolving TREE (E-TREE);
- una libreria di funzioni di pre- e post-processing specializzate per il clustering di immagini astronomiche (di tipo FITS), generiche (di tipo Jpeg, GIF, PNG) e/o di tabelle di pattern estratti da immagini astronomiche. Tali funzioni permetteranno anche la visualizzazione dell'output grafico dello strato di Kohonen delle SOM e relativi metodi di post-processing e della griglia dinamica dell'E-SOM.

[1] http://dame.dsf.unina.it/

L'originalità del presente lavoro si basa sui seguenti aspetti:

- ✓ vengono messi a confronto diretto tre tecniche di clustering:
  - **Two-stage clustering**, metodo gerarchico, composto dalla sequenza **SOM + algoritmo di clustering**, quest'ultimo implementato in tre diversi modi:
    - **K-Means**: metodo di raffinamento del clustering della SOM, basato sul partizionamento dello spazio dei parametri;
    - **U-matrix a componenti connesse** (Umat-CC): metodo di raffinamento del clustering della SOM, basato sulla tecnica della U-matrix;
    - **Two Winners Linkage** (TWL): metodo di aggregazione, basato sull'analisi dei BMU forniti dalla SOM. Il metodo si ispira ad uno dei passi dell'algoritmo E-SOM e rappresenta un sistema innovativo proposto in questo lavoro;
  - **Evolving SOM** (E-SOM), modello derivato dalla SOM, caratterizzato da uno sviluppo dinamico della griglia neurale di Kohonen. Esso dunque appartiene alla famiglia dei modelli di clustering incrementale derivati dalla SOM, in cui viene evoluta la funzione di addestramento;
  - **Evolving TREE** (E-TREE), modello derivato dalla SOM, caratterizzato da uno sviluppo dinamico ad albero decisionale. Esso dunque appartiene alla famiglia di modelli di clustering dinamico derivati dalla SOM, in cui viene evoluta la topologia della rete neurale;

- ✓ Nell'ambito dello sviluppo del modello Umat-CC, viene proposto un metodo innovativo di rappresentazione grafica dell'effetto di clustering sulla U-matrix, basato sulla colorazione del gradiente topologico;
- ✓ Il metodo di visualizzazione sviluppato è stato progettato per essere adattato anche alla rappresentazione grafica dell'output del modello E-SOM. In questo caso la regola basata sul gradiente topologico è stata modificata ad hoc per sopperire alle differenze intrinseche tra SOM ed E-SOM;
- ✓ L'impiego di tali tecniche per il clustering di immagini astronomiche rappresenta una novità assoluta.

Il presente documento è organizzato come segue: il capitolo 2 è dedicato alla descrizione delle tecnologie utilizzate, inclusa l'infrastruttura software entro cui i vari modelli sono innestati e resi fruibili alla comunità. Il capitolo 3 elenca tutti i requisiti funzionali e implementativi per i vari modelli sviluppati, elaborati durante la fase di progettazione degli algoritmi. I capitoli dal 4 al 7 sono dedicati alla descrizione teorica e funzionale dei singoli modelli di data mining sviluppati, rispettivamente, SOM, le 3 tecniche di post-processing della SOM (K-means, Umat-CC e TWL), il modello E-SOM e il modello E-TREE. Il capitolo 8 racchiude la parte ingegneristica relativa all'organizzazione del software per i vari modelli e alla loro interfaccia con l'utente finale. Il capitolo 9 riporta vari test per ogni modello e relativi confronti tra loro e con modelli esterni (K-means, MLP e SVM) in termini di prestazioni. Infine diverse appendici riportano dettagli implementativi sui vari algoritmi e sull'uso di software esterno.

2 TECNOLOGIE SOFTWARE IMPIEGATE

Questo capitolo è dedicato alla descrizione delle principali tecnologie software impiegate durante lo sviluppo dei componenti software, nonché tenuti in conto nella fase di progettazione dei vari moduli del progetto.

2.1 LA PIATTAFORMA OSPITE: DAMEWARE

2.1.1 Architettura generale

Il progetto DAMEWARE[1] (DAME Web Application REsource) è uno dei servizi web 2.0 inclusi nella Collaborazione DAME (DAta Mining and Exploration), (Brescia et al. 2010). Questo progetto si propone di realizzare una suite service-oriented e web-based per effettuare esperimenti di data mining su Massive Data Sets (MDS) su piattaforme di calcolo distribuito, siano esse *resource-based* (GRID) sia *service-based* (CLOUD), nonché basate su programmazione sequenziale (multi-core) e parallela (GPGPU+CUDA).

L'architettura generale della suite si basa sulla suddivisione del sistema in componenti software, ciascuno preposto ad un particolare compito o espletamento di servizi e comunicanti tra loro attraverso lo scambio di documenti in formato XML (Figura 2):

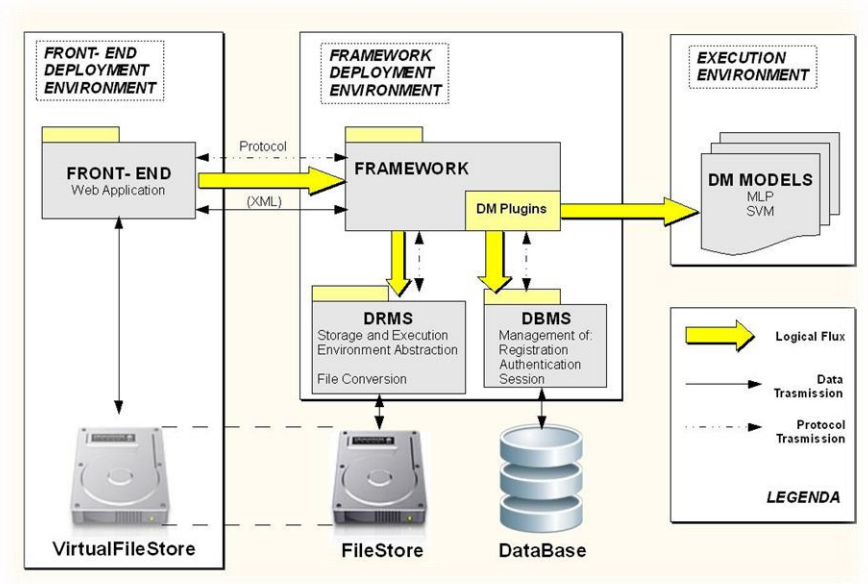


Figura 2 – Architettura modulare interna della suite DAMEWARE

Mostriamo in breve come avviene lo scambio di informazioni tra le varie componenti.

Il Front-End è la componente che si occupa direttamente delle interazioni tra la suite e l'utente finale tramite una GUI. L'utente, una volta registrato ed autenticato avrà a disposizione, tramite il proprio browser Web, strumenti per creare ed eseguire esperimenti di data mining (navigazione tra le proprie sessioni di lavoro e relativi esperimenti, scelta della funzionalità e relativo modello di data analysis, dowload ed upload di file, configurazione dei propri dataset, monitoraggio e download dei risultati, sia grafici che testuali).

Una volta lanciato un esperimento il Front-End comunica con il Framework per ottenere file di output, ma non solo: anche la lista delle sessioni e dei file di un utente, la lista delle funzionalità e descrizione delle stesse, nonché la lista degli esperimenti effettuati per ogni sessione (workspace).

[1] http://dame.dsf.unina.it/beta_info.html

Il Framework è il cuore dell'applicazione ed è stato pensato per poter funzionare su diverse piattaforme di calcolo: Cloud, Grid e Stand-Alone. Per realizzare tale requisito è stato sviluppato un componente apposito, denominato Driver Management System (DRMS), in grado di "virtualizzare" (quindi di astrarre) l'interfaccia tra il sistema operativo nativo dell'HW sottostante e lo strato di software della Suite DAMEWARE.

Il Framework è principalmente basato su:

1. Restful, stateless Web-service:gestisce e smista le richieste da/verso il Front-End;
2. Interfaccia di amministrazione: permette all'aministratore di installare e disinstallare funzionalità e di effettuare analisi statistiche sulla suite;
3. Xml generator: utilizzato per generare file Xml per interagire con il Front-End;
4. Data Types: oggetti che rappresentano entità fondamentali della suite (ad es. utente, esperimento, etc…);
5. Security: classi utilizzate per la sicurezza della suite;
6. Error Management: gestore degli errori che possono verificarsi durante l'utlizzo della suite;

I compiti del DRMS sono i seguenti:

- Gestione dei file fisici (upload, download, copia ed eliminazione);
- Conversione di un file in diversi formati;
- Esecuzione di un esperimento.

Per gestire tutte le informazioni relative agli utenti, le loro sessioni di lavoro, i vari files usati negli esperimenti etc., esiste il componente REDB (Registry & DataBase). E' primariamente composto da un DBMS relazionale ed implementato mediante sistema MySQL.

Infine, il componente che racchiude i modelli di data mining implementati (sottoforma di API gestite tramite interfaccia Java) è il DMM (Data Mining Model), in particolare oggetto di modifica/integrazione per il presente lavoro.

2.1.2 Uso della suite DAMEWARE

La piattaforma DAME è raggiungibile all' indirizzo: http://dame.na.astro.it:8080/MyDameFE/ che reindirizza l'utente alla pagina contenente informazioni in merito alla registrazione e all'autenticazione (Figura 3).

Figura 3 – Login DAMEWARE

Tramite gli appositi campi (Figura 3 - **a**), sarà possibile effettuare il login alla Suite. Inserite le credenziali si verrà reindirizzati alla schermata dalla quale è possibile effettuare le diverse operazioni (Figura 4).

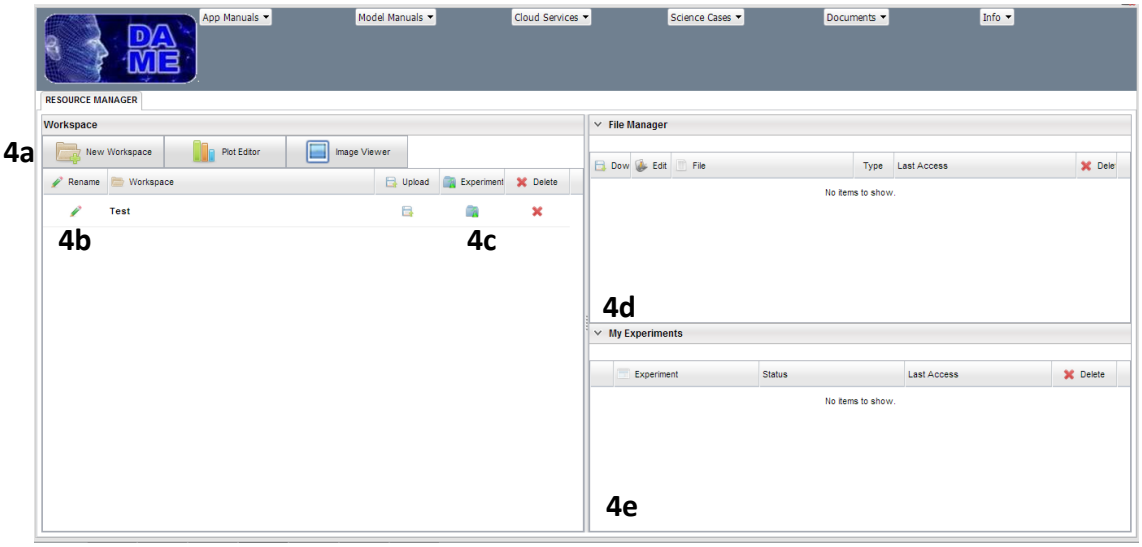


Figura 4 - Home Page DAMEWARE

In questa pagina l'utente avrà la possibilità di creare un nuovo Workspace, generare plot a partire da dati tabulari e manipolare immagini (Figura 4 - **a**). Al fine di avviare un nuovo esperimento, sarà necessario effettuare la prima delle operazioni appena elencate. A questo punto il workspace comparirà nella lista (Figura 4 **- b**) e sarà possibile aggiungervi un esperimento tramite l'apposito pulsante (Figura 4 **- c**).

Un esperimento viene generato scegliendo prima di tutto il nome. Successivamente occorrerà scegliere la funzionalità, il caso d'uso e i relativi parametri. Il tutto in una schermata simile alla seguente.

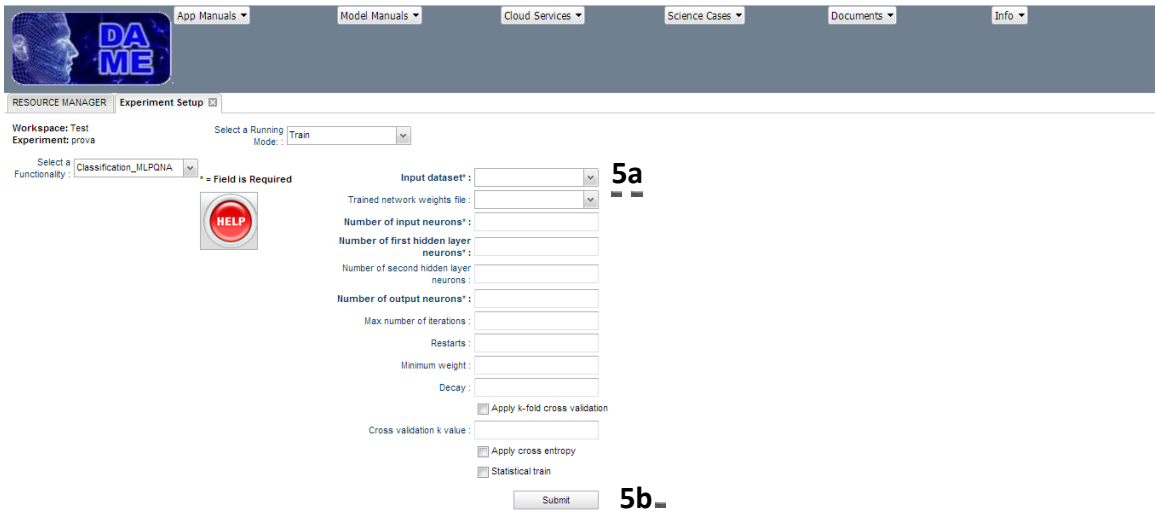


Figura 5 - Creazione di un esperimento in DAMEWARE

Tutti i file da passare in input ad un esperimento, come il dataset (Figura 5 - **a**), dovranno essere caricati all'interno dell'apposita sezione (Figura 4 - **d**). Una volta selezionati tutti i parametri desiderati l'esperimento potrà essere lanciato (Figura 5 **- b**) e al termine dell'esecuzione i risultati compariranno nella sezione dedicata (Figura 4 - **e**).

Per maggiori spiegazioni, guide pratiche e dimostrazioni è possibile consultare il canale del progetto al seguente indirizzo http://www.youtube.com/user/DAMEmedia.

NOTA: la guida appena fornita si riferisce alla release beta 3 dell'applicazione.

2.2 LIBRERIE ESTERNE

Di seguito vengono descritte le librerie esterne utilizzate nella progettazione e implementazione del codice.

2.2.1 CFITSIO

La libreria CFITSIO è stata inizialmente sviluppata dall' **HEASARC** (**High Energy Astrophysics Science Archive Research Center**) per convertire i dati astronomici acquisiti in formato FITS. Tale libreria, scritta in ANSI C e indipendente dalla piattaforma, fornisce infatti routine di alto livello per lettura e scrittura di file FITS isolando così il programmatore dalla complessità interna di questo tipo di formato.

Nella fattispecie, all' interno del programma, tale libreria è stata utilizzata al fine di convertire in formato testuale, immagini FITS date in input, in modo da poter poi analizzare i dati.

Per maggiori informazioni visitare http://heasarc.gsfc.nasa.gov/fitsio/

2.2.2 DevIL

DevIL sta per Developer's Image Library e, come suggerisce il nome, è una libreria contenente funzioni per la manipolazione di immagini. Tramite le funzioni di questa libreria open source è infatti possibile caricare, salvare, convertire, manipolare e applicare filtri ad una grande varietà di formati di immagini tra cui .ico, .png, .jpg, .png, .bmp ed altri ancora. All'interno del software la libreria risulta utile sia per la conversione in formato testuale di immagini input, sia per la generazione di immagini in output che mostrino l'effetto di clustering. DevIL è formato da tra librerie separate: IL, ILU e ILUT. Per i nostri scopi, sarà sufficiente l'uso della prima sotto-libreria citata. Per maggiori informazioni visitare http://openil.sourceforge.net/

2.2.3 Stilts

STILTS, ovvero **STIL** **T**ool **S**et, è una raccolta di strumenti, utilizzabili tramite linea di comando, basati sulla **S**tarlink **T**ables **I**nfrastructure **L**ibrary. Contiene una gran quantità di strumenti che riguardano manipolazione, ordinamento, visualizzazione e analisi di dati letti da file. Nel software la suddetta libreria è utilizzata per due differenti scopi: la conversione in formato ASCII di file CSV, VOTABLE e immagini FITS contenenti tabelle e la realizzazione di grafici a partire da file di testuali. Per maggiori informazioni visitare http://www.star.bristol.ac.uk/~mbt/stilts/

3 REQUISITI

Questo capitolo è dedicato alla sezione SRS (*Software Requirement Specifications*) del progetto. Anziché dedicare un documento separato, per semplicità, l'analisi dei requirements è inglobata nel presente documento.

REQUISITI SCIENTIFICI		
ID	REQUISITO	VERIFICA
R1	Lo scopo primario del presente progetto è estendere la Suite DAMEWARE (in particolare il suo componente DMM), mediante integrazione di nuovi modelli di machine learning unsupervised. Nella fattispecie il modello Self Organizing Map (SOM) e relative tecniche di post-processing (K-means, UmatCC e TWL), il modello E-SOM ed il modello E-TREE.	*Capitoli 2 e 8*
R2	I modelli SOM ed E-SOM devono essere dotati di routine di post-processing, in grado di ottenere un clustering di immagini e/o di dati in forma tabellare	*Capitolo 5*
R3	E' necessario dotare il modello di un tool software di visualizzazione dell'output grafico dello strato di Kohonen delle SOM e della griglia dell'E-SOM, a seguito di un esperimento	*Paragrafo 4.3* *Paragrafo 6.3*

Tabella 2 – Tabella riassuntiva dei requisiti generali (obiettivi scientifici)

REQUISITI SUITE DAMEWARE		
ID	REQUISITO	VERIFICA
R4	*I vari modelli dovranno essere integrati nella suite DAMEWARE.* Occorre quindi rispettare i vincoli strutturali e d'interfaccia richiesti dai componenti già attivi. Si prevede di creare un plugin per ogni modello con il tool di wrapping fornito dalla web application.	*Paragrafo 2.1.1*
R5	Per poter essere creato il plugin di ciascun modello, da integrare nella suite DAMEWARE, *l'eseguibile di ogni modello deve essere a linea di comando o mediante uso di file di configurazione esterni*.	*Paragrafo 2.1.1*
R6	*L'utente deve essere correttamente loggato alla suite DAME e avere creato un proprio workspace nel quale siano caricati i file necessari*	*Paragrafo 2.1.2*
R7	*Il plugin clustering-SOM, E-TREE ed E-SOM deve prevedere la scelta utente tra i vari casi d'uso TRAIN, TEST, RUN.*	*Paragrafo 8.1* *Paragrafo 8.3* *Paragrafo 8.4*

Tabella 3 – Tabella riassuntiva dei requisiti relativi alla piattaforma ospite DAMEWARE

REQUISITI PARAMETRI INPUT – MODELLO SOM + Post-Processing		
ID	REQUISITO	VERIFICA
R8	*Il numero di neuroni dello strato di input deve essere > 0*	*Paragrafo 12.1, Pag. 111*
R9	*Il numero di righe della griglia deve essere un intero maggiore di 1*	*Paragrafo 12.1, Pag. 111*
R10	*Il numero di colonne delle griglia deve essere un intero maggiore di 1*	*Paragrafo 12.1, Pag. 111*
R11	*Il numero di iterazioni deve essere un intero maggiore di 0*	*Paragrafo 12.1, Pag. 111*
R12	*Il diametro del vicinato deve essere un intero maggiore di 0*	*Paragrafo 12.1, Pag. 111*
R13	*L' indice di apprendimento iniziale deve essere un numero reale compreso tra 0 e 1*	*Paragrafo 12.1, Pag. 111*
R14	*L'indice di apprendimento finale dev'essere un numero reale compreso tra 0 e l'indice di apprendimento iniziale*	*Paragrafo 12.1, Pag. 111*
R15	*I pesi iniziali della rete devono essere sempre normalizzati tra -1 e 1*	*Paragrafo 12.1, Pag. 112*
R16	*Le features dei pattern input possono essere normalizzati, tra –1 e 1, a discrezione dell'utente.*	*Paragrafo 12.1, Pag. 112*
R17	*In caso di file FITS_IMAGE 3D, è obbligatorio prevedere una griglia di Kohonen 3D. Indipendentemente dalla scelta dell'utente. In tutti gli altri casi, il corretto uso dell'opzione 3D è delegato all'utente*	*Paragrafo 12.5, Pag. 118*
R18	*In caso di opzione 3D immessa dall'utente su dati tabulari con numero di features inferiore a 3, occorre generare un'eccezione con interruzione dell'esecuzione*	*Paragrafo 12.5, Pag. 118*
R19	*In caso di opzione 3D scelta dall'utente per file immagini 2D, occorre generare un'eccezione con interruzione dell'esecuzione*	*Paragrafo 12.5, Pag. 118*
R20	*Il formato del dataset deve essere coerente con l'estensione:* - *Il formato ascii ammette le estensioni: .txt e .dat* - *Il formato table ammette le estensioni: .csv, .votable e .fits* - *Il formato image ammette le estensioni: .jpg, .JPG, .png, e .gif* - *Il formato fits_image ammette le estensioni: .fits*	*Paragrafo 12.1, Pag. 111*
R21	*Il numero di cluster attesi deve essere maggiore di 1*	*Paragrafo 12.1, Pag. 111*
R22	*Il parametro specificato per il post-processing della SOM deve corrispondere ad un metodo effettivamente implementato*	*Paragrafo 12.1, Pag. 111*
R23	*Il numero di neuroni dello strato di input deve coincidere con il numero di features dei pattern*	*Paragrafo 12.1, Pag. 112*
R24	*In caso di FITS image composita, il risultato del clustering deve fornire un unico output di clustering, relativo al dataset composito*	*Paragrafo 8.5.1.2*
R25	*L'utente deve obbligatoriamente specificare un dataset in input nei formati specificati al paragrafo 8.5.1.1*	*Paragrafo 12.1, Pag. 111*
R26	*L'utente, nei casi di Test e Run, deve obbligatoriamente specificare un file di configurazione della rete*	*Paragrafo 12.1, Pag. 111*
R27	*L'utente, in caso di Test, deve obbligatoriamente specificare un file contenente il cluster atteso per ogni pattern contenuto nel dataset*	*Paragrafo 12.1, Pag. 111*
R28	*In caso di Test, ill numero di target deve essere pari al numero di pattern del dataset*	*Paragrafo 12.1, Pag. 112*

Tabella 4 - Tabella riassuntiva dei requisiti relativi ai parametri input del modello SOM + post-processing

REQUISITI PARAMETRI INPUT – MODELLO E-SOM		
ID	REQUISITO	VERIFICA
R29	*Il numero di neuroni dello strato di input deve essere > 0*	*Paragrafo 13.1, Pag. 122*
R30	*Le features dei pattern input possono essere normalizzati, tra –1 e 1, a discrezione dell'utente.*	*Paragrafo 13.1, Pag. 123*
R31	*Il formato del dataset specificato deve essere coerente con l'estensione:* - *Il formato ASCII ammette le estensioni: .txt e .dat* - *Il formato table ammette le estensioni: .csv, .votable e .fits* - *Il formato image ammette le estensioni: .jpg, .JPG, .png, e .gif* - *Il formato fits_image ammette le estensioni: .fits*	*Paragrafo 13.1, Pag. 122*
R32	*Il numero di neuroni dello strato di input deve coincidere con il numero di features dei pattern*	*Paragrafo 13.1, Pag. 123*
R33	*La soglia minima di distanza tra input pattern e nodi prototipo (candidati) deve essere un reale > 0*	*Paragrafo 13.1, Pag. 122*
R34	*In caso di FITS image composita, il risultato del clustering deve fornire un unico output di clustering, relativo al dataset composito*	*Paragrafo 8.5.1.2*
R35	*L'utente deve obbligatoriamente specificare un dataset in input nei formati specificati al paragrafo 8.5.1.1*	*Paragrafo 13.1, Pag. 122*
R36	*L'utente, nei casi di Test e Run, deve obbligatoriamente specificare un file di configurazione della rete*	*Paragrafo 13.1, Pag. 122*
R37	*L'utente, in caso di Test, deve obbligatoriamente specificare un file contenente il cluster atteso per ogni pattern contenuto nel dataset*	*Paragrafo 13.1, Pag. 122*
R38	*In caso di Test, ill numero di target deve essere pari al numero di pattern del dataset*	*Paragrafo 13.1, Pag. 122*

Tabella 5 - Tabella riassuntiva dei requisiti relativi ai parametri input del modello E-SOM

REQUISITI INPUT – MODELLO E-TREE		
ID	REQUISITO	VERIFICA
R39	*Il modello prevede un file di configurazione esterno per i parametri interni*	*Paragrafo 8.6.1*
R40	*Il modello viene eseguito a riga di comando con una serie di opzioni dipendenti dal caso d'uso*	*Paragrafo 8.6*
R41	*Il modello prevede casi d'uso TRAIN, TEST/RUN e FULL*	*Paragrafo 8.4*
R42	*I file tabulari input sono in formato ASCII, con la riga interposta tra header e pattern contenente un solo valore intero, corrispondente al numero di features dei pattern elencati.*	*Paragrafo 8.6.1*
R43	*Il modello deve prevedere il caso TRAIN con funzione di cross validation opzionale*	*Paragrafo 8.4*

Tabella 6 - Tabella riassuntiva dei requisiti relativi ai parametri input del modello E-TREE

Circa i requisiti relativi agli output dei modelli, la lista mostrata in Tabella 7 riporta i tipi di file input/output previsti per i due modelli. Per i dettagli si veda i paragrafi dedicati all'I/O dei modelli.

REQUISITI FILE OUTPUT					
ID	NOME	CASO D'USO	TIPO	MODELLO	VERIFICA
R44	***<Dataset>***	Tutti	Input	SOM E-SOM	*Paragrafo 8.5*
R45	***Model_XXX_network_configuration.txt***	Train Test Run	Input	SOM E-SOM	*Paragrafo 8.5*
R46	***Model_Test_target.txt***	Test	Input	SOM E-SOM	*Paragrafo 8.5*
R47	***<File dei parametri>***	Train	Input	E-TREE	*Paragrafo 8.6*
R48	***outNewickTree.txt***	Train	Output	E-TREE	*Paragrafo 8.6*
R49	***tempNewickTree.txt***	Train	Output	E-TREE	*Paragrafo 8.6*
R50	***testStatOut.txt***	Train	Output	E-TREE	*Paragrafo 8.6*
R51	***<Albero>.ebd***	Train Test	Output Input	E-TREE	*Paragrafo 8.6*
R52	***trainLog.txt***	Train	Output	E-TREE	*Paragrafo 8.6*
R53	***testLog.txt***	Test	Output	E-TREE	*Paragrafo 8.6*
R54	***testOut.txt***	Test	Output	E-TREE	*Paragrafo 8.6*
R55	***Model_XXX_network_configuration.txt***	Train	Output	SOM E-SOM	*Paragrafo 8.5*
R56	***Model_XXX_status.log***	Tutti	Output	SOM E-SOM	*Paragrafo 8.5*
R57	***Model_XXX_params.xml***	Tutti	Output	SOM E-SOM	*Paragrafo 8.5*
R58	***Model_XXX_results.txt***	Tutti	Output	SOM E-SOM	*Paragrafo 8.5*
R59	***Model_XXX_histogram.png***	Tutti	Output	SOM E-SOM	*Paragrafo 8.5*
R60	***Model_XXX_clusters.txt***	Tutti	Output	SOM E-SOM	*Paragrafo 8.5*
R61	***Model_XXX_U_Matrix.png***	Tutti	Output	SOM E-SOM	*Paragrafo 8.5*
R62	***Model_XXX_normalized_clusters.txt***	Tutti	Output	SOM E-SOM	*Paragrafo 8.5*
R63	***Model_XXX_Output_layer.txt***	Tutti	Output	SOM E-SOM	*Paragrafo 8.5*
R64	***Model_XXX_clustered_image.png***	Tutti	Output	SOM E-SOM	*Paragrafo 8.5*
R65	***Model_XXX_clustered_image.txt***	Tutti	Output	SOM E-SOM	*Paragrafo 8.5*
R66	***Model_error.log***	Tutti	Output	SOM E-SOM	*Paragrafo 8.5*
R67	***Model_XXX_clustered_datacube_image.zip***	Tutti	Output	SOM E-SOM	*Paragrafo 8.5*

Tabella 7 – Tabella riassuntiva dei requisiti relativi all'input/output dei modelli

4 IL MODELLO SOM

Il più conosciuto ed applicato modello di rete neurale auto-organizzante prende il nome dal suo inventore (Kohonen, 2001) ed è costituito da una rete a due strati, dei quali uno di input e l'altro di output. I neuroni dei due strati sono completamente connessi tra loro, mentre i neuroni dello strato di output sono connessi, ciascuno, con un "vicinato" di neuroni secondo un sistema di inibizione laterale definito a "cappello messicano" (Mexican Hat). I pesi dei collegamenti intra-strato dello strato di output, o strato di Kohonen come è comunemente chiamato, non sono soggetti ad apprendimento, ma sono fissi e positivi nella periferia adiacente ad ogni neurone. Le figure successive rappresentano una rete di Kohonen e il collegamento a cappello messicano.

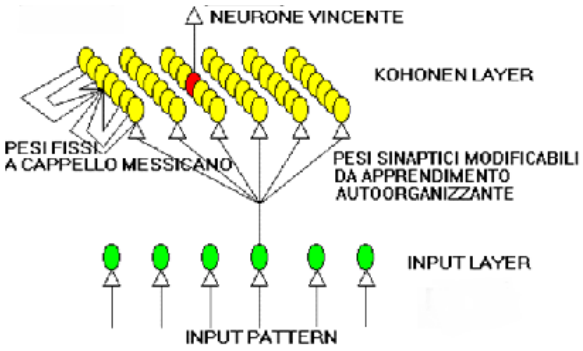


Figura 6 – Architettura di una SOM

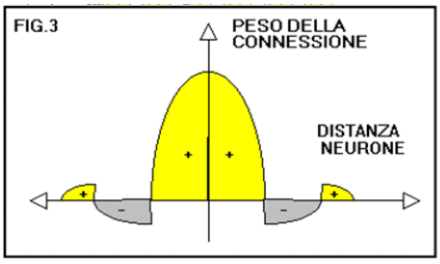


Figura 7 – Legge di attivazione di un neurone SOM basata sulla funzione "Mexican Hat"

Nello strato di output un solo neurone deve risultare "vincente" per ogni input pattern che viene fornito alla rete. Tale neurone identifica una classe a cui l'input appartiene. Il collegamento a *cappello messicano*, nella versione originale di questo tipo di rete, tende a favorire il formarsi di "bolle di attivazione" che identificano input simili. Ogni neurone dello strato di Kohonen riceve uno stimolo che è pari alla sommatoria degli input moltiplicati per il rispettivo peso sinaptico:

$$\boldsymbol{A(j) = S(k)w(j,k) * x(k)} \qquad \textbf{(1)}$$

Tra tutti i neuroni di output viene scelto quello con valore di attivazione maggiore che assume quindi il valore 1, mentre tutti gli altri assumono il valore 0 secondo la tecnica "WTA" (Winner Takes All). Lo scopo di una rete di Kohonen è quello di avere, per input simili, neuroni vincenti vicini, così che ogni bolla di attivazione rappresenti una classe di input aventi caratteristiche similari. Il comportamento sopra descritto si raggiunge dopo la presentazione di molti input per un certo numero di volte alla rete, modificando, per ogni presentazione di input, solo i pesi che collegano il neurone vincente (appartenente allo strato di Kohonen) con i neuroni dello strato di input.

Ciò secondo la seguente formula:

$$\boldsymbol{W(k,j)_{new} = W(k,j)_{old} + \eta * (X(k) - W(k,j)_{old})} \quad \textbf{(2)}$$

dove:

$W(k,j) = peso\ sinaptico\ del\ collegamento\ tra\ input\ k\ e\ neurone\ vincente;$
$\mathrm{X(k)} = input\ k_{esimo}\ dell'input\ pattern;$
$\eta\ = tasso\ di\ apprendimento\ (learning\ rate), nel\ range\ ]0,\ 1[\ .$

In pratica i pesi da aggiornare per ogni presentazione di input sono tanti quanti sono i neuroni di input. In taluni casi si aggiornano con lo stesso criterio, magari usando valori di η decrescenti (learning decay), anche i pesi dei neuroni di un opportuno vicinato (dimensionalmente sovrapponibile alla parte positiva del cappello messicano). Leggendo con attenzione la formula (2), ci si accorge che il suo ruolo è quello di ruotare il vettore dei pesi sinaptici verso il vettore di input, nello spazio dei parametri. In questo modo il neurone vincente viene ancor più sensibilizzato al riconoscimento della forma dell'input presentato (Figura 8).

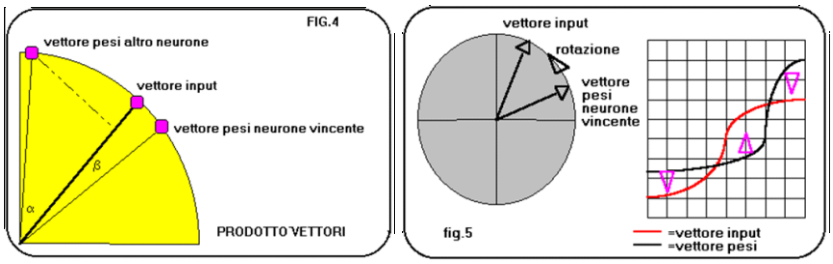


Figura 8 – Schema di apprendimento con direzione dei vettori dei pesi

Quando viene impiegata una rete auto-organizzante, la matrice di neuroni di output si presenta con un insieme di bolle di attivazione che corrispondono alle classi in cui la rete ha suddiviso gli input per similitudine. Se addestriamo una rete con dati di cui conosciamo a priori l'appartenenza a specifiche classi, potremo associare ciascuna delle bolle di attivazione a ognuna di queste classi (Figura 9).

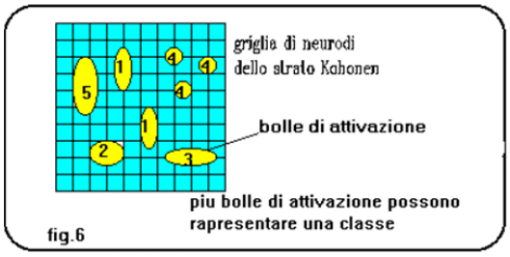


Figura 9 – Clusters in output sullo strato di Kohonen dopo la fase di apprendimento

Il vero fascino dei paradigmi non supervisionati è, però, quello di estrarre informazioni di similitudine tra input pattern (esempio un segnale sonar o radar) molto complessi, lavorando su grandi quantità di dati (MDS) misurati nel mondo reale. Questo tipo di rete evidenzia molto bene il fatto che le reti neurali in genere sono processi di tipo "*data intensive*" (con molti più dati che calcoli), anziché "*number crunching*" (con molti calcoli su pochi dati).

Come si può vedere in Figura 9, più bolle di attivazione possono rappresentare una classe di input: questo può essere dovuto all'eterogeneità di una classe (che deve essere ovviamente conosciuta) o anche alla sua estensione nello spazio delle forme possibili (in questo caso due pattern agli estremi della classe possono dare origine a bolle di attivazione separate). I dati che vengono forniti in input alla rete di Kohonen possono essere normalizzati in un intervallo (di solito [0,1] o, come nel nostro caso, [-1, 1]), nel qual caso quindi è necessario effettuare un pre-processing che preveda la normalizzazione dei dati in ingresso. La normalizzazione o comunque il pre-processing dei dati può essere effettuato in modi differenti a seconda del tipo di problema.

In alcuni casi è molto importante riconoscere la forma dell'input pattern (es: riconoscimento segnali/immagini), mentre in altri casi è importante anche mantenere intatta una relazione dimensionale tra i pattern di input (es: distinguere i punti definiti da coordinate x/y all' interno di certe aree). Viceversa, per i pesi iniziali della rete, si prevede sempre la loro normalizzazione tra -1 e 1.

4.1 ALGORITMO DI TRAINING

Nella fase di training i neuroni più vicini ai dati, detti vincitori o BMU (Best Matching Unit), sono attivati e quindi avvicinati ai dati in modo da simulare l'apprendimento. Decretati i vincitori e cercando di simulare il comportamento in Figura 10, anche i neuroni posti in vicinanza ad essi apprendono la disposizione dei dati, ma in quantità ridotta, secondo la tecnica "WTM" (Winner Takes Most), meno rigida rispetto alla WTA precedentemente descritta.

Di seguito è riportato l'algoritmo che è alla base del modello, considerando una griglia di output di *l* neuroni e *d* neuroni di input:

1. $w_{ij}^{(0)} =$ valori di inizializzazione casuali, tra -1 e 1, ordinati sulla griglia, dove $i = 1 \dots l, j = 1 \dots d$;
2. Seleziona pattern di neuroni di input $x^n = (x_1^n, x_2^n, \dots, x_d^n)$;
3. $D = (D_1, \dots, D_l)$ vettore delle distanze tra il pattern in input ed i pesi dei neuroni dove $D_i = dist(x^n, w_i^{(t)})$;
4. Seleziona il BMU index tale che $\mathrm{D_{BMUindex}} = \min(\mathrm{D})$;
5. Aggiorna i pesi nel modo seguente: $w_{ij}^{(t+1)} = w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$ per $i = 1 \dots l$, dove $\Delta w_{ij}^{(t)}$ è data dall'eq. (2);
6. Torna al passo 2 finché non è raggiunto un criterio di arresto.

Nell'algoritmo, al punto 3, è stata utilizzata la seguente funzione per il calcolo delle distanze tra il pattern in input ed il vettore dei pesi:

$$\boldsymbol{D_i = dist(x^n, w_i^{(t)})} \quad \textbf{(3)}$$

In questo modo si ottiene un vettore di grandezza *l*, ovvero la dimensione del vettore dei pesi. Per il calcolo delle distanze è possibile utilizzare diverse funzioni, che non si distacchino troppo però da quella euclidea. La distanza euclidea garantisce infatti che l'aggiornamento dei pesi (al punto 5) avvenga in maniera tale da seguire il gradiente della funzione di distanza stessa, ovvero la differenza aritmetica. Nel caso della distanza euclidea otteniamo:

$$\boldsymbol{dist(x^n, w_j^t) = \sum_{j=1}^{d} (x_j^n - w_{ij}^{(t)})^2} \quad \textbf{(4)}$$

4.1.1 Aggiornamento dei pesi

Poniamo ora l'attenzione sull'aggiornamento dei pesi dei neuroni. Nell'algoritmo di apprendimento appena visto è stata utilizzata l'equazione (2) per il calcolo di $\Delta w_{ij}^{(t)}$. Tale equazione può essere riformulata come segue (quantità di variazione associata ai pesi dei neuroni al tempo t):

$$\boldsymbol{\Delta w_{ij}^{(t)} = \eta_t \Lambda^{(t)}(i, \mathrm{BMUindex})(x_j^n - w_{ij}^t)} \quad \textbf{(5)}$$

Vediamo ora il significato della formula (5).

4.1.1.1 Tasso di apprendimento (Learning Rate)

Il parametro η_t è il fattore di rapidità dell'algoritmo. In genere questo valore può essere costante in t, oppure assumere valori che decrescono progressivamente. In generale valori elevati permettono un apprendimento elevato ma al tempo stesso rischioso a causa dell'elevata dimensione che caratterizza il passo dell'apprendimento.

Nel caso di valori non costanti esistono le seguenti varianti:

- **Lineare**: decresce nel tempo in modo lineare;
- **Esponenziale**: $\eta_t = \eta_0 \exp(-\frac{t}{\tau})$, decresce nel tempo, basandosi anche su τ, un parametro che non varia durante l'esecuzione dell'algoritmo, in genere dipendente dal massimo numero di iterazioni che si vuole eseguire;

4.1.1.2 Funzione di vicinato

Il parametro $\Lambda^{(t)}$ è la funzione di vicinato. Dati due indici dei pesi dei neuroni i_1 e i_2 la funzione restituisce il fattore vicinanza $0 \leq \Lambda^{(t)}(i_1, i_2) \leq 1$. Il suo compito è quello di dosare la diffusione dell'apprendimento ai neuroni vicini del BMU, simulando il comportamento già descritto. Il calcolo delle distanze di vicinato è legato alla topologia della rete. Infatti, due neuroni non interconnessi tra loro, ma spazialmente vicini, non lo saranno considerati dal risultato di questa funzione. Il modello non specifica rigide direttive per la sua realizzazione e le soluzioni sono dunque molteplici, a seconda del tipo di aggiornamento del vicinato richiesto. Le varianti più note sono:

- **Bubble**: la funzione restituisce 1 o 0 a seconda se ci si trovi entro il raggio del BMU:

$$\boldsymbol{\Lambda^{(t)}(i_1, i_2) = \begin{cases} 1 & dgrid(i_1, i_2) < r \\ 0 & else \end{cases}} \quad \textbf{(6)}$$

  con $dgrid(i_1, i_2)$ = distanza calcolata in base alle coordinate dei nodi sulla griglia;

- **Gaussian**: la funzione restituisce valori compresi tra 0 e 1 che si distribuiscono intorno al BMU come una comune gaussiana. Nel caso venga usata questa tipologia di soluzione, la funzione necessita di ulteriori parametri per la modellazione della forma della gaussiana. In tal caso potremmo calcolare:

$$\boldsymbol{\Lambda^{(t)}(i_1, i_2) = \exp\left(-\frac{dgrid(i_1, i_2)}{2\sigma_t^2}\right)} \quad \textbf{(7)}$$

  con $\sigma_t = \sigma_0 \exp(-\frac{t}{\tau})$, ovvero, quest'ultima, una funzione che decresce nel tempo con andamento esponenziale. Come τ, anche σ_0 è un parametro costante pre-calcolato;

- **Cutted-Gaussian**: un mix fra i due precedenti che, entro un certo raggio, restituisce un valore gaussiano; ma al di fuori del raggio restituisce valore zero.

Nell'implementazione attuale, la funzione di vicinato è la Gaussian ed il learning rate decresce in maniera esponenziale.

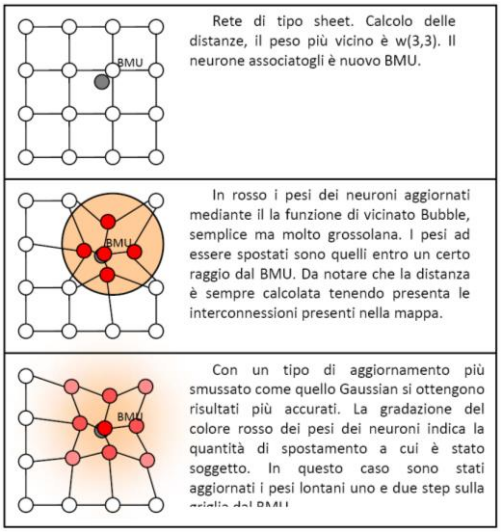


Figura 10 – Esempi del meccanismo di apprendimento

In Figura 10 è riportato un esempio di rappresentazione della fase di apprendimento. La rete SOM può essere infatti rappresentata mediante il tracciamento sugli assi cartesiani del valore dei pesi dei neuroni di output. Il cerchio in grigio è il pattern di input selezionato, mentre i cerchi in bianco sono i nodi della rete.

4.1.2 Versione con Apprendimento *Batch*

In un processo di regressione (approssimazione di una funzione) o clustering, l'algoritmo discusso in precedenza può essere adattato in modo da rendere il flusso di apprendimento in modalità batch, ossia presentando l'insieme dei pattern input in blocco ad ogni iterazione, applicando poi la formula (5) di apprendimento (modifica dei pesi nello strato di Kohonen) dopo la presentazione dell'intero blocco di pattern input.

4.1.3 Versione con Apprendimento *Incremental*

Nella versione Incremental, l'aggiornamento dei pesi della rete avviene dopo ogni presentazione di un pattern input singolo. **E' questa la modalità realmente implementata nell'attuale versione dell'algoritmo SOM.**

4.2 DIAGRAMMA DI FLUSSO GENERALE DEL MODELLO SOM

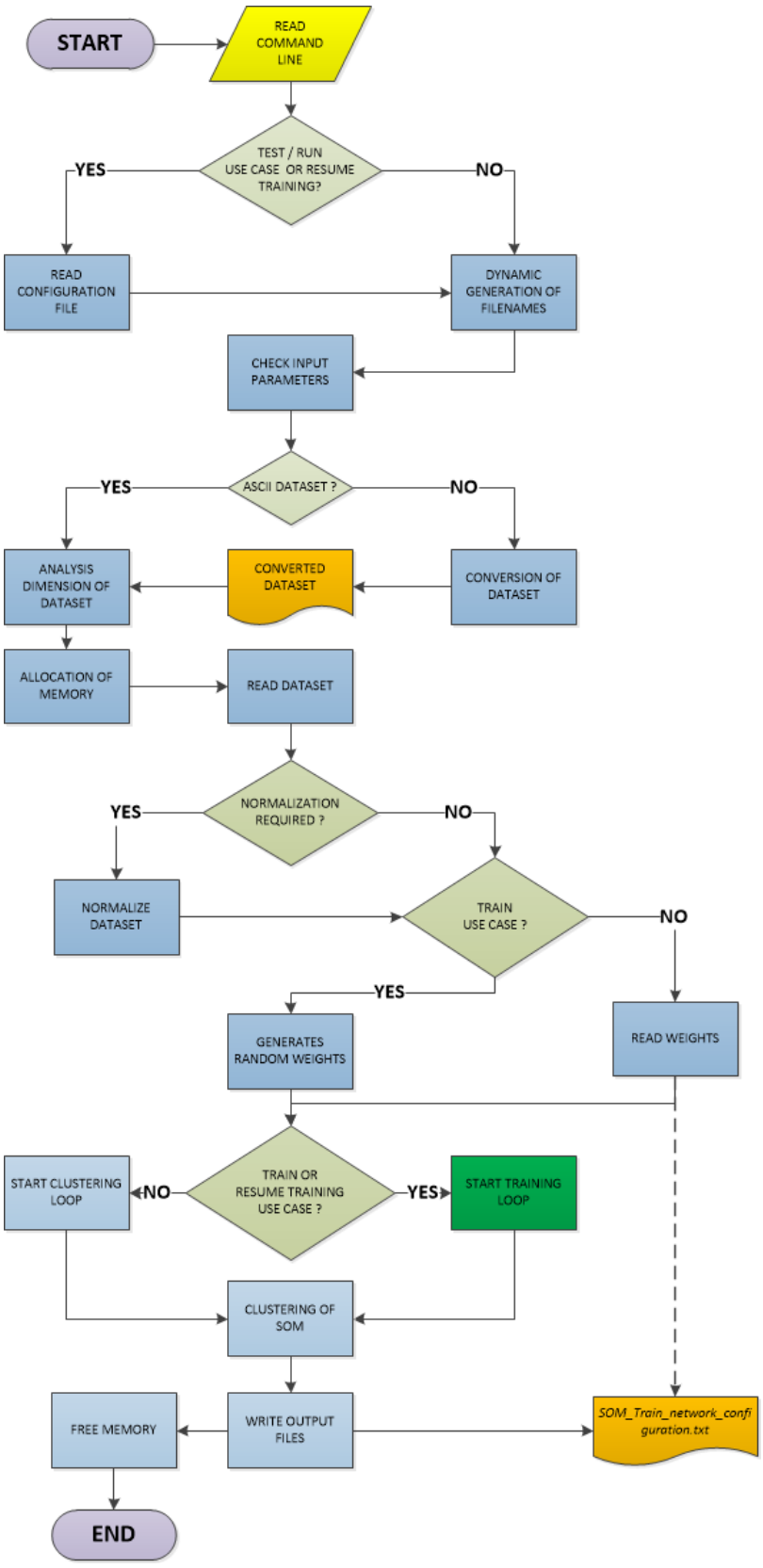


Figura 11 - Flow Chart generale dell'algoritmo SOM. In verde scuro il blocco relativo alla procedura di training

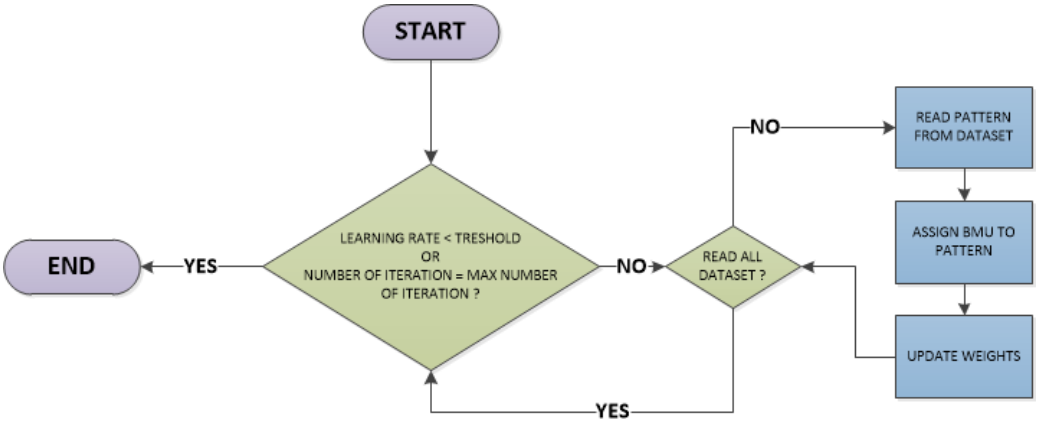


Figura 12 - Flow Chart del loop di training del modello SOM

4.3 VISUALIZZAZIONE DELLA MAPPA DI KOHONEN: U-MATRIX

La **U-Matrix (Unified Distance Matrix)** è lo standard per la valutazione e l'interpretazione di una SOM. Durante l'addestramento della rete i pesi dei neuroni sono calcolati in maniera tale che elementi vicini tra loro sulla mappa siano anche vicini nello spazio dei pesi. In questo modo lo strato di Kohonen è in grado di rappresentare, rispettando la topologia, dati multi-dimensionali su una mappa di due o tre dimensioni.

Siano

n, $neurone\ della\ mappa$
$NN(n)$, $insieme\ dei\ nodi\ adiacenti\ ad\ n\ sulla\ mappa$
$w(n)$, $vettore\ dei\ pesi\ associato\ al\ neurone\ n$
$\|w(n), w(m)\|$, $distanza\ euclidea\ tra\ i\ vettori\ dei\ pesi\ dei\ neuroni\ n\ ed\ m$
$U_{height(n)}$, $valore\ (peso)\ associato\ al\ nodo\ n$

Ad ogni nodo dello strato di Kohonen verrà assegnato un valore secondo la seguente formula:

$$\boldsymbol{U_{height(n)} = \sum_{m \in NN(n)} \|w(n), w(m)\|} \quad \textbf{(8)}$$

Il valore così calcolato diventa identificativo della distanza di un nodo da tutti i suoi vicini più prossimi ed è visualizzabile utilizzando una heat map in cui colori chiari rappresentano neuroni vicini tra loro nello spazio dei pesi, viceversa colori scuri rappresentano neuroni lontani tra loro (Moutarde & Ultsch 2005). Tipicamente si usa rappresentare la mappa con una scala di grigi, come mostrato in Figura 13. Per aumentare ulteriormente il grado di interpretabilità della U-Matrix è possibile sovrapporre ad ogni nodo BMU di qualche pattern un colore che ne identifichi il cluster di appartenenza. **Quest'ultima è la versione implementata all'interno del software.** Ovviamente, i nodi, su cui non è sovrapposto alcun quadratino colorato, non sono mai risultati BMU di qualche pattern input.

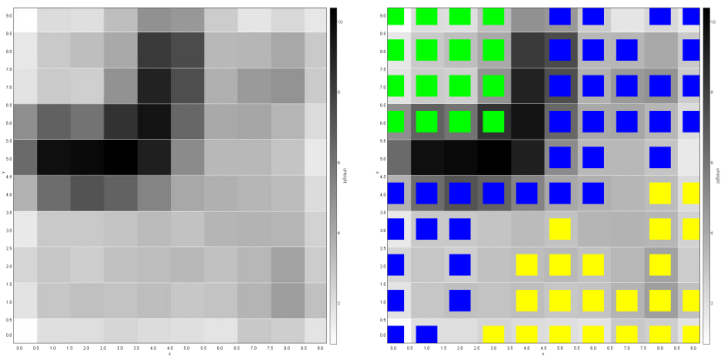


Figura 13 – Esempi di U-Matrix

4.4 INDICI DI QUALITÀ

4.4.1 Indici di qualità di una SOM

Per valutare la qualità di una mappa ottenuta come primo stadio di un processo di clustering, ci si può basare sui criteri suggeriti da Kiviluoto (1996):

i. Qual è il grado di continuità della mappa topologica?
ii. Qual è la risoluzione della mappa?
iii. La topologia della mappa riflette la probabilità di distribuzione dello spazio dei dati?

Per quanto concerne il terzo punto, in letteratura sono presenti diversi esempi di SOM con struttura incrementale in grado di preservare la topologia dello spazio dei dati (vedi E-SOM).

La quantificazione delle prime due proprietà può essere invece ottenuta tramite il calcolo dell'errore di quantizzazione e dell'errore topografico. (Chi & Yang 2008)

4.4.1.1 Errore di quantizzazione

L'errore di quantizzazione viene utilizzato per calcolare la similarità tra i pattern assegnati al medesimo BMU. Di seguito è mostrata la formula per il calcolo del suddetto errore.

$$\boldsymbol{QE} = \frac{\mathbf{1}}{\boldsymbol{N}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}}\|\overrightarrow{\boldsymbol{w_{BMUi}}} - \overrightarrow{\boldsymbol{x_i}}\| \qquad \textbf{(9)}$$

dove

$\overrightarrow{w_{BMUi}} = vettore\ dei\ pesi\ dell'\ i_{esimo}\ \ BMU$
$N = numero\ di\ pattern\ che\ compongono\ il\ dataset$
$\overrightarrow{x_i} = i_{esimo}\ vettore\ in\ input\ rappresentato\ dal\ BMU\ considerato$

La formula (9) corrisponde alla media delle distanze di ogni pattern dal vettore dei pesi del BMU che lo identifica.

4.4.1.2 Errore topografico

L'errore topografico viene utilizzato per calcolare la dissimilarità tra i pattern assegnati a BMU differenti. Anche per questo errore di seguito viene fornita la formula.

$$\boldsymbol{TE} = \frac{\mathbf{1}}{\boldsymbol{N}}\sum_{\boldsymbol{i}}^{\boldsymbol{N}}\boldsymbol{u}(\overrightarrow{\boldsymbol{w_i}}) \qquad \textbf{(10)}$$

dove

$N = numero\ di\ pattern\ che\ compongono\ il\ dataset$

$$u(\overrightarrow{x_i}) = \begin{cases} 1, & se\ il\ primo\ e\ il\ secondo\ BMU\ dell'i_{esimo}\ pattern\ non\ sono\ adiacenti \\ 0, & altrimenti \end{cases}$$

La formula (10) corrisponde quindi alla media del numero di volte in cui, per uno stesso pattern, primo e secondo BMU non sono adiacenti sulla griglia dello strato di Kohonen. Per primo e secondo BMU si intende in termini di vicinato (vettore dei pesi) rispetto al pattern di riferimento nella mappa di Kohonen.

5 SOM POST-PROCESSING: CLUSTERING

Tra le metodologie di clustering più comuni, il modello SOM di Kohonen rimane un campo relativamente inesplorato. Probabilmente l'inadeguata attenzione all'applicazione delle reti SOM in tal senso è dovuta alla mancanza di procedure per generare cluster a partire direttamente dall'output dello strato di Kohonen. Per sua natura infatti il modello SOM non è in grado di produrre un corretto raggruppamento dei nodi della mappa, limitandosi quasi sempre a produrre una corrispondenza 1:1 tra nodi e clusters, come mostrato in Figura 14.

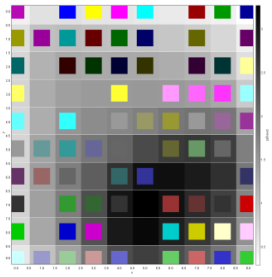


Figura 14 - U-Matrix in output del modello SOM. Quasi tutti i nodi individuano erroneamente un diverso cluster

Il modello SOM tuttavia è in grado di convertire dati complessi in forma semplice e graficamente intuitiva, riducendone l'elevata dimensionalità, configurandosi quindi come primo stadio di una tecnica di clustering a due stadi (**Two-stage clustering).**

5.1 CLUSTERING A DUE STADI

Lo scopo dell'attuazione di un metodo di clustering a due stadi è quello di superare i principali problemi di cui sono afflitti i metodi convenzionali, come la sensibilità ai prototipi iniziali (proto-cluster) e la difficoltà di determinare un appropriato numero di cluster attesi.

L'approccio maggiormente utilizzato è la combinazione di una tecnica di clustering gerarchico o una SOM, seguita da una tecnica di clustering partizionale. La funzione della SOM al primo stadio è quella di identificare un numero iniziale di cluster e relativi centroidi, superando, di fatto, i problemi precedentemente descritti. Nel secondo stadio invece, una tecnica di clustering partizionale assegnerà in maniera definitiva un cluster per ogni pattern (Chi & Yang 2008).

In alternativa è possibile utilizzare la SOM per mappare i dati in input sullo strato di Kohonen, i cui nodi verranno utilizzati per la fase di clustering successiva che può essere di nuovo una SOM, un metodo gerarchico o partizionale. Il vantaggio principale del secondo approccio risiede nel fatto che il secondo stadio viene applicato a partire dai nodi (proto-cluster) dello strato di Kohonen, che generalmente sono in numero inferiore ai pattern input. Tale tecnica si traduce quindi in un notevole vantaggio dal punto di vista computazionale. In tal caso però potrebbe essere necessaria la scelta di un numero di cluster attesi (clustering partizionale).

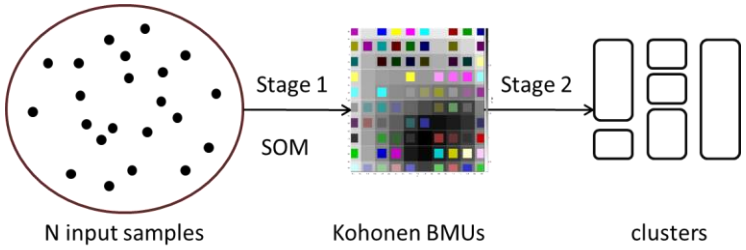


Figura 15 – Clustering a doppio stadio, basato sul pre-clustering con SOM seguita da un generico metodo di post-processing

Da quanto detto risulta inoltre evidente una ben nota dicotomia di approccio che distingue clustering di tipo gerarchico e clustering di tipo partizionale. Da notare che in entrambi i casi, occorre sempre distinguere tra le diverse tipologie di metrica utilizzata internamente. Il paragrafo 5.2 ne descrive le tipologie più comuni e relativi aspetti prestazionali.

5.1.1 Clustering partizionale

Appartengono a questa categoria tutti quegli algoritmi che partizionano i dati, operando nel seguente modo:

1. *Calcolo del numero di cluster;*
2. *Inizializzazione dei centri dei cluster;*
3. *Partizionamento dei dati in base ai centri calcolati ed alle relative distanze;*
4. *Aggiornamento dei centri dei cluster;*
5. *Se il partizionamento è cambiato e non è stato raggiunto un criterio d'arresto, torna al punto 3.*

Risulta evidente che in questo tipo di approccio il numero di cluster deve essere specificato a priori o selezionato dinamicamente cercando di minimizzare una qualche misura di validità. Una soluzione ampiamente adottata è per esempio quella di minimizzare la distanza intra-cluster e contemporaneamente massimizzare quella inter-cluster.

Tipicamente la complessità degli algoritmi appartenenti a tale categoria è O(N), dove N è il numero di pattern input.

5.1.2 Clustering gerarchico

Il clustering gerarchico consiste in una sequenza di partizionamenti organizzati in una struttura di tipo gerarchico nota come dendogramma. Questo tipo di clustering ha il vantaggio di non dover specificare a priori il numero di cluster, tuttavia la complessità è maggiore: tipicamente $O(N^2)$.

A seconda dell'approccio seguito gli algoritmi di clustering gerarchico possono essere classificati in:

- Algoritmi agglomerativi: seguono una strategia bottom-up, operando come segue:
    1. *Assegnazione di ogni pattern al proprio cluster;*
    2. *Calcolo delle distanze tra tutti i cluster;*
    3. *Unione dei due cluster più vicini tra loro;*
    4. *Torna al passo 2 finché non è rimasto un solo cluster.*

- Algoritmi divisivi: seguono una strategia top-down che opera analogamente a quanto descritto precedentemente, sebbene si parta da un unico cluster contenente tutti i punti, per poi dividerli finché ogni cluster non sia composto dal minimo numero di punti, come mostrato in Figura 16.

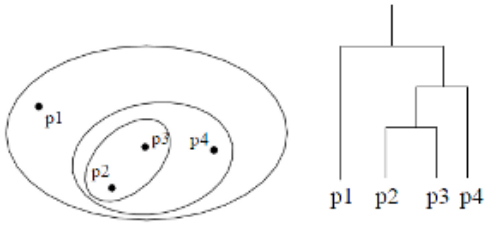


Figura 16 - Esempio di clustering gerarchico top-down

Gli algoritmi di clustering gerarchico non forniscono un clustering unico. Infatti, a seconda di dove venga "tagliato" il dendogramma, verrà prodotto un clustering differente. Anche in questo caso quindi, entra in gioco un criterio di validità da minimizzare durante il procedimento. Un esempio in tal senso è il metodo proposto da Kiang (2001), che, prendendo spunto da Murthag (1985), propone di minimizzare la varianza complessiva dei cluster in ogni fase del processo.

5.2 METRICHE PER IL CLUSTERING

Gran parte delle differenze tra algoritmi di tipo gerarchico e partizionale sono state già messe in evidenza durante la descrizione fornita nei paragrafi precedenti. In beve, gli algoritmi di tipo partizionale sono computazionalmente meno costosi e non sono influenzati da una precedente configurazione dei cluster. Tuttavia hanno lo svantaggio di effettuare implicite assunzioni sul numero di cluster attesi, quasi sempre arbitrario nei casi reali.

Indipendentemente dal tipo di metodo scelto, è importante notare il ruolo fondamentale che riveste il calcolo della distanza, sia essa utilizzata come criterio di aggregazione di cluster o come misura di validità.

Siano dati:

$$\begin{cases} C_p, C_t & generici\ cluster \\ \|\cdot\| & norma\ Euclidea \\ d_m(k) & distanza\ intra-cluster\ basata\ su\ un\ criterio\ m \\ D_m(p,t) & distanza\ inter-cluster\ basata\ su\ un\ criterio\ m \\ N_p, N_t & numero\ di\ pattern\ del\ relativo\ cluster \\ CE_k = {}^{1}/_{N_k} \sum_{x_i \in C_k} x_i, & centroide\ del\ cluster\ k \end{cases} \quad \textbf{(11)}$$

è possibile definire le seguenti distanze:

$$\begin{cases} d_{ce}(k) = \frac{\sum_i \|x_i - CE_k\|}{N_k}, & distanza\ intra-cluster\ basata\ sui\ centroidi \\ D_{ce}(p,t) = \|CE_p - CE_t\|, & distanza\ inter-cluster\ basata\ sui\ centroidi \end{cases} \quad \textbf{(12)}$$

$$\begin{cases} d_{av}(k) = \frac{\sum_{i,j\ (i \neq j)} \|x_i - x_j\|}{N_k(N_k-1)}, & distanza\ intra-cluster\ basata\ sulla\ media \\ D_{av}(p,t) = \frac{\sum_{i,j\ (i \neq j)} \|x_i - x_j\|}{N_p N_t}, & distanza\ inter-cluster\ basata\ sulla\ media \end{cases} \quad \textbf{(13)}$$

$$\begin{cases} d_{nn}(k) = \frac{\sum_i \min_j \{\|x_i - x_j\|\}}{N_k}, & distanza\ intra-cluster\ basata\ sul\ vicino\ più\ prossimo \\ D_{sl}(p,t) = \min_{i,j} \{\|x_i - x_j\|\}, & distanza\ inter-cluster\ basata\ sul\ criterio\ single-linkage \\ D_{cl}(p,t) = \max_{i,j} \{\|x_i - x_j\|\}, & distanza\ inter-cluster\ basata\ sul\ criterio\ complete-linkage \end{cases} \quad \textbf{(14)}$$

La selezione di uno di questo metodi dipende dal campo di applicazione. Molti di questi infatti risultano inadeguati in talune situazioni. Le distanze basate sul vicino più prossimo e sul criterio del single-linkage (o minima distanza euclidea tra due cluster, basata sui due oggetti dei rispettivi cluster più vicini tra loro), mostrate nell' equazione (14), sono eccessivamente sensibili al rumore ed agli outliers (Vesanto & Alhoniemi 2000). Anche un solo punto è infatti in grado di influenzare notevolmente il calcolo. Le distanze dell'equazione (13), si basano invece sulla media, che è computazionalmente costosa da calcolare su dataset di grandi dimensioni. La distanza basata sul criterio del complete-linkage (o massima distanza euclidea tra due cluster, basata sui due oggetti dei rispettivi cluster più lontani tra loro) dell'equazione (14), sebbene utilizzata i molti algoritmi di tipo agglomerativo, richiede, per agire in maniera ottimale, cluster molto compatti e ben separati, condizione che raramente è riscontrabile nei casi reali (Vesanto & Alhoniemi 2000).

Alla luce di questa breve disamina si evince che le distanze basate sui centroidi (eq. 12) sembrano essere una delle migliori soluzioni. Tale tipo di distanza viene adoperata anche nel calcolo dell'indice di Davies-Bouldin, utilizzato come indice di qualità di un clustering e descritto nel paragrafo successivo.

5.3 INDICI DI QUALITÀ DEL CLUSTERING

I risultati di un clustering possono essere valutati tramite l'utilizzo dell'indice statistico di Davies-Bouldin (DB), nonché di indici di valutazione facilmente derivabili avendo una base di conoscenza a priori sul dataset, che abbiamo definito rispettivamente Indice di Accuratezza (ICA) e Completezza (ICC) di clustering.

5.3.1 Indice di Davies-Bouldin

Tale indice misura il rapporto tra la distribuzione intra-cluster e le distanze inter-cluster, misurate a partire dai centroidi (Davies & Bouldin, 1979).

La distribuzione interna del cluster C_i può essere espresso come segue:

$$\boldsymbol{S_{i,q} = \frac{1}{|C_i|}\sum_{\vec{x_i}\in C_i}\{|\vec{x_i} - \vec{z}|^q\}^{1/q}} \ , \boldsymbol{i = 1 \dots K} \qquad \textbf{(15)}$$

dove $|C_i|$ denota il numero di vettori in input assegnati al suddetto cluster; x_i e z rappresentano rispettivamente un vettore di input del i_{esimo} cluster e il centroide del cluster stesso; q è un valore assoluto; K rappresenta il numero totale di cluster.

Successivamente la distanza tra due cluster C_i e C_j può essere scritta come:

$$\boldsymbol{d_{ij,t} = \left|\vec{z_i} - \vec{z_j}\right|_t = \left\{\sum_{s=1}^{D}\left|z_{si} - z_{sj}\right|^t\right\}^{1/t}} \qquad \textbf{(16)}$$

dove z_i e z_j rappresentano rispettivamente i centroide del i_{esimo} e j_{esimo} cluster; z_{si} e z_{sj} denotano il valore assoluto della differenza tra I vettori z_i e z_j calcolata sulla dimensione s; D è il numero di dimensioni dei vettori in input; t è un valore assoluto.

L'indice di Davies-Bouldin (DB) può quindi essere espresso dalla seguente equazione:

$$\boldsymbol{DB = \frac{1}{k}\sum_{i=1}^{K}\max_{j,j\neq i}\left\{\frac{S_{i,q}+S_{j,q}}{d_{ij,t}}\right\}} \qquad \textbf{(17)}$$

Dalla formula risulta evidente che a valori bassi del suddetto indice corrisponde un clustering migliore.

L'indice DB perde efficacia ed obiettività nel caso in cui i dati siano distribuiti in modo non lineare. In tal caso i centroidi possono essere individuati per oggetti con distanza intra-cluster molto elevata, ottenendo un valore dell'indice DB erroneamente grande, sebbene il risultato reale del clustering sia ottimale (si veda gli esempi di testing relativi ai dataset Chainlink e Target).

5.3.2 Indici di Accuratezza e Completezza

La valutazione il più possibile oggettiva del risultato di un esperimento di clustering può essere eseguita nel caso in cui si abbia la conoscenza dei cluster cui i dati dovrebbero appartenere. Inoltre, tali indici devono permettere di valutare il clustering in presenza di *outliers*, cioè di oggetti peculiari nello spazio dei parametri, così come in dati distribuiti in modo non lineare (vedi test su dataset Chainlink), per i quali gruppi di dati distanti, ma appartenenti alla stessa categoria, possono essere erroneamente attribuiti a categorie distinte (cluster disgiunti).

Per definire tali indicatori si assume di avere a disposizione le seguenti quantità:

$NC_t = numero\ di\ cluster\ teorici$

$NC_c = numero\ di\ cluster\ calcolati$

$NC_d = numero\ di\ cluster\ disgiunti$

In particolare, definiremo due cluster teorici disgiunti se l'intersezione delle label attribuite dal clustering (cioè calcolate) nei due cluster risulta l'insieme vuoto.

Il primo indicatore è chiamato Indice di Accuratezza di Clustering (ICA o *Index of Clustering Accuracy*), e si definisce nel modo seguente:

$$\boldsymbol{ICA} = \frac{|\boldsymbol{NC_c} - \boldsymbol{NC_t}|}{\boldsymbol{NC_c} + \boldsymbol{NC_t}} \quad \textbf{(18)}$$

Il secondo è chiamato Indice di Completezza di Clustering (ICC o *Index of Clustering Completeness*), e si definisce come segue:

$$\boldsymbol{ICC} = \mathbf{1} - \frac{\boldsymbol{NC_d}}{\boldsymbol{NC_t}} \quad \textbf{(19)}$$

Naturalmente, per costruzione, il calcolo di tali indici presuppone che l'esito del training di un modello su un dataset possa essere confrontato con una base di conoscenza a priori sul numero e attribuzione dei cluster teorici per il dataset. Quindi il calcolo di tali indici ha senso solo nel caso d'uso test dei modelli unsupervised sviluppati in questo lavoro.

Come si evince dalle equazioni (18) e (19), i due indici restituiscono valori compresi tra 0 e 1 (ICA in [0, 1[e ICC in [0, 1]). La qualità di un esperimento di clustering ha un'elevata accuratezza quando l'indice ICA è pari a zero. In tal caso infatti il numero di cluster calcolati sul dataset corrisponde al numero di cluster attesi. Quanto più il numero di cluster calcolati diverge, in più o in meno, dal numero di cluster attesi, tanto più vicino a 1 tenderà l'indice ICA.

L'elevata completezza di clustering, ossia la congruenza tra numero di cluster calcolati e numero di cluster disgiunti, è garantita quando l'indice ICC è pari a zero.

5.4 DESCRIZIONE SOLUZIONI IMPLEMENTATE

I metodi di post-processing implementati sono stati scelti in completa coerenza con la dicotomia messa in luce precedentemente. Il primo, infatti, è il classico modello K-Means (Hartigan & Wong 1979), appartenente quindi alla classe degli algoritmi partizionali. Viceversa, il metodo denominato "U-Matrix con componenti connesse" (Umat-CC; Hamel & Brown 2011) e il nuovo metodo proposto in questo lavoro, denominato Two Winners Linkage (TWL), sono di tipo gerarchico con approccio bottom-up.

5.4.1 K-Means

Il K-Means è il più classico esempio di clustering partizionale ed opera esattamente come esposto al paragrafo 5.1.1.

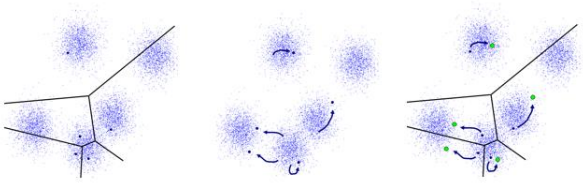


Figura 17 - Algoritmo K-Means

Al fine di escludere totalmente i pattern dall'analisi effettuata in questa fase, si è scelto di non selezionare i centri iniziali dei cluster in base alla distribuzione dei punti del dataset, bensì tra i BMU calcolati dalla SOM. Questo tipo di approccio permette di ridurre la sensibilità al rumore, in quanto i BMU sono medie locali dei dati e quindi meno sensibili a variazioni degli stessi. Anche gli outlier non sono un problema poichè, per definizione, rappresentano solo una piccolissima percentuale del numero di dati e in quanto tali incapaci di influenzare il processo. Tuttavia, se il nostro intento fosse quello di individuare effettivamente gli outlier, allora l'utilizzo di questo approccio sarebbe controproducente.

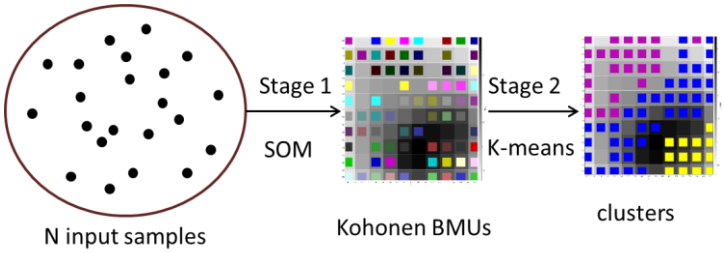


Figura 18 – Clustering a doppio stadio, basato sul pre-clustering con SOM seguita dal K-means

Come già detto, la suddivisone in partizioni avviene tra i nodi BMU dello strato di Kohonen. Affinché questo processo sia applicabile, è quindi necessario individuare un numero di cluster attesi inferiore al numero di BMU individuati dalla SOM.

5.4.2 U-Matrix Con Componenti Connesse

In dataset multi-dimensionali di grandi dimensioni, visualizzare i cluster sulla U-Matrix può risultare un compito non facile. Hamel & Brown (2011) propongono un metodo per migliorare l'interpretazione della mappa di Kohonen, immaginando i nodi della stessa come vertici di un grafo in cui delle componenti connesse (CC) siano identificative di cluster.

Il procedimento con cui le CC sono identificate si basa sul concetto che, per ognuna di esse, esisterà un nodo, definito nodo interno ad una CC, il cui gradiente sarà inferiore rispetto a quello di tutti gli altri. In questo caso per gradiente di un nodo si intende il suo valore output dell'equazione (8).

Per ogni nodo della mappa quindi verranno valutati i gradienti di tutti i suoi nodi adiacenti. Se il nodo esaminato non è quello con il minimo gradiente, allora ci si potrà muovere lungo un percorso attraversando di volta in volta il nodo con gradiente minimo, seguendo i suoi vicini e connettendo tutti i nodi attraversati. Nel caso di un nodo interno ad una CC, la procedura termina, passando ad esaminare un nodo successivo. Mostrando le CC così create, al di sopra della U-Matrix, l'appartenenza di un nodo ad un cluster piuttosto che ad un altro, diventa evidente, come è possibile notare nell'esempio proposto dagli stessi autori e mostrato in Figura 19.

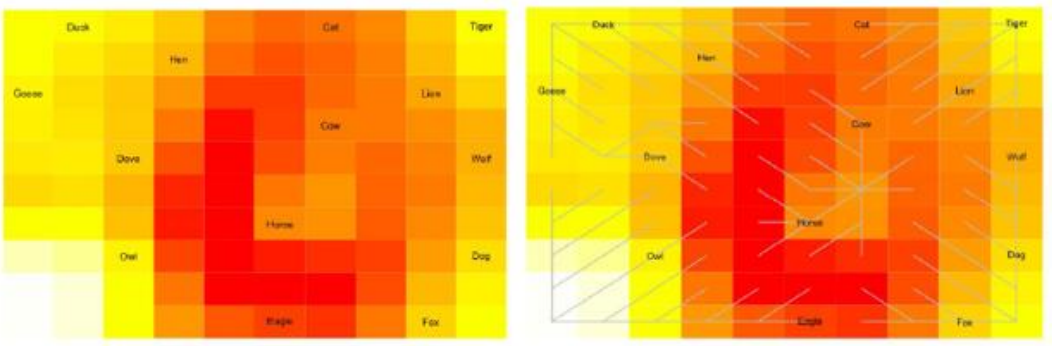


Figura 19 – A sinistra la U-Matrix standard e a destra quella ottenuta con le componenti connesse

L'aspetto grafico del modello Umat-CC, così come suggerito in Hamel & Brown (2011) e mostrato in Figura 19 non è stato implementato, preferendo il metodo di visualizzazione precedentemente esposto, in quanto ritenuto più efficace (vedi Figura 20).

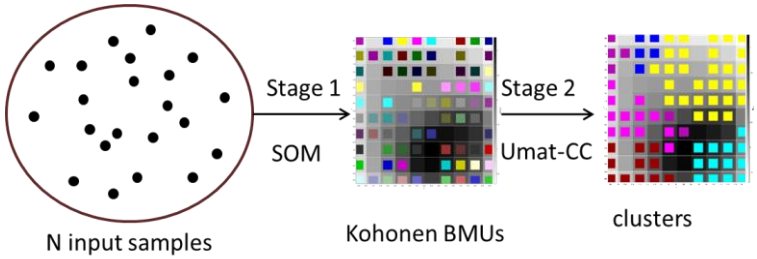


Figura 20 – Clustering a doppio stadio, basato sul pre-clustering con SOM seguita dal metodo Umat-CC

Di seguito viene illustrato il diagramma di flusso del metodo appena descritto.

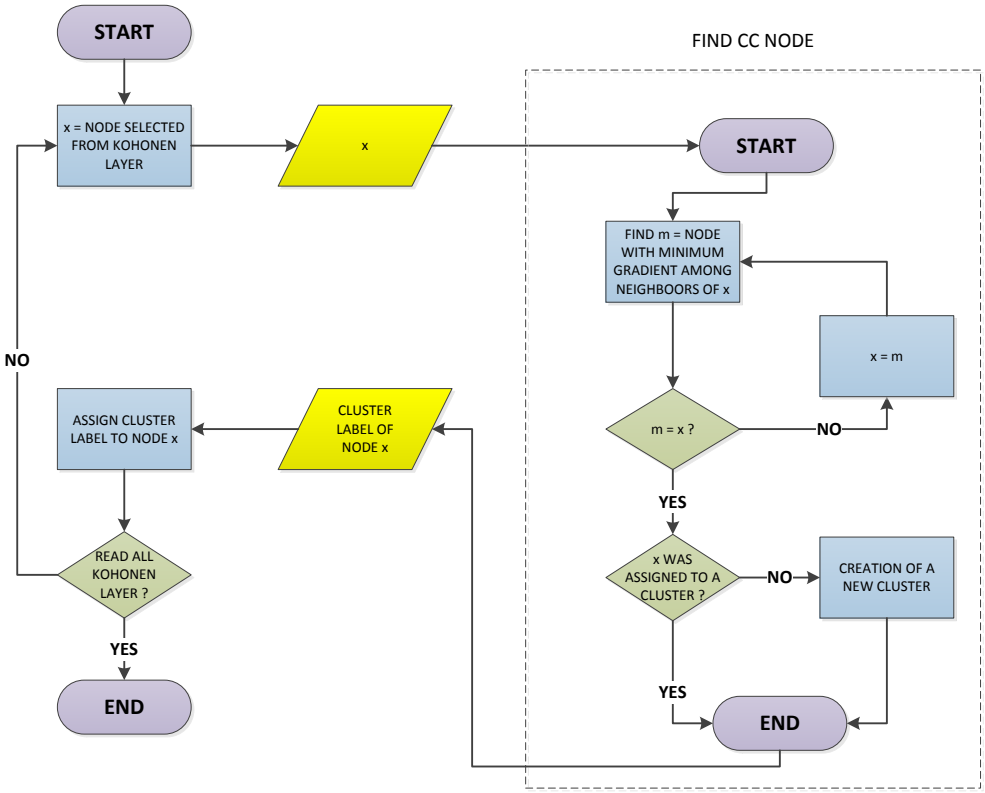


Figura 21 - Diagramma di flusso del metodo Umat-CC

Sperimentalmente gli autori del metodo hanno osservato che l'utilizzo di una tecnica in grado di eliminare inutili frammentazioni della U-Matrix rendendola più omogenea, migliora notevolmente i risultati ottenuti nella ricerca delle CC. Questo effetto è ottenibile tramite l'applicazione alla mappa, di un filtro di sfocatura comunemente utilizzato in diversi software grafici. L'idea di base di questo filtro è che ogni pixel diventi una media pesata dei pixel attorno ad esso. Il peso di ogni pixel è maggior in proporzione alla vicinanza al pixel che si sta attualmente sfocando. L'equazione gaussiana che si utilizza per il calcolo dei pesi è la seguente (Shapiro & Stockman 2001):

$$\boldsymbol{G(i,j) = \frac{1}{2\,\pi\,\sigma^2}\; e^{-\frac{\|i-j\|}{2\,\sigma^2}}} \qquad \textbf{(20)}$$

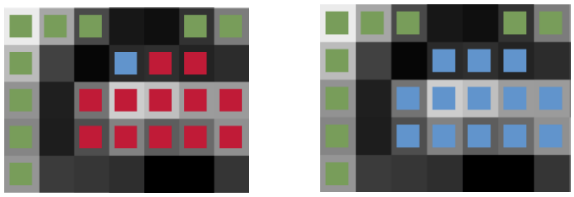
Nella formula precedente:

i, $pixel\ che\ si\ sta\ attualmete\ sfocando$
j, $pixel\ di\ cui\ si\ vuole\ calcolare\ il\ peso$
$\|i-j\|$, $distanza\ in\ termini\ di\ coordinate\ tra\ i\ pixel\ i\ e\ j$
σ, $ampiezza\ della\ gaussiana\ indicante\ grado\ di\ sfocatura$

L'applicazione di questa funzione, a tutti i pixel circostanti il pixel considerato per la sfocatura, fornirà una matrice dei pesi. Le dimensioni di questa matrice dipenderanno dal numero di pixel considerati vicini. Tale parametro, noto come *kernel,* può essere l'intera immagine o avere un raggio più piccolo. A questo punto, il nuovo valore del pixel correntemente analizzato sarà la somma dei valori dei pixel compresi nel raggio di sfocatura moltiplicati per il rispettivo peso.

E' bene notare che i risultati ottenuti con il metodo Umat-CC possono variare, in alcuni dettagli, proprio in base al grado di sfocatura applicato, il quale dipenderà dal parametro σ dell'equazione (20). Poiché non esiste una regola per il calcolo di questo parametro, in quanto dipendente dalle caratteristiche dello specifico esperimento configurato, esso dovrà essere settato di volta in volta qualora il risultato ottenuto non sia ritenuto soddisfacente.

In questo procedimento di regolazione può essere utile far ricorso alla U-Matrix, che verrà mostrata in output senza l'applicazione del filtro. In tal modo, ove l'utente si renda conto di possibili errori, potrà ripetere il post-processing settando un livello di sfocatura più basso o più alto a seconda dei casi.



(a) (b)

Figura 22 - Utilizzo del grado di sfocatura della U-Matrix

In Figura 22 (a), osservando i gradienti dei nodi e ricordando il procedimento con cui l'Umat-CC li clusterizza, si può facilmente intuire che l'attribuzione ad un cluster isolato del nodo colorato in azzurro, può essere un errore dovuto ad un livello di sfocatura eccessivo. In Figura 22 (b) è infatti mostrato il risultato ottenuto con un filtro di sfocatura meno invasivo.

5.4.3 Two Winners Linkage

Il Two Winners Linkage (TWL) è un metodo di clustering, post-processing dello strato di Kohonen, che si ispira alla tecnica per il calcolo del vicinato usata nel modello Evolving SOM (Deng & Kasabov 2003), che verrà analizzato nel capitolo successivo. Tale meccanismo consiste nell'instaurare una connessione tra i due BMU di ogni pattern in modo che, al termine dell'algoritmo, le componenti connesse rivelino i cluster individuati. A differenza di quanto avviene nell' Evolving SOM, le connessioni in questo caso non hanno un peso.

E' necessario, tuttavia, introdurre un meccanismo che consenta di evitare la connessione di nodi troppo lontani. Al fine di ottenere questo risultato, possiamo sfruttare il duale del concetto di nodo interno ad una CC, visto nel paragrafo precedente. Il nodo interno di una CC è stato definito come quel nodo che sulla U-Matrix risulta avere un gradiente inferiore a quello di tutti i nodi ad esso adiacenti. Ciò si verifica nel caso in cui i suddetti nodi siano molto vicini al nodo interno. In virtù di questo, potremmo quindi definire come nodo ***esterno*** ad una CC quel nodo il cui gradiente è maggiore rispetto a quello di tutti i nodi ad esso adiacenti. Per come è stato definito, un nodo esterno individuerà quindi un nodo isolato nello spazio dei pesi.

Figura 23 - Nodo esterno (in rosso) sulla U-Matrix

Un nodo esterno può essere interpretato in diversi modi. Il caso più semplice si verifica quando tale nodo non è risultato BMU di alcun pattern, individuando quindi una zona "vuota" dello spazio dei dati. Tali nodi sono quelli che in pratica rendono la U-Matrix un potente strumento di visualizzazione, distinguendo le zone ad alta densità di pattern (Figura 24).

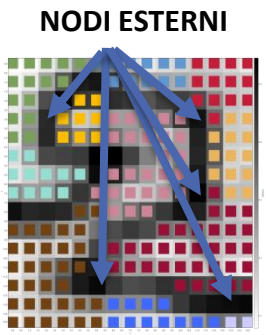


Figura 24 - Localizzazione dei nodi esterni sulla U-Matrix, che individuano le separazioni tra gruppi di BMU

Se viceversa, il nodo esterno è risultato BMU di qualche pattern, possono essersi verificate due situazioni. In primo luogo, il nodo può identificare degli outliers e, in quest'ottica, l'isolamento di tale nodo permette la loro individuazione (Figura 25).

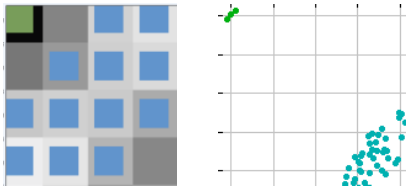


Figura 25 - Nodo esterno come individuatore di outliers

In secondo luogo, nel caso di dati particolarmente complessi, un nodo esterno può essere frutto di una particolare disposizione dei pattern. Quando i cluster non sono infatti nettamente distinguibili, i nodi esterni possono configurarsi come dei "separatori", che si posizionano a metà tra un cluster ed un altro, permettendo quindi al metodo di distinguerli (Figura 26).

Figura 26 - Nodo esterno come separatore di cluster

Nella seguente immagine (Figura 27) è mostrato in maniera chiara il ruolo che i suddetti nodi esterni possono avere nel processo di clustering con il metodo illustrato. Il nodo celeste e i due nodi di tonalità arancione si configurano nettamente come separatori di classe, mentre i pattern indiduati dal nodo verde potrebbero essere degli outliers.

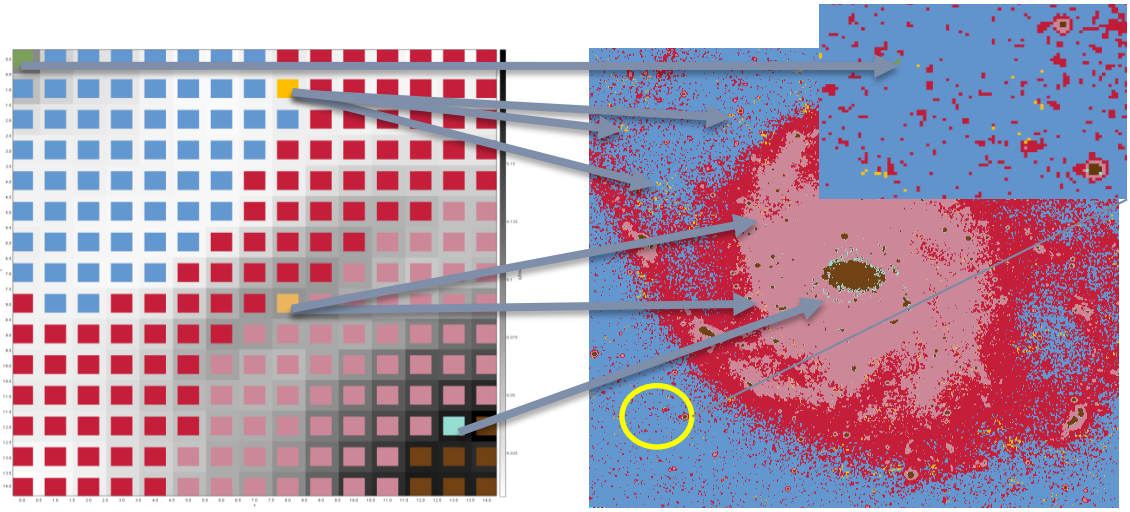


Figura 27 – Ruolo dei nodi esterni per l'individuazione delle zone di separazione e/o di outliers (oggetto in verde)

Alla luce delle considerazioni fatte, il filtro di sfocatura gaussiana della U-Matrix, esposto al paragrafo precedente, dovrà assumere valori molto bassi al fine di evitare di uniformare troppo la mappa, non riconoscendo i nodi esterni come tali.

Di seguito il diagramma di flusso del metodo appena esposto.

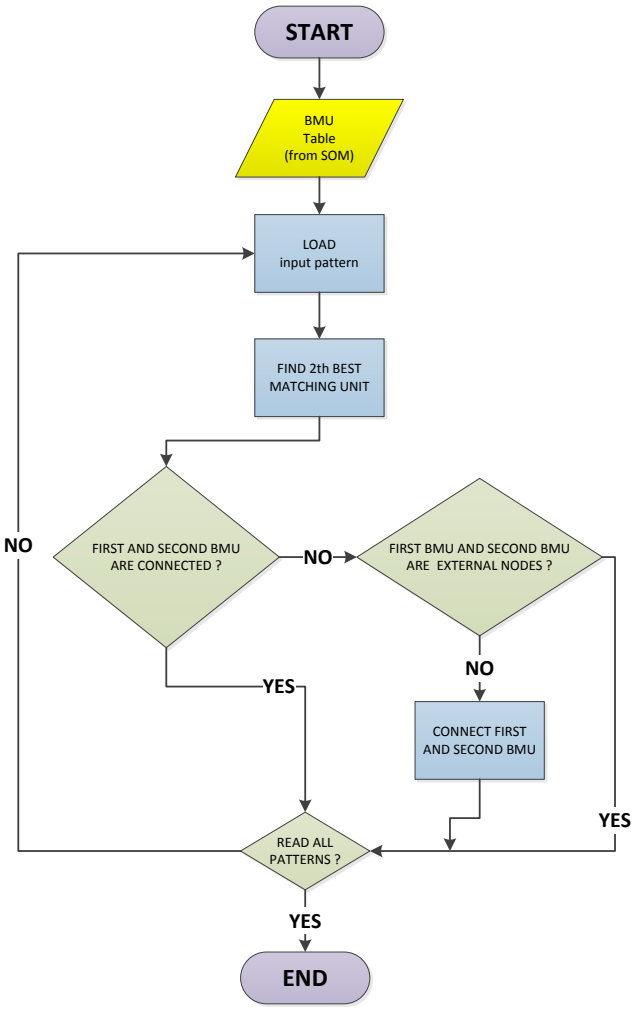


Figura 28 - Diagramma di flusso del metodo TWL

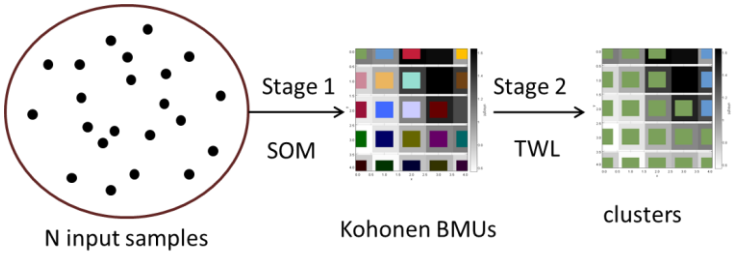


Figura 29 – Clustering a doppio stadio, basato sul pre-clustering con SOM seguita dal metodo Umat-CC

5.4.4 Architettura software del post-processing

Il sistema software relativo al secondo stadio del processo di clustering è stato pensato per venire incontro alle esigenze di diversi utenti, in base alla loro conoscenza sui dati e/o dei modelli implementati. La strategia di fondo è suddividere in tipologie di esecuzione differenti il processo a doppio stadio (Figura 30). In particolare sono disponibili quattro casi d'uso differenti:

1. **SINGLE-STAGE**: in questo caso è possibile evitare la fase di post-processing, rendendo quindi definitivo il proto-clustering effettuato dalla SOM;
2. **AUTOMATIC**: In tal caso sarà il software stesso, in base a dei parametri interni prestabiliti, a selezionare la tecnica di post-processing più opportuna tra K-means e Umat-CC; nel caso in cui l'utente abbia a disposizione delle informazioni sui dati, potrà utilizzarle selezionando il numero di cluster attesi;
3. **EXPERT**: Esiste infine il caso d'uso riservato all'utenza esperta, in cui spetterà all'utente la scelta del metodo da utilizzare e, nel caso il metodo lo richieda, il numero di cluster attesi.

Nell'immagine seguente è mostrato il diagramma di flusso della fase di post-processing che include le tre modalità appena descritte.

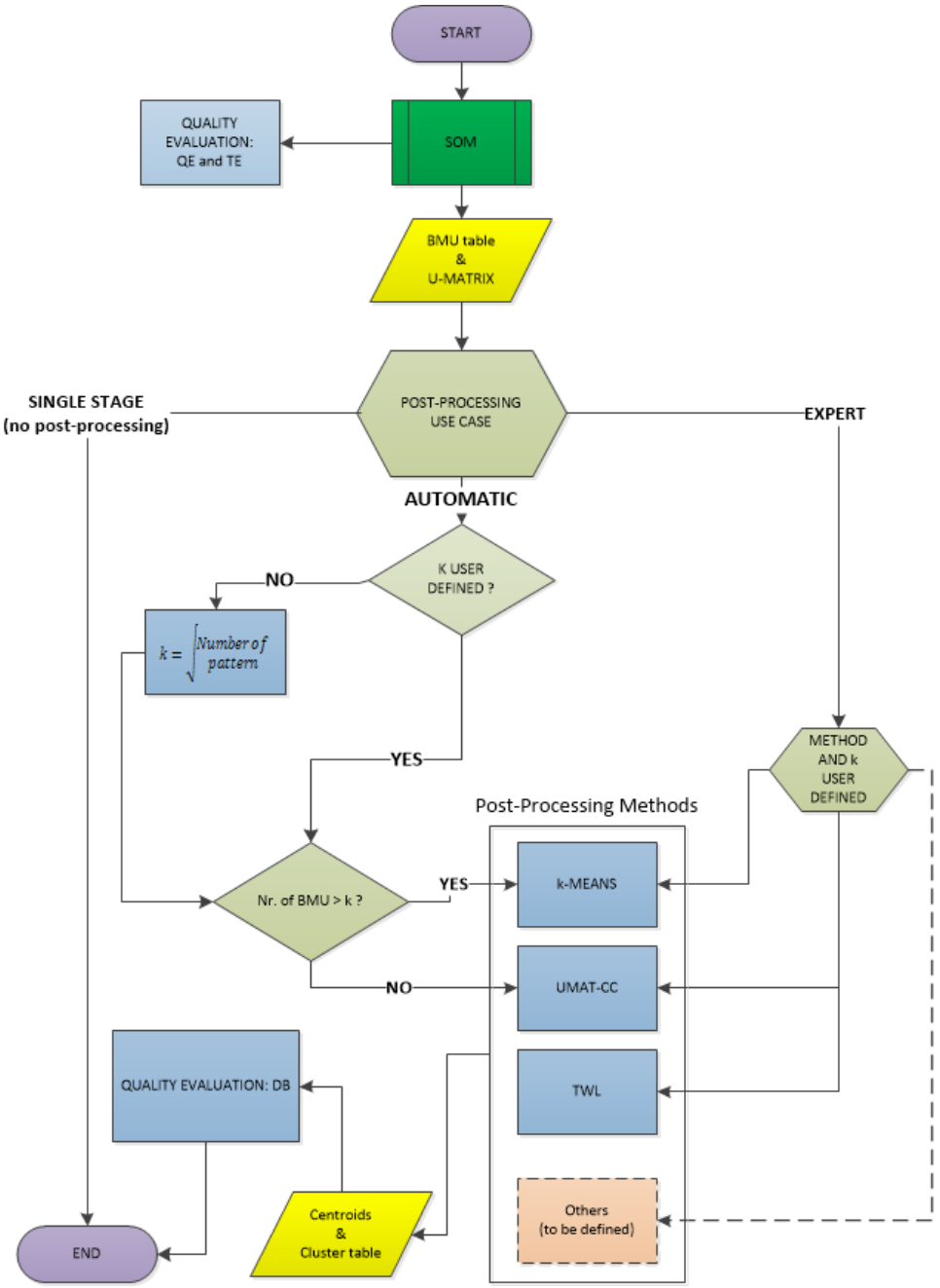


Figura 30 - Flow chart della fase di post-processing

5.4.5 Estensione dei metodi di post-processing

Quanto discusso nel capitolo 5 mette in luce l'impossibilità di stabilire la superiorità di un metodo rispetto ad un altro. E' tuttavia possibile, che alcuni metodi risultino migliori di altri in determinate situazioni e, proprio in virtù di questo, il programma è stato concepito in maniera modulare, in modo da essere facilmente espandibile con altri metodi di post-processing con caratteristiche differenti da quelli attualmente proposti. Il caso d'uso riservato all'utenza esperta permetterà quindi l'applicazione di uno qualsiasi di essi a scelta e tramite i criteri di validità di cui discusso, sarà possibile confrontare i diversi risultati ottenuti.

6 IL MODELLO EVOLVING SOM

In generale il clustering richiede l'uso di una regola basata sulla competizione tra unità di processo, in cui il vettore vincitore (BMU) si candida a rappresentare un certo sottoinsieme di dati input, in teoria accomunati da qualche caratteristica simile nello spazio dei parametri, non nota a priori, né individuabile in modo diretto.

Come abbiamo visto nei capitoli precedenti, l'approccio più semplice consiste nel metodo basato sull'algoritmo del K-Means, che calcola e aggiorna ogni centro dei cluster sottoforma della media dei dati contenuti nel cluster. Il processo di ottimizzazione del K-Means, avulso dalla conoscenza a priori sulla distribuzione dei dati, è in pratica basato sull'uso di una metrica euclidea con cui identificare l'appartenenza di un dato ad un certo cluster, forzando l'ipotesi iniziale che lo spazio dei dati sia partizionabile in un imposto numero K di cluster.

Il modello SOM di Kohonen introduce invece una griglia in cui individuare i prototipi BMU, formando uno spazio dei prototipi vincolato topologicamente (lo strato di Kohonen). In questo modello, la capacità di preservare topologicamente lo spazio di partenza è ottenuta attraverso l'applicazione di una funzione vicinanza per la modifica della posizione dei nodi all'interno della griglia. Poiché ogni BMU è candidato a rappresentare un determinato sottoinsieme dei dati input, il grado di complessità dei dati può influenzare la trasformazione nello spazio di Kohonen, solitamente ridotto rispetto a quello di partenza. Risulta quindi frequente, in presenza di dati molto complessi, che la riduzione delle dimensioni nello spazio di Kohonen non corrisponda al clustering ottimale, ottenendo una ridondanza di BMU, che nella fattispecie si possono considerare come proto-clusters.

Una delle principali limitazioni del modello SOM originario è la staticità della dimensione e della struttura topologica della griglia output. Questo aspetto ha un impatto particolarmente negativo nel caso di problemi di clustering dinamico (on-line clustering) o incrementale, in cui è altresì desiderabile che la struttura della rete e la sua dimensione possa evolvere con l'andamento dei dati acquisiti.

Diverse soluzioni sono presenti in letteratura. In Fritzke (1994), venne proposto il modello Growing Cell Structure (GCS), in cui si adottava una fissata dimensione topologica dello spazio dei vettori prototipo, ma senza un prefissato ordine di rappresentazione dei nodi della rete. Quest'ultima creava nuovi nodi nella misura in cui un dato input non fosse rappresentabile da alcun nodo già presente. Successivamente, in Fritzke (1995) fu introdotto il modello Growing Neural Gas (GNG), in cui spariva la limitazione di una topologia statica del GCS. Parallelamente in Bruske & Sommer (1995) veniva introdotto l'algoritmo Dynamic Cell Structure GCS (DCS-GCS), che differiva dal GNG nel modo in cui un nuovo nodo veniva aggiunto dinamicamente alla rete. Tuttavia tutti i modelli citati richiedevano un surplus di tempo computazionale legato all'inserimento di un nuovo nodo nella griglia, riducendo notevolmente le prestazioni (Esposito 2013).

In questo lavoro abbiamo scelto un metodo simile, denominato Evolving Self Organizing Map (E-SOM; Deng & Kasabov 2003), basato sulla legge di apprendimento e di preservazione della topologia della rete delle SOM, in grado di evolvere dinamicamente l'estensione dello strato di Kohonen, ma in modo più efficiente dei modelli dinamici citati.

In estrema sintesi, le principali caratteristiche del modello E-SOM sono:

- Implementazione di una tecnica self-creating, in grado di superare i limiti imposti da una struttura fissa (SOM);
- L'inizializzazione consiste in una griglia vuota, popolata durante l'avvicendarsi dei dati input;
- Se un dato input non trova corrispondenza adeguata, ovvero la distanza dai nodi già presenti nella rete supera una certa soglia ε, viene aggiunto un nuovo nodo alla rete;
- Il nodo aggiunto corrisponde al pattern input che ne ha causato la creazione. In questo modo si ottiene un'allocazione più semplice che migliora le prestazioni computazionali rispetto ad altri algoritmi a struttura incrementale già citati (GCS, GNG);
- Per ogni pattern input si calcolano i migliori due BMU e se una connessione tra loro esiste, quest'ultima viene aggiornata, altrimenti viene creata (regola basata sul criterio Competitive Hebbian Learning, o CHL);

Da notare che tale modello rappresenta una delle due più interessanti varianti del modello SOM. In generale, infatti, evoluzioni della SOM possono essere derivate partendo da un elemento comune (estensione dinamica del numero di nodi dello strato di Kohonen), ma direzionate su criteri distinti: (i) mantenere la legge di apprendimento di Kohonen invariata, modificandone la topologia; oppure (ii) modificare la legge di apprendimento, mantenendo inalterata la topologia. Al primo criterio appartiene ad esempio il modello Evolving Tree (E-TREE), in cui la topologia dello strato di Kohonen è strutturata sottoforma di albero di ricerca, in cui nuovi nodi sono creati in base alla soglia del numero di volte in cui un nodo ha assunto il ruolo di BMU (Pakkanen et al. 2004).

Il modello E-SOM invece appartiene al secondo filone di varianti evolute della SOM. Di seguito viene descritto il funzionamento interno e la relativa implementazione.

6.1 ALGORITMO DI TRAINING

L'algoritmo inizia con una rete vuota, senza cioè nodi e connessioni. I nodi prototipo vengono creati incrementalmente con l'avvicendarsi dei dati input. Quando viene presentato un pattern input, i nodi prototipo presenti nella rete competono tra loro e si aggiornano le connessioni dei vincitori (BMU). In particolare quando i primi due BMU non sono interconnessi, viene creata una connessione diretta tra loro. Un nuovo nodo nella rete viene creato quando nessun altro nodo preesistente risulta candidato a rappresentare un pattern input. In tal caso il nuovo nodo stabilisce anche delle connessioni con i primi due BMU.

Siano:

$x, \quad pattern\ in\ input$
$W = \{w_1, w_2, \dots, w_N\}, \quad insieme\ dei\ prototipi\ attualmente\ esistenti$
$\varepsilon, \quad soglia\ minima\ di\ distanza$

Un nuovo nodo sarà inserito se:

$$\|\boldsymbol{w_j} - \boldsymbol{x}\| > \boldsymbol{\varepsilon}, \ \forall \boldsymbol{w_j} \in \boldsymbol{W} \tag{21}$$

Il neurone appena inserito è inizializzato come segue:

$$\boldsymbol{w_{N+1}} = \boldsymbol{x} \tag{22}$$

Come si evince dall'eq. (22), un nodo prototipo appena aggiunto viene inizializzato esattamente come rappresentante del pattern che non è stato possibile assegnare a nessuno dei nodi esistenti. Se da un lato questo tipo di approccio risulta computazionalmente molto efficiente (poiché evita i calcoli necessari alla computazione del punto cui inserire il nuovo nodo, come invece accade nel GNG), è altresì importante notare che è notevolmente sensibile al rumore e che può quindi influenzare il clustering eseguito. Tuttavia è possibile porre rimedio a questo problema utilizzando un meccanismo di eliminazione di connessioni obsolete.

Quando un pattern presentato può essere invece assegnato ad un nodo, allora l'attivazione a_j di quest'ultimo sarà calcolata tramite la seguente equazione:

$$\boldsymbol{a_{j=}\, e^{\frac{-2\left\|x-w_j\right\|^2}{\varepsilon^2}}} \quad \textbf{(23)}$$

6.1.1 Aggiornamento dei pesi

La regola generale per l'aggiornamento dei pesi è la seguente:

$$\boldsymbol{\Delta\, w_i = \gamma\, h_i(x,b)\,(x - w_i)} \quad \textbf{(24)}$$

Dove:

γ, $tasso\ di\ apprendimento, tipicamente\ costante\ settato\ a\ 0.05$
h_i, $funzione\ di\ vicinato$
b, BMU

E' evidente come, lasciando invariata la funzione di vicinato, la regola di aggiornamento dei pesi sia esattamente quella proposta da Kohonen.

Tuttavia, nell'E-SOM, il vicinato di un nodo è definito come l'insieme dei prototipi con cui il suddetto nodo ha instaurato una connessione, come mostrato nella seguente formula:

$$\boldsymbol{\Omega(i) = \{j \mid s(i,j) > 0\}} \quad \textbf{(25)}$$

Dove:

$s(i,j)$, $peso\ della\ connessione\ tra\ i\ e\ j$

La funzione di vicinato è la seguente:

$$\boldsymbol{h_{i,b}(x) = \frac{a_i(x)}{\sum_k a_k(x)}} \quad \textbf{(26)}$$

Alla luce di quanto detto, l'aggiornamento dei pesi può essere così formulato:

$$\boldsymbol{\Delta\, w_i = \begin{cases} \gamma \frac{a_i}{\sum_k a_k}\,(x - w_i), & if\ i \in \Omega(b) \\ 0\ else \end{cases}} \quad \textbf{(27)}$$

6.1.2 Connessioni tra i nodi

Come anticipato nei precedenti paragrafi, quando non vi è la necessità di aggiungere un nuovo nodo, si controlla che i primi due BMU per il pattern presentato, siano o meno connessi tra di loro. In caso di connessione assente, quest'ultima sarà creata, dopodiché si proseguirà con l'aggiornamento delle connessioni tra il BMU ed i suoi vicini, secondo la seguente formula:

$$\boldsymbol{s(i,j) = \beta\, s(i,j) + (1 - \beta)\, a_i\, a_j} \quad \textbf{(28)}$$

Dove β è una costante euristicamente settata a 0.8.

Il meccanismo utilizzato delle connessioni tra i nodi nel modello E-SOM presenta diversi vantaggi:

- Non essendo presente la tipica griglia dello strato di Kohonen, le connessioni risultano un metodo computazionalmente poco costoso di stabilire il vicinato di un nodo. Altri metodi, come ad esempio la classifica di vicinanza stilata nell'algoritmo del GNG, hanno bisogno di un numero di operazioni più elevato;
- Le connessioni possono essere utilizzate per visualizzare i cluster. Infatti, se è stato applicato un metodo in grado di eliminare connessioni deboli e/o obsolete, gruppi di neuroni connessi possono essere interpretati come cluster. Il modello prevede, dopo la presentazione di un certo numero di cluster, il taglio della connessione più debole. All'inizio del processo di training, viene avviato il *pruning timer* allo scadere del quale si procederà con il taglio. Il suddetto timer verrà quindi resettato e il processo andrà avanti per tutto il dataset.

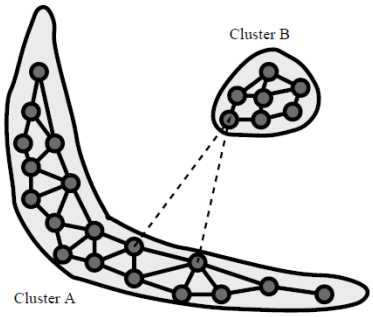


Figura 31 - Nel modello E-SOM i nodi connessi identificano i cluster

6.2 DIAGRAMMA DI FLUSSO DEL MODELLO E-SOM

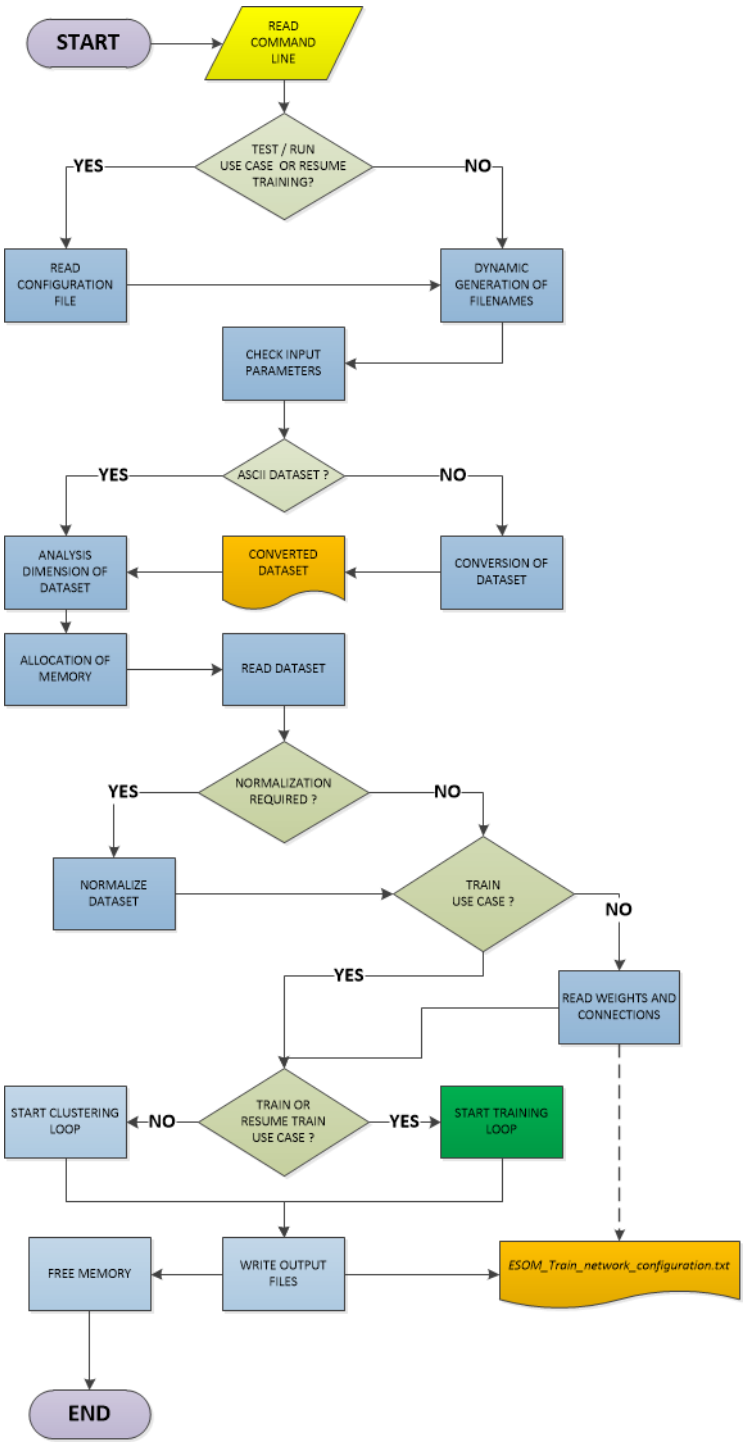


Figura 32 - Flow Chart generale dell'algoritmo E-SOM. In verde scuro il blocco relativo alla procedura di training

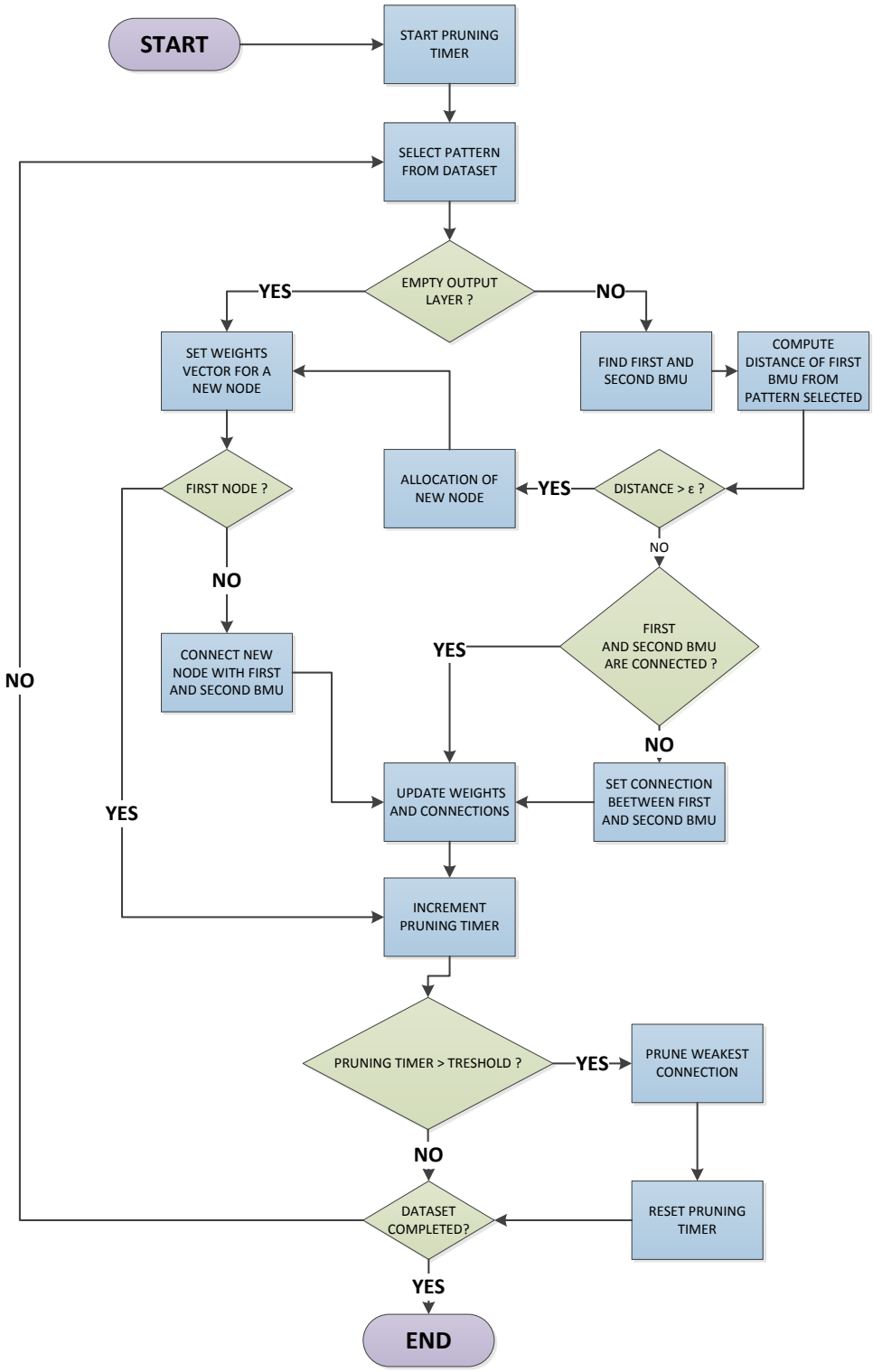


Figura 33 - Flow Chart del loop di training del modello E-SOM

6.3 RAPPRESENTAZIONE GRAFICA CON ADATTAMENTO DELLA U-MATRIX

Poiché nel modello E-SOM i neuroni non sono inseriti all'interno di una struttura rigida come la griglia dello strato di Kohonen, è evidente l'impossibilità di utilizzare la classica U-Matrix come strumento di visualizzazione (descritto nel paragrafo 4.3). Tuttavia, al fine di fornire un metodo di visualizzazione del risultato del clustering, è stata implementata una U-Matrix modificata, in cui i neuroni sono disposti sulla griglia non in base all'effettiva posizione nello spazio dei pesi, bensì raggruppati per cluster di appartenenza.

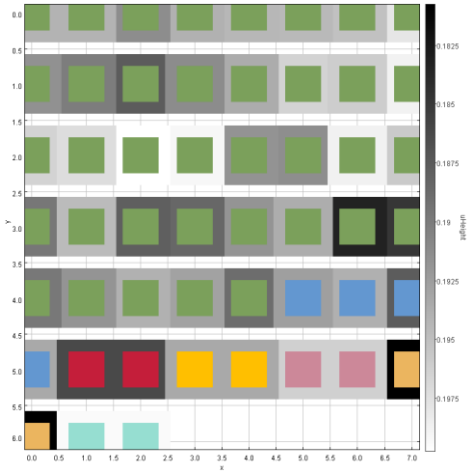


Figura 34 - U-Matrix modificata del modello E-SOM

Come si può notare in Figura 34, il numero di cluster trovati è immediatamente visibile, così come il numero di nodi assegnato ad ognuno di essi.

Poiché una delle peculiarità di questo modello è la presenza di connessioni pesate tra i nodi di output, si è scelto di mostrare come gradiente su scala di grigi, la media dei pesi delle connessioni di ogni nodo. In quest'ottica i nodi scuri non hanno connessioni, oppure le hanno molto deboli. Viceversa i nodi chiari sono nodi le cui connessioni con il vicinato sono molto forti.

7 IL MODELLO EVOLVING TREE

Come accennato, il modello SOM è largamente utilizzato in diverse operazioni di analisi dei dati. Ma alcune delle sue caratteristiche intrinseche lo rendono inadatto per analisi di problemi caratterizzati da grandi volumi di dati. La maggior parte delle operazioni sulla SOM iniziano localizzando il BMU tra i nodi dello strato di Kohonen. Il costo di quest'operazione aumenta al crescere delle dimensioni della mappa. Dataset di grandi dimensioni richiedono un gran numero di nodi, rendendo quindi il procedimento molto più lento. Un altro limite è che le dimensioni della mappa vanno scelte a priori.

Gli approcci per la risoluzione di questi problemi possono essere fondamentalmente divisi in due categorie (Brescia 2012). Il primo approccio consiste nell'utilizzare sistemi in grado di evolvere la struttura topologica durante il tempo, come ad esempio gli algoritmi con strato di output incrementale quali l'E-SOM, le GCS e le DCS-GCS. Il secondo approccio consiste nel costruire efficienti strutture di ricerca per velocizzare le operazioni. Infatti, sebbene alcuni algoritmi usino delle strutture gerarchiche solo come aiuto nella classificazione, altri le usano per aumentare la velocità dell'addestramento e della ricerca. Il problema di questi approcci è che diventano esponenzialmente lenti al crescere delle dimensioni dei dati. Non sono cioè scalabili.

L'Evolving Tree (Etree) è un nuovo tipo di rete neurale auto-organizzante creata per adattarsi a problemi di grandi dimensioni (Pakkanen 2003; Pakkanen & Livarinen 2003; Pakkanen et al. 2004). E' un veloce metodo di classificazione gerarchico particolarmente adatto per grandi problemi multi-dimensionali, in grado di affrontare problemi troppo complessi per i metodi classici.

Può essere visto come un mix tra un albero di decisione supervisionato e una mappa auto-organizzante, ovvero non supervisionata. Come un albero di decisione, è in grado di incrementare gerarchicamente la struttura, prendendo decisioni sulla base delle vicinanze tra pattern e vettori dei pesi, senza avere informazioni sulle classi di appartenenza. Come mappa auto-organizzante usa la stessa regola di addestramento il concetto di ricerca del BMU visti nel modello SOM, seppur con utilizzo di struttura e concetto di distanza differenti. Ottimizza inoltre il carico computazionale risultando particolarmente scalabile rispetto alla grandezza del dataset.

Ciò nonostante, come in tutti i modelli di classificazione, ha bisogno che venga, per ogni pattern, specificata una classe di appartenenza, non per effettuare un addestramento supervisionato, bensì per effettuare una successiva fase di post-processing che assegni ad ogni nodo foglia una classe di appartenenza.

7.1 ALGORITMO DI TRAINING

Come mostrato in Figura 35, l'albero consiste di nodi foglia neri e nodi interni bianchi. Ogni nodo ha un vettore dei pesi w_i, che lo colloca da qualche parte nello spazio dei parametri. In più ad esso è associato un contatore b_i, che tiene traccia del numero di volte in cui è stato BMU. Si assuma di avere un dataset di addestramento i cui pattern siano sottoposti, uno alla volta, alla rete.

L'addestramento inizia trovando il BMU con una ricerca *greedy* nell'albero. Per ogni pattern x_i si parte dalla radice e si seleziona il figlio più vicino. Seguendo lo stesso principio, si esaminano i figli di quest'ultimo e si prosegue in questo modo fino al raggiungimento di un nodo foglia che rappresenterà il BMU del pattern.

A questo punto è necessario l'aggiornamento della posizione del nodo foglia. Questo passaggio avviene utilizzando la regola standard di Kohonen:

$$w_i(t+1) = w_i(t) + h_{ci}(t)[x(t) - w_i(t)] \quad \textbf{(29)}$$

$$h_{ci}(t) = \alpha(t) e^{\left[-\frac{d(r_c, r_i)^2}{2\sigma^2(t)}\right]} \quad \textbf{(30)}$$

L' equazione (29) definisce il grado di aggiornamento ed è, come nel modello SOM, una funzione gaussiana. I parametri $\alpha(t)$ e $\sigma(t)$ sono usati per controllare l'ampiezza e il tempo di decadimento della funzione di vicinato. Sono generalmente funzioni che decrescono esponenzialmente nel tempo (eq. (37) e (38)). Il problema sorge nel calcolo della funzione $d(r_c, r_i)$ che indica la distanza tra i nodi r_c ed r_i. L'Evolving Tree infatti non forma una griglia come il modello SOM e quindi non è possibile calcolare tale distanza in termini di coordinate su di essa. Viene usata tuttavia una metrica equivalente, chiamata *tree distance*, visibile in Figura 35 (b).

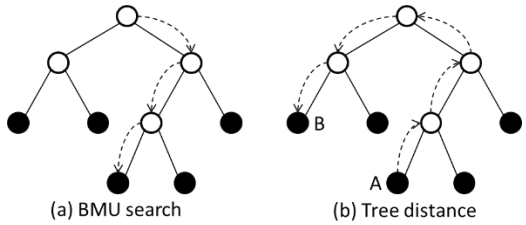


Figura 35 – Un esempio di Evolving Tree. Sulla sinistra la regola per la ricerca di un nodo BMU e sulla destra la regola per il calcolo della *tree distance*.

La *tree distance* è quindi definita come il numero di archi da attraversare per andare dal nodo A al nodo B.

Questi due passaggi, ovvero la ricerca del BMU e l'aggiornamento del vicinato, costituiscono gli aspetti salienti dell'algoritmo di training.

Il terzo passaggio consiste nell'incremento dell'albero. Ogni nodo ha un contatore che indica il numero di volte in cui quest'ultimo sia risultato BMU. Quando il contatore raggiunge un certo valore, chiamato *soglia di splitting,* il nodo viene diviso. Ciò significa che, al termine della procedura di splitting, il nodo diviso avrà diversi figli, diventando quindi un nodo interno. I vettori dei pesi dei figli appena creati sono inizializzati con il valore del vettore dei pesi del padre.

L'addestramento può così continuare usando il nuovo albero divenuto più grande. Una delle conseguenze di questa regola di addestramento è che nodi appartenenti a rami differenti hanno distanze maggiori rispetto a nodi appartenenti allo stesso ramo. Ciò giustifica il fatto che diversi rami crescano in maniera tale da coprire aree differenti dello spazio dei dati.

Di seguito e riportato, in versione semplificata, l'algoritmo di addestramento dell'E-Tree per un singolo pattern:

1. *Creazione di un albero formato da un singolo nodo;*
2. *Calcolo del BMU tramite la ricerca nell'albero (distanza Euclidea tra vettore dei pesi e pattern). La distanza di un pattern x con N features ed un vettore dei pesi w di un nodo j è:* $dist(x_k, w_j) = \sqrt{\sum_{i=1}^{N}(x_k(i) - w_j(i))^2}$
3. *Aggiornamento del nodo foglia usando la regola di Kohonen con la distanza su griglia rimpiazzata dalla tree distance;*
4. *Incremento del contatore di occorrenze del BMU trovato;*
5. *Divisione del nodo se il contatore ha raggiunto la soglia di splitting;*
6. *Se l'albero è cresciuto più del 5% rispetto alla precedente iterazione torna al passo 2, altrimenti termina l'addestramento.*

Questi passi vengono ripetuti finché il sistema non è ritenuto abbastanza buono. Solitamente questo significa scorrere i dati un prefissato numero di volte. Poiché le distanze tra rami sono generalmente abbastanza grandi, l'aggiornamento dei pesi diventa molto piccolo. Per questo il carico computazione può essere ridotto aggiornando solo i nodi adiacenti al BMU.

Una parte molto importante del processo consiste nello stabilire quanto può diventare grande un albero. Se ci sono troppi pochi nodi otterremo solo una rappresentazione grossolana dello spazio dei dati. Viceversa troppi nodi possono causare un'eccessiva specializzazione della rete e, soprattutto, aumentare inutilmente lo sforzo computazionale necessario all'esecuzione. Risulta quindi necessario avere un efficiente criterio d'arresto, ad esempio basato su regolarizzazione e decadimento dei pesi.

Il concetto base della regolarizzazione è penalizzare modelli eccessivamente complessi al fine di prevenire la specializzazione eccessiva dei nodi. Un possibile approccio è quello di creare una funzione di penalizzazione e aggiungerla alla funzione da minimizzare durante l'addestramento, come avviene ad esempio nel Multi Layer Perceptron (MLP), in cui il parametro di penalità è la somma di tutti i pesi (Rumelhart et al. 1986). Questo parametro può essere ridotto portando il valore dei pesi a 0 durante le iterazioni. Poiché nell' E-Tree il motivo dell'eccessiva specializzazione non è la posizione dei nodi, bensì il loro numero, non si guadagnerebbe niente dal decadimento dei vettori dei pesi. Per contenere le dimensioni dell'albero il decadimento dei pesi viene applicato al contatore di occorrenze b_i del BMU:

$$\boldsymbol{b_i(t+1) = \gamma b_i(t)} \quad \textbf{(31)}$$

Questo porta alle seguenti fasi dell'algoritmo che hanno la funzione di controllare la crescita dell'albero e automaticamente interrompere l'addestramento:

1. *Al termine di un'iterazione setta l'equazione (29) per tutti i nodi i, con* γ nell'intervallo $]0, 1[$;
2. *Se l'albero è cresciuto meno di una quantità prefissata, ad esempio il 5%, nell'ultima iterazione, termina l'addestramento.*

Come in tutti gli algoritmi di training, il cuore dell'addestramento risiede nell'aggiornamento dei nodi definito dalle formule (29) e (30). In questa fase sono calcolate le seguenti equazioni:

Come è facile intuire, il punto chiave dell'algoritmo di addestramento è lo step 3. In particolare, la funzione di aggiornamento dei pesi, che implementa le equazioni (29) e (30). Infatti, in questo step vengono calcolare le seguenti equazioni:

$$\boldsymbol{weightbase = \eta_0 e^{-\frac{round}{\tau_2}}} \quad \textbf{(32)}$$

$$\boldsymbol{\sigma = \sigma_0 e^{-\frac{round}{\tau_1}}} \quad \textbf{(33)}$$

$$\boldsymbol{foo = e^{-\frac{d^2}{2\sigma^2}}} \quad \textbf{(34)}$$

$$\boldsymbol{weight = weightbase * foo} \quad \textbf{(35)}$$

$$\boldsymbol{proto(round+1) = proto(round) + weight * [targetvector - proto(round)]} \quad \textbf{(36)}$$

Confrontandole con I termini delle equazioni (29) e (30), si può verificare che:

- l'equazione (35) corrisponde alla (30), con:

  $$\boldsymbol{\alpha(t) = \eta_0 e^{-\frac{round}{\tau_2}}} \quad \textbf{(37)}$$

  $$\boldsymbol{\sigma(t) = \sigma_0 e^{-\frac{round}{\tau_1}}} \quad \textbf{(38)}$$

- l'equazione (36) corrisponde alla (29), cioè la regola di aggiornamento.

7.2 DIAGRAMMA DI FLUSSO DEL MODELLO E-TREE

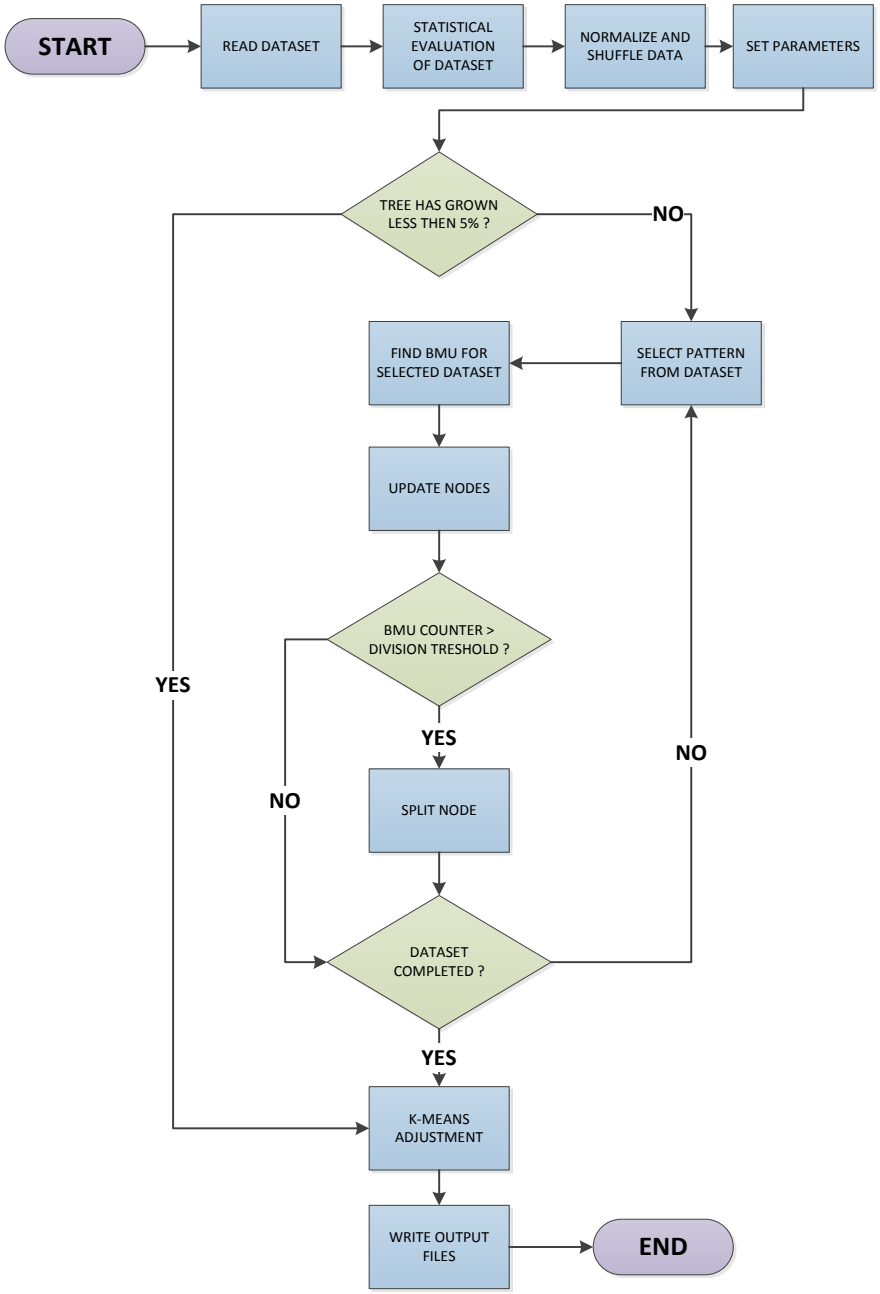


Figura 36 - Flow chart di training del modello E-TREE

8 PROGETTAZIONE DEL SISTEMA SOFTWARE

8.1 ANALISI DEI CASI D'USO DEL MODELLO SOM

I casi d'uso previsti per il modello SOM sono quelli comuni ad un generico modello di machine learning:

- **TRAIN:** fase di apprendimento del modello durante il quale la rete viene addestrata su un insieme di input pattern (dataset), a scelta tra immagini o tabelle;
- **TEST:** fase in cui si sottopone alla rete un insieme di dati di cui sia noto l'output (target), al fine di verificare la qualità dell'addestramento precedentemente eseguito;
- **RUN:** fase in cui il modello viene usato come funzione generica eseguita su input nuovi e non noti a priori;

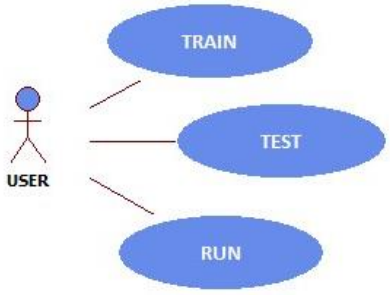


Figura 37 - Diagramma dei casi d'uso del modello SOM

8.2 ANALISI DEI CASI D'USO DELLA FASE DI POST-PROCESSING

I casi d'uso previsti per la fase di post-processing sono i seguenti:

- **SINGLE-STAGE:** in tal caso il post-processing non verrà eseguito lasciando inalterati i risultati della SOM;
- **AUTOMATIC:** il software sceglierà in maniera automatica quale tecnica di post processing applicare;
- **EXPERT:** l'utente avrà facoltà di inserire la tecnica di post-processing da utilizzare e, nel caso la tecnica lo richieda, il numero di cluster attesi;

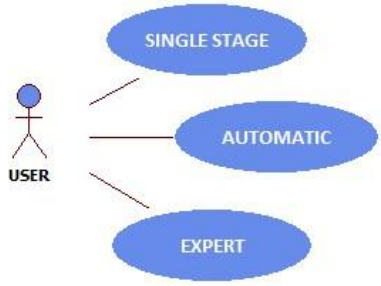


Figura 38 - Diagramma dei casi d'uso del post-processing

8.3 ANALISI DEI CASI D'USO DEL MODELLO E-SOM

I casi d'uso previsti per il modello SOM sono quelli comuni ad un generico modello di machine learning:

- **TRAIN:** fase di apprendimento del modello durante il quale la rete viene addestrata su un insieme di input pattern (dataset), a scelta tra immagini o tabelle;
- **TEST:** fase in cui si sottopone alla rete un insieme di dati di cui sia noto l'output (target), al fine di verificare la qualità dell'addestramento precedentemente eseguito;
- **RUN:** fase in cui il modello viene usato come funzione generica eseguita su input nuovi e non noti a priori;

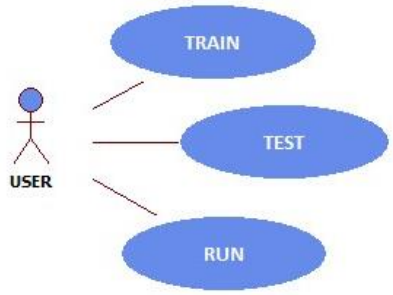


Figura 39 - Diagramma dei casi d'uso del modello E-SOM

8.4 ANALISI DEI CASI D'USO DEL MODELLO E-TREE

Il programma implementato prevede per l'E-TREE i seguenti casi d'uso:

- **TRAIN:** fase di apprendimento del modello durante il quale la rete viene addestrata su un insieme di input pattern (dataset) in formato tabulare. La struttura dell'albero creato in questa fase viene salvata per essere riusata in altri casi d'uso.
- **TRAIN + CV:** aggiunta della K-fold cross validation al caso d'uso Train;
- **TEST (RUN):** fase in cui il modello viene usato come funzione generica eseguita su input nuovi e non noti a priori;
- **FULL:** sequenza automatica dei casi d'uso Train e Test;
- **FULL + CV:** aggiunta della K-fold cross validation al caso d'uso Full.

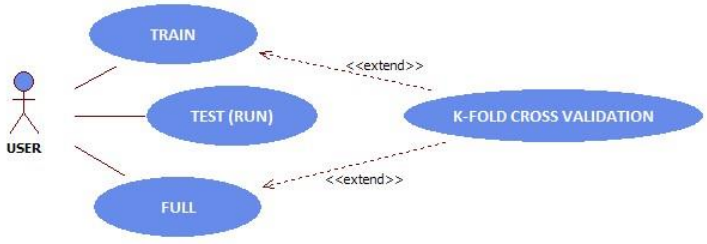


Figura 40 - Diagramma dei casi d'uso del modello E-TREE

8.4.1 K-Fold Cross Validation

La K-fold cross validation è una tecnica che permette di analizzare l'accuratezza di un addestramento eseguito (Kohavi 1995). Consiste nel suddividere casualmente il dataset in *k* partizioni. Di queste, una sarà usata come insieme di validazione e le restanti come insieme di addestramento. Il processo è ripetuto cambiando di volta in volta l'insieme di validazione, finché tutte le partizioni del dataset non siano state usate per la validazione esattamente una volta. I *k* risultati ottenuti possono essere usati in combinazione, ad esempio calcolandone la media, oppure selezionando il peggiore di essi come indice conservativo di qualità.

8.5 INPUT/OUTPUT DEI MODELLI SOM E E-SOM

In questo paragrafo vengono descritti i file di I/O, utilizzati e/o generati per ogni caso d'uso e i parametri da fornire in input ai fini dell'esecuzione dell'esperimento. L'unica differenza è il termine *XXX*, presente nel nome del file, identificante lo specifico caso d'uso cui si riferisce e il prefisso *Model* con cui si intende identificare il modello (SOM oppure E-SOM), relativo al file. Ove vi siano delle eccezioni, il suddetto prefisso verrà sostituito dal nome dello specifico modello.

8.5.1 I/O Generale

8.5.1.1 Input File

- ***Dataset***: può essere composto da immagini o tabelle. Le immagini possono essere di tipo astronomico in formato FITS (2D o 3D) o generiche 2D nei formati Jpeg, GIF o PNG. Le tabelle possono essere estratte dalle immagini oppure inserite nei formati ASCII (DAT, TXT), CSV, FITS-TABLE e VOTABLE. Tali tabelle contengono pattern con la medesima lunghezza.

8.5.1.2 Output File

Come output il programma fornirà i seguenti file:

- ***Model_XXX_params.xml***: file generato automaticamente che contiene tutti i parametri della configurazione dell'esperimento;

- ***Model_XXX_results.txt:*** file con un riga per ogni pattern e che per ognuno di essi riporta

Model_XXX_results.txt	
COLONNA 1	Indice del pattern
COLONNE 2...N	N-1 generiche colonne delle features del pattern (Variabili in base al dataset)
COLONNA N+1	Indice del neurone BMU assegnato
COLONNA N+2	Indice del cluster assegnato
COLONNA N+3	Output della rete relativo al BMU assegnato

Tabella 8 - Struttura del file *Model_XXX_results.txt*

- ***Model_XXX_normalized_results.txt:*** file che viene **generato solo nel caso in cui sia stata richiesta la normalizzazione del dataset**. Presenta la medesima struttura del file precedentemente descritto, con la differenza che le features sono normalizzate tra -1 e 1;

- ***Model_XXX_clusters.txt:*** file che, per ogni cluster, indica il numero di pattern associati durante il clustering e la percentuale di associazione rispetto al numero totale di pattern. Segue la struttura riportata di seguito:

Model_XXX_clusters.txt	
COLONNA 1	Etichetta del cluster
COLONNA 2	Numero totale dei pattern associati al cluster
COLONNA 3	Percentuale di associazione
COLONNA 4	Centroide del cluster

Tabella 9 – Struttura del file *Model_XXX_clusters.txt*

- ***Model_XXX_status.log***: file che contiene informazioni relative all'esperimento. Di seguito è riportata la lista di informazioni ivi contenute, distinte a seconda del modello utilizzato:

Model_XXX_status.log				
ITEM	**SOM**	**SOM row index**	**E-SOM**	**E-SOM row index**
Output directory	x	1	x	1
Input dataset	x	2	x	2
Dataset convertito (Se è stata necessaria la conversione)	x	3	x	3
Caso d'uso	x	4	x	4
File di configurazione della rete (Obbligatorio nei casi di Test e Run)	x	5	x	6
File dei cluster target (Solo in caso di Test)	x	6	x	7
Numero di neuroni dello strato di input	x	7	x	8
Numero di neuroni dello strato di output	x	8		
Numero di righe dello strato di output	x	9		
Numero di colonne dello strato di output	x	10		
Numero massimo di iterazioni	x	11		
Tasso di apprendimento iniziale	x	12	x	9
Soglia minima del tasso di apprendimento	x	13		
Soglia di distanza minima			x	10
Intervallo di tempo per il taglio delle connessioni			X	11
Diametro del vicinato	x	14		
Flag per l'uso di uno strato di output tridimensionale	x	15		
Flag per la richiesta di normalizzazione del dataset	x	16	x	12
Caso d'uso della fase di post-processing	x	17		
Metodo di post-processing selezionato	x	18		
Numero di cluster attesi (Solo se il metodo di post-processing lo richiede)	x	19		
Grado di sfocatura della U-Matrix (Solo sei il metodo di post-processing lo richiede)	x	20	x	13
Numero di iterazioni completate	x	21		
Indice di apprendimento finale	x	22		
Data di inizio dell'esperimento	x	23	x	14
Data di fine dell'esperimento	x	24	x	15
Tempo trascorso	x	25	x	16

Tabella 10 – Struttura del file *Model_XXX_status.log*

- ***Model_error.log:*** file generato in caso di errore che comporta l'interruzione del programma. Nel file viene riportata la linea di comando immessa, il codice dell'errore e la relativa descrizione;

- ***Model _XXX_histogram.png***: file immagine contenente i valori relativi all'istogramma dei cluster individuati;

- ***Model_XXX_validity_indices.txt:*** file che riporta gli indici di qualità dell'esperimento appena eseguito;

- ***Model_XXX_U_Matrix.png:*** immagine raffigurante la U-Matrix relativa alla mappa;

- ***Model_XXX_output_layer.txt:*** file contenente informazioni sullo strato di Kohonen nella SOM oppure sulla griglia modificata implementata nel modello E-SOM, necessario per la visualizzazione della U-Matrix. Il file contiene, per ogni nodo di output:

Model_XXX_output_layer.txt	
COLONNA 1	Indice incrementale del nodo
COLONNA 2	Coordinata x sulla griglia
COLONNA 3	Coordinata y sulla griglia
COLONNA 4	Coordinata z sulla griglia **(solo in caso di griglia tridimensionale del modello SOM)**
COLONNA 5	Cluster assegnato al nodo
COLONNA 6	Numero di pattern per cui il nodo è risultato BMU
COLONNA 7	Valore relativo alla distanza media da tutti i suoi vicini sulla griglia

Tabella 11 - Struttura del file *Model_XXX_output_layer.txt*

Nel caso in cui il dataset input sia un'immagine, si genera:

- ***Model _XXX_clustered_image.png***: immagine raffigurante l'effetto di clusterizzazione;

Da notare che, nel caso in cui l'immagine in input sia un'immagine FITS tridimensionale, al nome precedentemente proposto verrà aggiunto il prefisso ***SliceX,*** dove X rappresenta la slice cui l'immagine si riferisce.

- ***Model_XXX_clustered_datacube_image.zip:*** file zip, generato in caso di immagine tridimensionale, contenente un numero di immagini pari al numero di slice dell'immagine in input;

- ***Model_XXX_clustered_image.txt:*** file che per ogni pixel (pattern) indica ID, coordinate sull'immagine e cluster assegnato. Segue la struttura sotto riportata:

Model_XXX_clustered_image.txt	
COLONNA 1	ID del pixel (pattern)
COLONNA 2	Coordinata x
COLONNA 3	Coordinata y
COLONNA 4	Coordinata z (solo in caso di data cube)
COLONNE (4)5…N	N-1 generiche colonne delle features del pattern (Variabili in base al dataset)
COLONNA N+1	Cluster assegnato

Tabella 12 - Struttura del file *Model_XXX_clustered_image.txt*

8.5.2 I/O per il caso d'uso Train

Caso d'uso: Train
Attore/i: Utente
Input: i parametri, elencati nella seguente tabella, verranno passati da linea di comando

SOM - PARAMETRI IN INPUT – FASE DI TRAIN				
PARAMETRO	OPZIONE LINEA DI COMANDO	DESCRIZIONE	VALORE DI DEFAULT	VALORI AMMESSI
useCase(*)	-u	Use case selezionato	-	1 → Train
inputDataset (*)	-p	Nome del dataset in input	-	-
nInputNodes (*)	-i	Numero di neuroni dello strato di input	-	Intero > 0
gridRows (*)	-r	Numero di righe della griglia dello strato di output	-	Intero > 1
gridColumns (*)	-c	Numero di colonne della griglia dello strado di output	-	Intero > 1
maxIterations	-t	Numero di iterazioni	200	Intero > 0
neighborSize	-n	Diametro di vicinanza	3	Intero > 2
eta0	-e	Indice di apprendimento iniziale	0.7	Reale ϵ]0, 1[
eta1	-f	Indice di apprendimento finale	0.005	Reale ϵ]0, eta0]
normalization	-z	Flag di normalizzazione input in [-1, 1]	0	0 → no normalizzazione 1 → normalizzazione
data_type (*)	-d	Formato del dataset in input	-	ASCII table image fits_image
treD	-g	Flag di creazione strato di Kohonen a 2 o 3 dimensioni	0	0 → (2D) 1 → (3D)
postProcessCase	-o	Caso d'uso del post processing	1	0 → No PostProcessing 1 → Automatico 2 → Esperto
postProcessMethod	-m	Metodo di post processing	0	Intero > 1
expectClusters	-x	Numero di cluster attesi	$\sqrt{Numero\ di\ pattern}$	Intero > 1
SigmaGaussianBlur	-b	Grado di sfocatura della U-Matrix	1.5	Reale > 0
configurationFile	-w	Nome del file di configurazione	-	-

(Il parametro è obbligatorio da parte dell'utente. Ove non specificato altrimenti, un parametro è opzionale).*

Tabella 13 – Parametri in input alla fase di Train per il modello SOM

E-SOM - PARAMETRI IN INPUT – FASE DI TRAIN				
PARAMETRO	OPZIONE LINEA DI COMANDO	DESCRIZIONE	VALORE DI DEFAULT	VALORI AMMESSI
useCase(*)	-u	Use case selezionato	-	1 → Train
inputDataset (*)	-p	Nome del dataset in input	-	-
nInputNodes (*)	-i	Numero di neuroni dello strato di input	-	Intero > 0
Normalization	-z	Flag di normalizzazione input in [-1, 1]	0	0 → no normalizzazione 1 → normalizzazione
Epsilon(*)	-t	Soglia di distanza minima tra pattern input e nodi	-	reale > 0
Pruning_time(*)	-r	Intervallo di tempo dopo il quale effettuare il taglio della connessione più debole	-	Reale > 0
learning rate	-e	Tasso di apprendimento	0.05	Reale in]0, 1[
data_type (*)	-d	Formato del dataset in input	-	ASCII table image fits_image
configurationFile	-w	Nome del file di configurazione	-	-

(Il parametro è obbligatorio da parte dell'utente. Ove non specificato altrimenti, un parametro è opzionale).*

Tabella 14 – Parametri in input alla fase di Train per il modello E-SOM

Precondizioni: l'utente deve essere correttamente loggato alla suite DAMEWARE e avere creato un proprio workspace nel quale siano caricati i file necessari.

8.5.2.1 Input File

- ***Model_Train_network_configuration.txt:*** il file ottenuto come output della fase di Train che segue la struttura riportata al paragrafo 8.5.2.2. L'utilizzo di tale file di input, in caso di Train, non è obbligatorio, bensì necessario nel caso in cui si voglia continuare l'addestramento di una rete partendo da valori precedentemente ottenuti. In tal caso si parlerà di **Resume Training.**

8.5.2.2 Output File

- ***Model_Train_network_configuration.txt***: file contenente tutti i parametri della rete configurata e che sarà inoltre utilizzato come input nelle fasi di Test e Run. La struttura del suddetto file è la seguente:

Model_Train_network_configuration.txt		
	SOM	**E-SOM**
RIGA 1	Numero di features dei pattern input	Numero di features dei pattern input
RIGA 2	Valori relativi al learning rate iniziale, finale (soglia di convergenza) e numero massimo di iterazioni. Gli ultimi 2 valori sono le due condizioni di STOP dell'algoritmo, ovviamente l'una alternativa all'altra	Numero di neuroni dello strato di output
RIGA 3	Numero di righe colonne di cui si compone lo strato di Kohonen e diametro di vicinanza	Soglia minima di distanza
RIGA 4	Flag indicante il tipo di griglia dello strato di Kohonen da usare (0 2D, 1 3D) e flag indicante la normalizzazione o meno del dataset in input (0 No, 1 Si)	Flag indicante la normalizzazione o meno del dataset in input (0 No, 1 Si)
RESTANTI RIGHE	Contengono i pesi associati ad ogni coppia di neuroni (input-output) della rete	Contengono i pesi associati ad ogni coppia di neuroni (input-output) della rete

Tabella 15 – Struttura del file *Model_Train_network_configuration.txt*

8.5.3 I/O per il caso d'uso Test

Caso d'uso: Test
Attore/i: Utente
Input: i parametri, elencati nella seguente tabella, verranno passati da linea di comando

SOM - PARAMETRI IN INPUT – FASE DI TEST				
NOME PARAMETRO	OPZIONE LINEA DI COMANDO	DESCRIZIONE	VALORE DI DEFAULT	VALORI AMMESSI
useCase(*)	-u	Use case selezionato	-	2 → Test
inputDataset(*)	-p	Nome del dataset in input	-	-
data_type (*)	-d	Formato del dataset in input	-	ASCII table image fits_image
postProcessCase	-o	Caso d'uso del post processing	1	0 → No PostProcessing 1 → Automatico 2 → Esperto
postProcessMethod	-m	Metodo di post processing	0	Intero > 1
expectClusters	-x	Numero di cluster attesi	$\sqrt{Numero\ di\ pattern}$	Intero > 1
SigmaGaussianBlur	-b	Grado di sfocatura della U-Matrix	1.5	Reale > 0
configurationFile (*)	-w	File di configurazione	-	-
targetClustersFile(*)	-k	File con cluster attesi	-	-

(Il parametro è obbligatorio da parte dell'utente. Ove non specificato altrimenti, un parametro è opzionale).*

Tabella 16 – Parametri in input alla fase di Test per il modello SOM

E-SOM - PARAMETRI IN INPUT – FASE DI TEST				
NOME PARAMETRO	OPZIONE LINEA DI COMANDO	DESCRIZIONE	VALORE DI DEFAULT	VALORI AMMESSI
useCase(*)	-u	Use case selezionato	-	2 → Test
inputDataset(*)	-p	Nome del dataset in input	-	-
data_type (*)	-d	Formato del dataset in input	-	ASCII table image fits_image
configurationFile (*)	-w	File di configurazione	-	-
targetClustersFile(*)	-k	File con cluster attesi	-	-

(Il parametro è obbligatorio da parte dell'utente. Ove non specificato altrimenti, un parametro è opzionale).*

Tabella 17 – Parametri in input alla fase di Test per il modello E-SOM

Precondizioni: l'utente deve aver eseguito una fase di Train e aver caricato nel workspace il file di configurazione ottenuto e il file dati input.

8.5.3.1 Input File

L'utente dovrà fornire in input al programma i seguenti file:

- ***Model_Train_network_configuration.txt:*** il file ottenuto come output della fase di Train che segue la struttura riportata al paragrafo 8.5.2.2;
- ***Model_Test_target.txt:*** file ASCII contenente le etichette dei cluster noti a cui appartengono i pattern del dataset in input. Il numero di cluster attesi riportati nel file deve essere uguale al numero di pattern del dataset

8.5.4 I/O per il caso d'uso Run

Caso d'uso: Run
Attore/i: Utente
Input: i parametri, elencati nella seguente tabella, verranno passati da linea di comando

SOM - PARAMETRI IN INPUT – FASE DI RUN				
NOME PARAMETRO	OPZIONE LINEA DI COMANDO	DESCRIZIONE	VALORE DI DEFAULT	VALORI AMMESSI
useCase(*)	-u	Use case selezionato	-	3 → Run
inputDataset(*)	-p	Nome del dataset in input	-	-
data_type (*)	-d	Formato del dataset in input	-	ASCII table image fits_image
postProcessCase	-o	Caso d'uso del post processing	1	0 → No PostProcessing 1 → Automatico 2 → Esperto
postProcessMethod	-m	Metodo di post processing	0	Intero > 1
expectClusters	-x	Numero di cluster attesi	$\sqrt{Numero\ di\ pattern}$	Intero > 1
SigmaGaussianBlur	-b	Grado di sfocatura della U-Matrix	1.5	Reale > 0
configurationFile(*)	-w	File di configurazione	-	Capitolo 5.3.1

(Il parametro è obbligatorio da parte dell'utente. Ove non specificato altrimenti, un parametro è opzionale).*

Tabella 18 – Parametri in input alla fase di Run del modello SOM

E_SOM - PARAMETRI IN INPUT – FASE DI RUN				
NOME PARAMETRO	OPZIONE LINEA DI COMANDO	DESCRIZIONE	VALORE DI DEFAULT	VALORI AMMESSI
useCase(*)	-u	Use case selezionato	-	3 → Run
inputDataset(*)	-p	Nome del dataset in input	-	-
data_type (*)	-d	Formato del dataset in input	-	ASCII table image fits_image
configurationFile(*)	-w	File di configurazione	-	-

(Il parametro è obbligatorio da parte dell'utente. Ove non specificato altrimenti, un parametro è opzionale).*

Tabella 19 – Parametri in input alla fase di Run del modello E-SOM

Precondizioni: l'utente deve aver eseguito una fase di Train e aver caricato nel workspace il file di configurazione ottenuto ed il file dati input

8.5.4.1 Input File

- ***Model_Train_network_configuration.txt:*** il file ottenuto come output della fase di Train che segue la struttura riportata al paragrafo 8.5.2.2;

8.6 INPUT/OUTPUT DEL MODELLO E-TREE

Questa sezione è dedicate alla descrizione della configurazione I/O per il modello E-TREE.

8.6.1 Input file

- ***Dataset:*** il programma prevede che il dataset in input, sia esso estratto da immagini o meno, deve essere un file ASCII con una precisa struttura: prima dei pattern, e dopo eventuali righe di intestazione, deve essere presente un intero che indica il numero di features di cui si compongono i pattern che occupano le restanti righe del file. L'ultima colonna di ogni riga deve essere una stringa o un intero, identificativo della classe di appartenenza del relativo pattern. Di seguito viene riportato un esempio:

  FILE data_fake.dat:

```
# Prima riga di intestazione
# Può esserci più di una riga
3
0.1 0.2 0.1 good
3 5.5 0 bad
0 0.2 0.2 good
1.1 2.2 2.1 bad
0.3 0.2 0.3 good
```

  Il file di esempio mostrato presenta due righe di intestazione cui segue la riga con il valore 3, corrispondente al numero di features di ogni pattern. Le restanti righe sono occupate dai pattern, ognuno dei quali reca, nell'ultima colonna, la classe di appartenenza, good o bad. Come già detto nei paragrafi precedenti, l'etichetta assegnata ad ogni pattern è usata solo per la valutazione finale e non per l'addestramento che rimane non supervisionato.

- ***File dei parametri:*** file ASCII contenente i parametri di configurazione dell'esperimento. Segue la struttura riportata:

FILE DEI PARAMETRI DEL MODELLO E-TREE	
RIGA 1	Soglia di splitting
RIGA 2	Numero di nodi generati a seguito di uno splitting
RIGA 3	η_0 tasso di apprendimento iniziale
RIGA 4	σ_0 diametro iniziale del vicinato
RIGA 5	τ_1 costante temporale
RIGA 6	τ_2 costante temporale
RIGA 7	Fattore di decadimento del contatore di occorrenze dei BMU
RIGA 8	Numero di esecuzioni del K-Means per riordinare le foglie dell'albero

Tabella 20 - Struttura del file di configurazione del modello E-Tree

8.6.2 Parametri in input

L'esecuzione del programma avviene da linea di comando specificando i parametri necessari. Seguono le possibili linee di comando immettibili:

- `Etree TRAIN NOCV <training data file> <tree file> <parameter file>`
- `Etree TRAIN CV <k> <training data file> <trained tree output file> <parameter file>`
- `Etree TEST <training data file> <testing data file> <tree file> <test output file>`
- `Etree FULL NOCV <training data file> <testing data file> <parameter file> <tree file> <test output file>`
- `Etree FULL CV <k> <training data file> <testing data file> <parameter file> <tree file> <test output file>`

La seguente tabella fornisce invece una descrizione di ogni parametro.

E-TREE – DESCRIZIONE PARAMETRI IN INPUT	
CV / NOCV	Specifica se il caso d'uso selezionato deve essere eseguito con o senza Cross Validation. Se il parametro è uguale a CV, ovvero la Cross Validation è stata richiesta, il parametro *k* deve essere specificato immediatamente dopo.
<training data file>	Dataset da usare per il Train
<testing data file>	Dataset da usare per il Test / Run
<parameter file>	File di configurazione contenente i parametri
<tree file>	File contenente la struttura dell'albero creato durante il Train, fase di cui il file è output. Sarà invece necessario come input nella fase di Test / Run
<test output file>	File dove salvare i risultati della fase di Test / Run

Tabella 21 - Parametri in input del modello E-Tree

8.6.3 Output

La seguente tabella mostra i file di output generati dal modello E-Tree

E-TREE – FILE DI OUTPUT	
Trainlog.txt	File riportante lo stato dell'esecuzione della fase di Train
Testlog.txt	File riportante lo stato dell'esecuzione della fase di Test / Run
TrainCVlog.txt	File riportante lo stato dell'esecuzione della fase di Train con Cross Validation
outNewickTree.txt	File contenente la struttura dell'albero ottenuto dalla fase di Train.
testStatOut.txt	File riportante, per ogni pattern, ID, classe assegnata e classe attesa

Tabella 22 – File di output del modello E-Tree

La prima riga del file ***outNewickTree.txt*** salva l'albero in un formato visualizzabile tramite uno specifico tool reperibile all'indirizzo http://en.wikipedia.org/wiki/T-REX_(Webserver). In Figura 41 è mostrato un esempio riferito all'esempio di dataset in input riportato precedentemente (data_fake.dat).

Figura 41 – Un esempio di output del modello E-TREE dopo una fase di Train. Sulla sinistra la rappresentazione in format testuale e sulla destra il corrispondente format grafico.

9 TESTING

Questo paragrafo è dedicato alla validazione tecnica e scientifica dei vari componenti software oggetto del presente lavoro. Per ogni dataset proposto saranno mostrati i risultati relativi a tutti i modelli discussi nel documento. I risultati relativi ai diversi post-processing della SOM, salvo alcuni casi specifici, sono ottenuti a partire dalla medesima rete addestrata.

9.1 CONFIGURAZIONI DEI PARAMETRI DI INPUT

In questo paragrafo sono state riassunte le varie configurazioni dei parametri di input per tutti i modelli, nei vari casi di test, i cui risultati sono descritti nei successivi paragrafi. Ciò per permettere la riproducibilità dei test enucleati in questo documento. Per ogni tabella seguente, i parametri non definiti si presuppone assumano i rispettivi valori di default.

SOM	Hepta	Chainlink	Target	Golfball	M101	POSSII-XP559	CFHT-D1
Nodi input	3	3	2	3	1	1	5
Colonne grid output	20	15	15	8	15	15	15
Righe grid output	20	15	15	8	15	15	15
Vicinato iniziale	20	15	20	10	10	10	10
N° max iterazioni	200	300	100	1000	200	100	200
Normalizzazione	No	No	No	No	No	No	Si
Modalità post-processing	0	0	0	0	0	0	0

Tabella 23 – Configurazione parametri di input del modello SOM

SOM + K-Means	HEPTA	Chainlink	Target	M101	POSSII-XP559	CFHT-D1
Nodi input	3	3	2	1	1	5
Colonne grid output	20	15	15	15	15	15
Righe grid output	20	15	15	15	15	15
Vicinato iniziale	20	15	20	10	10	10
N° max iterazioni	200	300	100	200	100	200
Normalizzazione	No	No	No	Si	Si	Si
Modalità post-processing	2	2	2	2	2	2
Metodo di post-processing	1	1	1	1	1	1
N° clusters attesi (k)	7	2	6	6	6	11

Tabella 24 – Configurazione parametri di input del modello SOM + K-Means

SOM + Umat-CC	Hepta	Chainlink	Target	Golfball	M101	POSSII-XP559	CFHT-D1
Nodi input	3	3	2	3	1	1	5
Colonne grid output	20	15	15	8	15	15	15
Righe grid output	20	15	15	8	15	15	15
Vicinato iniziale	20	15	20	10	10	10	10
N° max iterazioni	200	300	100	1000	200	100	200
Normalizzazione	No	No	No	No	Si	Si	Si
Modalità post-processing	2	2	2	2	2	2	2
Metodo di post-processing	2	2	2	2	2	2	2
Grado sfocatura U-matrix	2	2	0.8	1.5	3.5	3.5	2.5

Tabella 25 – Configurazione parametri di input del modello SOM + Umat-CC

SOM + TWL	Hepta	Chainlink	Target	Golfball	M101	POSSII-XP559	CFHT-D1
Nodi input	3	3	2	3	1	1	5
Colonne grid output	5	15	15	8	15	15	15
Righe grid output	5	15	15	8	15	15	15
Vicinato iniziale	5	15	20	10	10	10	10
N° max iterazioni	5	300	100	1000	200	100	200
Normalizzazione	No	No	No	No	Si	Si	Si
Modalità post-processing	2	2	2	2	2	2	2
Metodo di post-processing	3	3	3	3	3	3	3
Grado sfocatura U-matrix	1.5	2	2.4	1.5	0.6	0.3	0.1

Tabella 26 – Configurazione parametri di input del modello SOM + TWL

E-SOM	Hepta	Chainlink	Target	Golfball	M101	POSSII-XP559	CFHT-D1
Nodi input	3	3	2	3	1	1	5
Soglia di distanza minima	1	0.5	0.2	0.5	0.5	0.5	0.05
Pruning time	10	30	5	50	200	100	50
Normalizzazione	No	No	No	No	Si	Si	Si

Tabella 27 – Configurazione parametri di input del modello E-SOM

E-TREE	Hepta	Chainlink	Target	Golfball
Soglia di splitting	10	10	10	10
Numero di nodi generati per splitting	2	2	2	2
η_0 tasso di apprendimento iniziale	0.8	0.8	0.8	0.8
σ_0 diametro iniziale del vicinato	0.6	0.6	0.6	0.6
τ_1 costante temporale	4	4	4	4
τ_2 costante temporale	5	5	5	5
Fattore di decadimento occorrenze BMU	0.8	0.8	0.8	0.8
Numero di esecuzioni del K-Means	2	2	2	2

Tabella 28 – Configurazione parametri di input del modello E-TREE

9.2 HEPTA

Hepta è il primo dei dataset appartenenti alla *Fundamental Clustering Problems Suite* (FCPS) proposta da Ultsch (2005). E' un dataset tridimensionale contenente 212 pattern appartenenti a 7 classi non sovrapposte. In quanto tali ci si aspetta quindi che una procedura di clustering individui sette gruppi come mostrato nella seguente immagine.

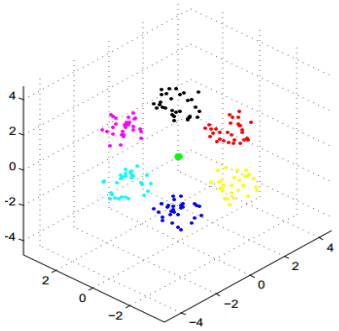


Figura 42 - Clustering teorico del dataset Hepta

E' altresì importante sottolineare che in letteratura tale dataset è considerato l'esempio più semplice di clustering. Pertanto il suo utilizzo risulta la base di verifica delle prestazioni per un qualunque algoritmo di clustering.

9.2.1 SOM Single-Stage

La natura del dataset, con classi ben distinte, fa si che, nonostante l'assenza del secondo stadio di clustering, alcune classi vengano correttamente riconosciute. Tuttavia, risultati di questo tipo non possono essere comunque considerati soddisfacenti ai fini del clustering dei dati.

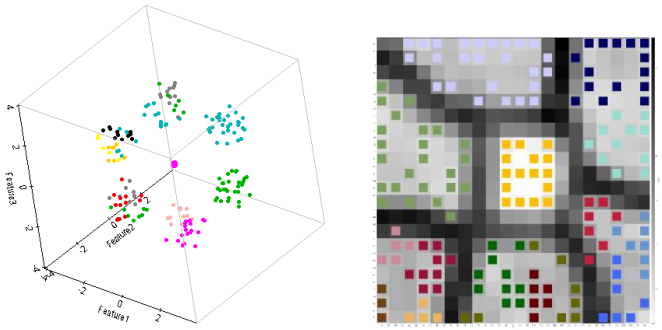


Figura 43 - SOM Single-Stage applicato al dataset Hepta (a destra la relativa U-matrix)

9.2.2 SOM + K-Means

Di seguito sono proposti i risultati ottenuti applicando l'algoritmo del K-Means come post-processing del modello SOM. Essendo noto il numero di cluster attesi, ovvero 7, imposteremo questo numero come parametro del K-Means. Come è possibile notare sia dal grafico che dalla U-Matrix, i sette cluster vengono riconosciuti in maniera corretta.

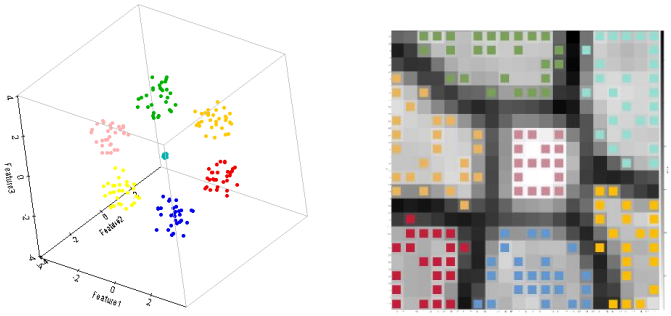


Figura 44 - SOM + K-Means applicato al dataset Hepta (a destra la relativa U-matrix)

9.2.3 SOM + Umat-CC

Come è possibile intuire osservando i risultati del test, di seguito riportati, anche l'Umat-CC si dimostra in grado di effettuare un clustering corretto sul dataset proposto.

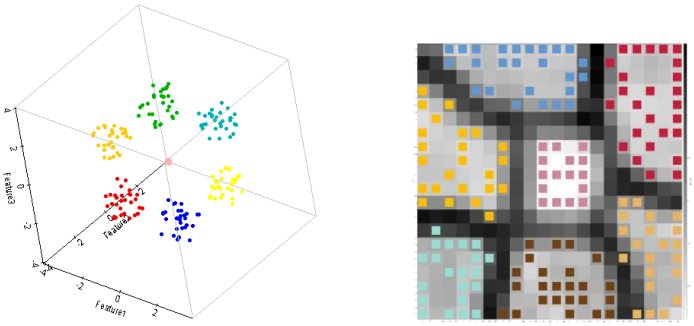


Figura 45 - SOM + Umat-CC applicato al dataset Hepta (a destra la relativa U-Matrix)

Ai fini di un confronto che seguirà nel paragrafo successivo, mostriamo anche i risultati ottenuti su una rete di dimensioni ridotte.

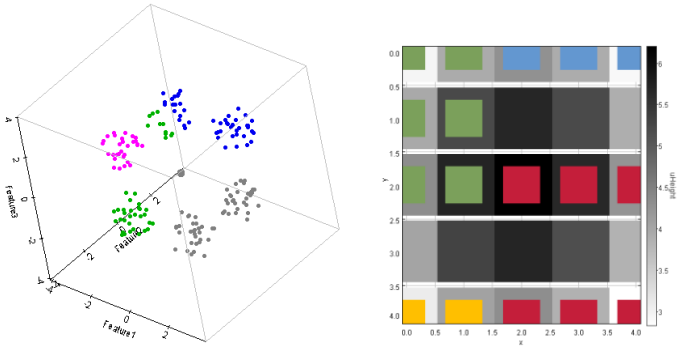


Figura 46 - SOM + Umat-CC (con dimensioni della rete ridotte) applicato al dataset Hepta (a destra la relativa U-Matrix)

9.2.4 SOM + TWL

Si era già detto, nel paragrafo dedicato alla descrizione del metodo TWL, come quest'ultimo fosse, basandosi esso sulla distribuzione dei nodi, influenzato da un numero eccessivo di neuroni della rete in relazione al numero di pattern. In questo caso abbiamo infatti una rete di 400 nodi applicata ad un dataset che consta di 212 pattern. Ciò porta chiaramente la rete a specializzarsi in maniera eccessiva riconoscendo quindi i cluster come mostrato nella seguente U-Matrix.

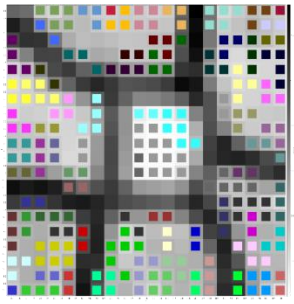


Figura 47 - U-Matrix della SOM + TWL applicata al dataset Hepta

Per confermare la veridicità di quanto detto mostriamo i risultati ottenuti a partire da una rete differente rispetto a quella utilizzata per i precedenti metodi di post-processing. La rete è composta da un numero inferiore di nodi e il risultato è evidente.

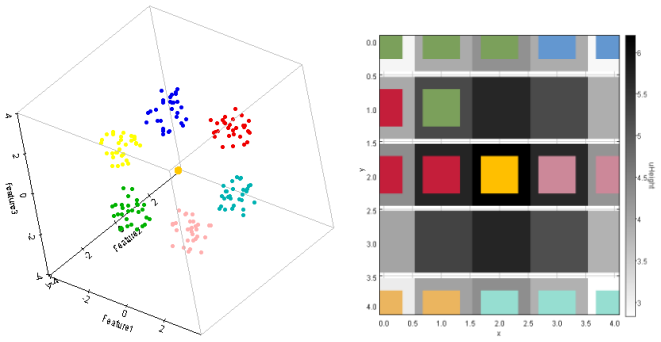


Figura 48 - SOM + TWL (con dimensioni della rete ridotte) applicata al dataset Hepta (a destra la relativa U-Matrix)

Ricordando quanto visto al paragrafo precedente, risulta quindi lampante una differenza di comportamento tra i metodi di post-processing Umat-CC e TWL, rispetto alle dimensioni della rete. L' Umat-CC ha infatti bisogno, per funzionare in maniera ottimale, di reti di grandi dimensioni, che permettano quindi ai nodi di disporsi in maniera tale da formare delle nette zone di separazione. Viceversa, il metodo TWL, onde evitare di riconoscere cluster piccoli e troppo specifici, ha bisogno di un numero di nodi adeguatamente inferiore alle dimensioni del dataset.

9.2.5 E-SOM

Ricordando che il metodo di post-processing TWL prende spunto dal meccanismo delle connessioni proposto nel modello E-SOM, risulta naturale la similarità dei risultati ottenuti. Da notare che, nel modello E-SOM, è il parametro ε a decretare il numero di nodi creati. In questo caso infatti il numero di nodi è rimasto abbastanza contenuto, come si può notare dalla U-Matrix. Tuttavia, riducendo questo parametro, la rete sarebbe indotta a creare un numero maggiore di nodi, ricadendo quindi nell'errore esposto al paragrafo precedente.

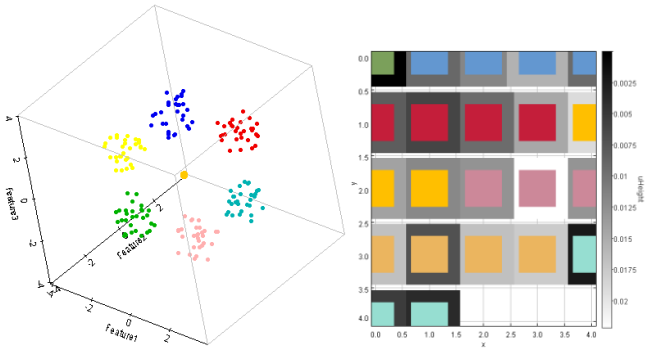


Figura 49 - E-SOM applicato al dataset Hepta (a destra la relativa U-Matrix)

9.2.6 E-TREE

Anche l'E-Tree, come tutti i metodi precedentemente esposti, riconosce le sette classi in maniera perfetta.

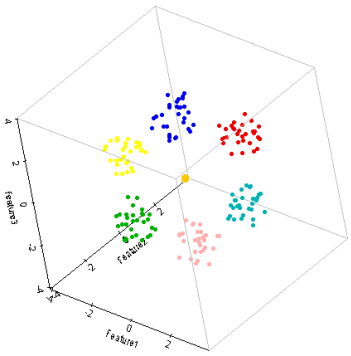


Figura 50 - E-Tree applicato al dataset Hepta

9.2.7 Confronto

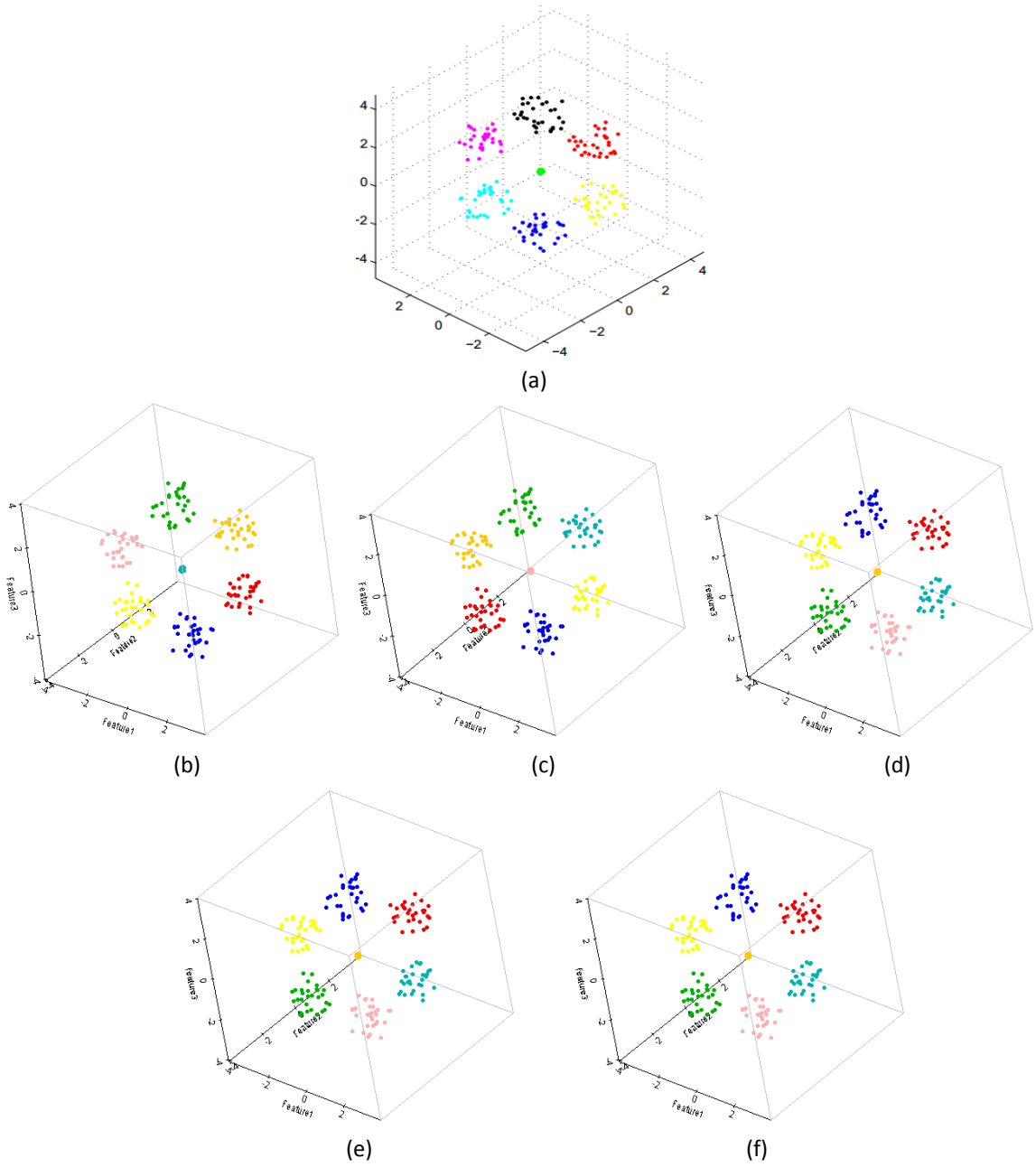


Figura 51 - Confronto del clustering del dataset Hepta. (a) teorico; (b) SOM+K-Means; (c) SOM+Umat-CC; (d) SOM+TWL; (e) E-SOM; (f) E-TREE

Volendo quantificare numericamente il risultato dei confronti di clustering tra i vari metodi su citati, si è proceduto a calcolare gli indici statistici di valutazione dell'errore di clustering per tutti i modelli.

CONFRONTO STATISTICO SU PRESTAZIONI DI CLUSTERING (Hepta)						
MODELLO	E-TREE	SOM	SOM+K-means	SOM+UmatCC	SOM+TWL*	E-SOM
errore di quantizzazione	-	0.16	0.16	0.16	0.48	0.50
errore topografico	-	0.12	0.12	0.12	0.15	-
indice di Davies-Bouldin	-	-	0.36	0.31	0.37	0.40
Accuratezza	0	-	0	0	0	0
Completezza	0	-	0	0	0	0

Tabella 29 – Confronto di prestazioni di clustering tra i vari modelli proposti

** Il test è stato eseguito a partire da una rete di Kohonen diversa da quella addestrata per gli altri metodi di post-processing.*

9.3 CHAINLINK

E' un dataset tridimensionale, tratto anche esso dalla FCPS, contenente 1000 pattern divisi equamente tra due classi, con la peculiarità che le due classi non sono linearmente separabili. Vedremo quindi, il comportamento dei metodi proposti in relazione a questo tipo di problema.

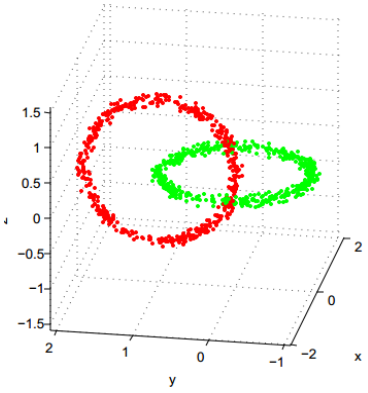


Figura 52 - Clustering teorico del dataset Chainlink

9.3.1 SOM Single-Stage

La U-Matrix ottenuta mette in evidenza la presenza di due cluster ben distinti e, come già detto, la necessità di un meccanismo di post-processing in grado di individuarli.

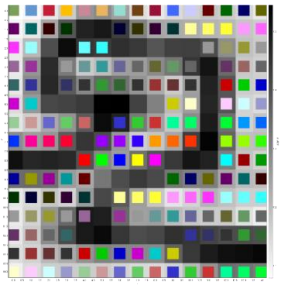


Figura 53 - U-Matrix della SOM Single-Stage applicata al dataset Chainlink

9.3.2 SOM + K-Means

Anche in questo caso sfruttiamo la conoscenza a priori del numero di classi per impostare il parametro del K-Means, con i seguenti risultati.

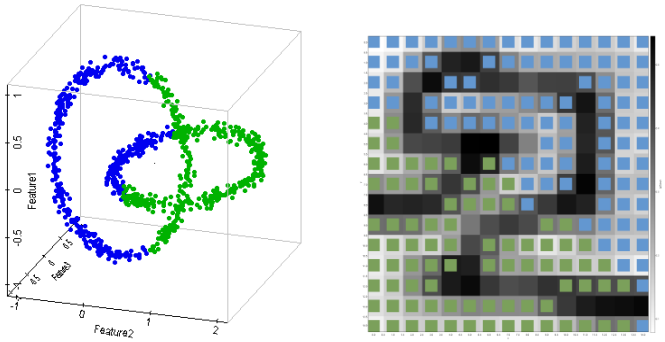


Figura 54 SOM + K-Means applicato al dataset Chainlink (a destra la relativa U-Matrix)

Le immagini mostrate evidenziano il limite che, i metodi partizionali in generale, hanno rispetto a dataset non linearmente divisibili come quello proposto.

9.3.3 SOM + Umat-CC

E' evidente cha il riconoscimento delle due classi, nella loro interezza, non è avvenuto. Tuttavia, è importante notare che nessuno dei cluster individuati contiene punti che appartengono a più di un anello. Più semplicemente si potrebbe concludere che il riconoscimento dei due anelli avviene in maniera corretta ma l'Umat-CC è indotto, a causa della disposizione assunta dai neuroni della rete, a individuare più sotto-cluster all'interno di ogni anello.

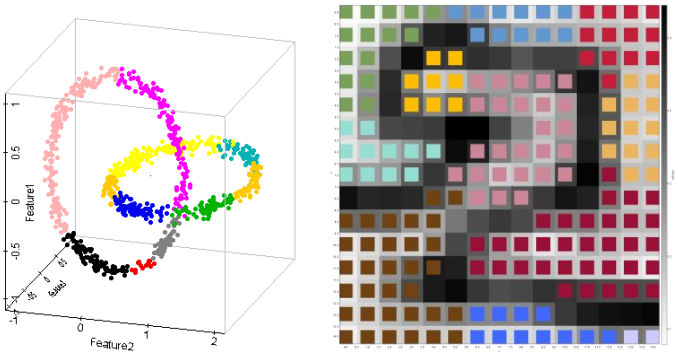


Figura 55 - SOM + Umat-CC applicato al dataset Chainlink (a destra la relativa U-Matrix)

9.3.4 SOM + TWL

I risultati proposti mostrano in maniera netta la superiorità, in questo caso, del metodo di post-processing TWL che è in grado di riconoscere in maniera perfetta i due cluster.

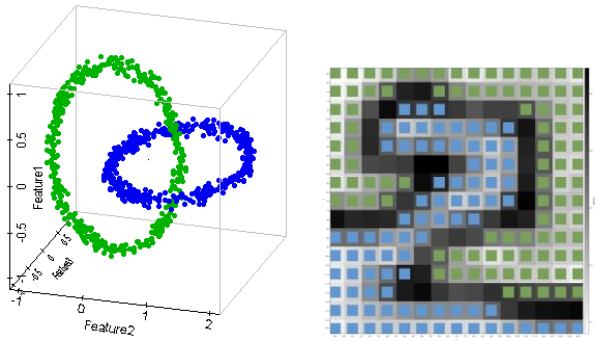


Figura 56 - SOM + TWL applicato al dataset Chainlink (a destra la relativa U-Matrix)

9.3.5 E-SOM

Ancora una volta, i risultati dell'E-SOM sono comparabili a quelli del modello SOM + TWL. Il riconoscimento delle due classi avviene in maniera corretta.

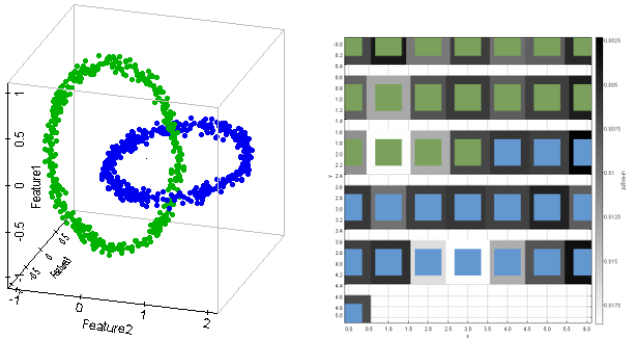


Figura 57 - E-SOM applicato al dataset Chainlink (a destra la relativa U-Matrix)

9.3.6 E-TREE

Anche l'E-TREE si mostra in grado di riconoscere dataset non linearmente divisibili.

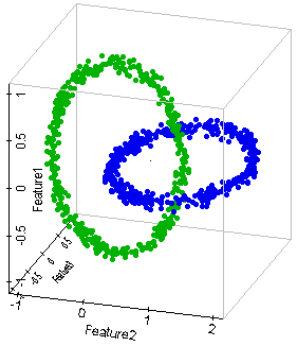


Figura 58 - E-TREE applicato al dataset Chainlink

9.3.7 Confronto

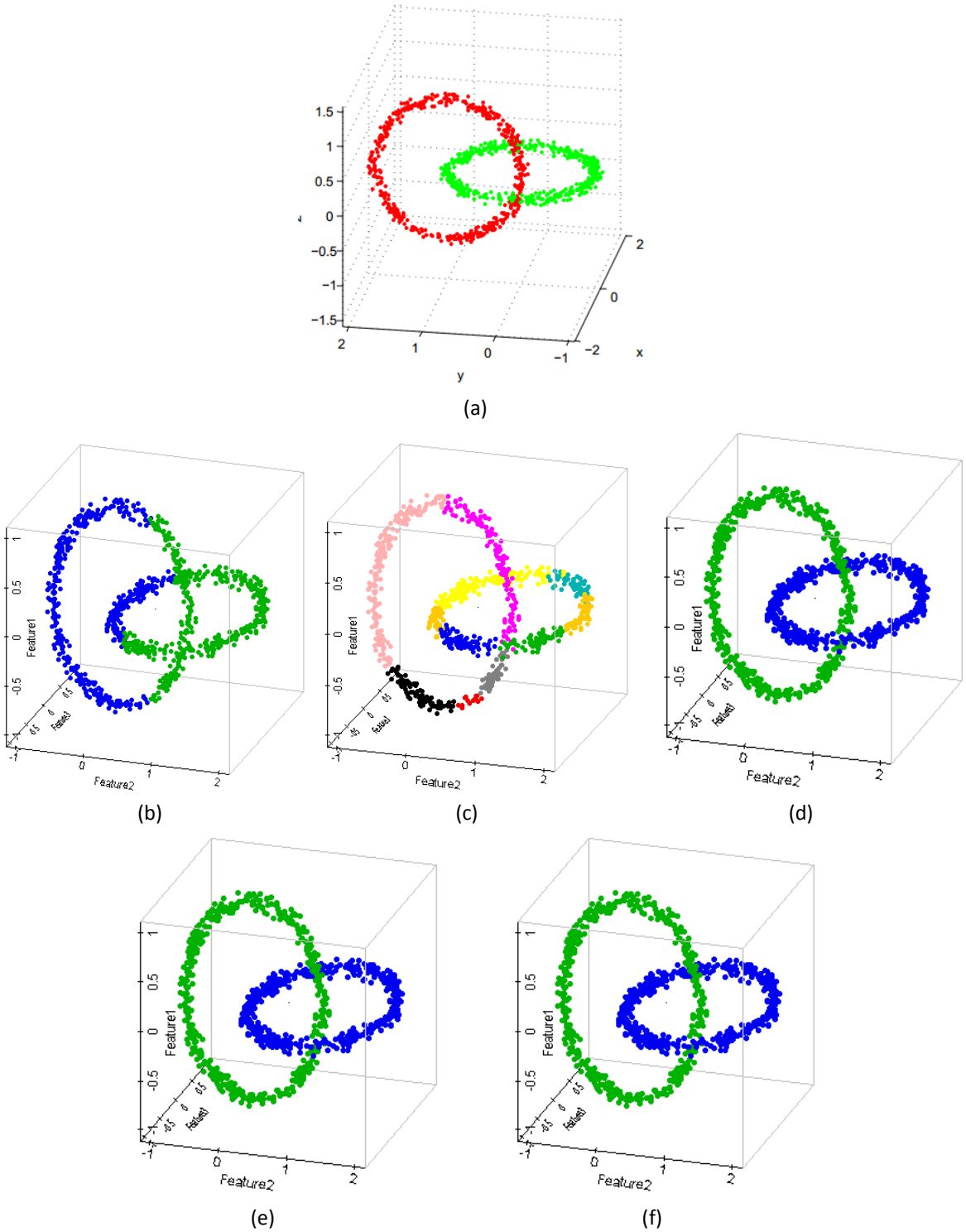


Figura 59 - Confronto del clustering del dataset Chainlink; (a) teorico; (b) SOM+K-Means; (c) SOM+Umat-CC; (d) SOM+TWL; (e) E-SOM; (f) E-TREE

Per la quantificazione numerica dei risultati di clustering ottenuti, si è proceduto a calcolare gli indici statistici di valutazione dell'errore di clustering per tutti i modelli.

CONFRONTO STATISTICO SU PRESTAZIONI DI CLUSTERING (Chainlink)						
MODELLO	E-TREE	SOM	SOM+K-means	SOM+UmatCC	SOM+TWL	E-SOM
errore di quantizzazione	-	0.05	0.05	0.05	0.05	0.12
errore topografico	-	0.28	0.28	0.28	0.28	-
indice di Davies-Bouldin	-	-	1.15	0.68	2.02	1.99
Accuratezza	0		0	0.69	0	0
Completezza	0		1	0	0	0

Tabella 30 – Confronto di prestazioni di clustering tra i vari modelli proposti

9.4 TARGET

Proponiamo in questo paragrafo un dataset ancora una volta estratto dalla FCPS. Si tratta di un dataset bidimensionale composto da 770 pattern. Come è possibile vedere in Figura 60, il dataset presenta due problemi principali. Il primo è quello già esposto al paragrafo 9.3, in merito al dataset Chainlink. I cluster non sono infatti linearmente divisibili. In aggiunta a questo sono presenti 12 outlier, divisi in gruppi di 3, disposti agli angoli dell'immagine.

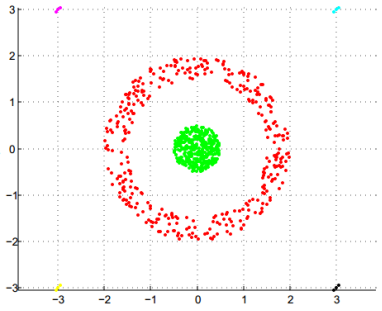


Figura 60 - Clustering teorico del dataset Target

9.4.1 SOM Single-Stage

Dalla U-Matrix ottenuta è possibile identificare i due cluster e i quattro gruppi di outlier in maniera abbastanza evidente. Notiamo infatti l'esistenza di un agglomerato centrale di nodi molti vicini tra loro, separato dagli altri, da una fascia di nodi con distanza media maggiore. Agli angoli della mappa sono posizionati quattro nodi, risultati BMU, il cui gradiente risulta particolarmente scuro, segno di grandi distanze dagli altri nodi, ovvero bassa densità. Confrontando tale analisi con la Figura 61 ci si rende conto che la disposizione dei neuroni rispetta la topologia del dataset.

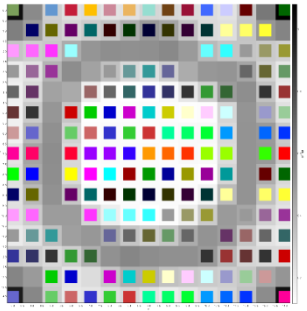


Figura 61 - U-Matrix del modello SOM Single Stage applicata al dataset Target

9.4.2 SOM + K-Means

L'incidenza di un dataset non linearmente divisibile su un algoritmo di clustering partizionale come il K-Means, era già stata messa in luce durante il test sul dataset Chainlink. Tuttavia quest'ulteriore test, non solo conferma quanto anticipato nel suddetto paragrafo, ma mostra l'influenza che gli outlier hanno sul processo di clustering, poiché essi non vengono ne riconosciuti come tali, ne ignorati.

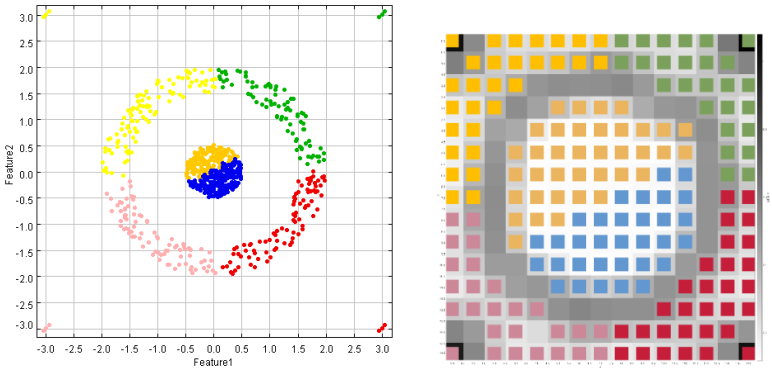


Figura 62 - SOM + K-Means applicato al dataset Target

9.4.3 SOM + Umat-CC

Come si può notare l'Umat-CC classifica perfettamente il cluster centrale. Tuttavia, come avvenuto nel clustering del dataset Chainlink, suddivide il secondo anello in più sotto-cluster. La scarsa robustezza del metodo rispetto agli outlier, era già intuitivamente ipotizzabile ricordando il procedimento dell'Umat-CC. Un nodo della mappa infatti, verrà sempre unito in un cluster con il suo vicino con gradiente più basso. In virtù delle considerazioni, fatte all'inizio del paragrafo, sull'analisi della U-Matrix, un nodo che individua un outlier sarà quasi certamente più scuro di tutti i suoi vicini ed esisterà quindi un percorso che lo colleghi ad una zona più densa di nodi. Il test mostrato conferma quanto appena detto.

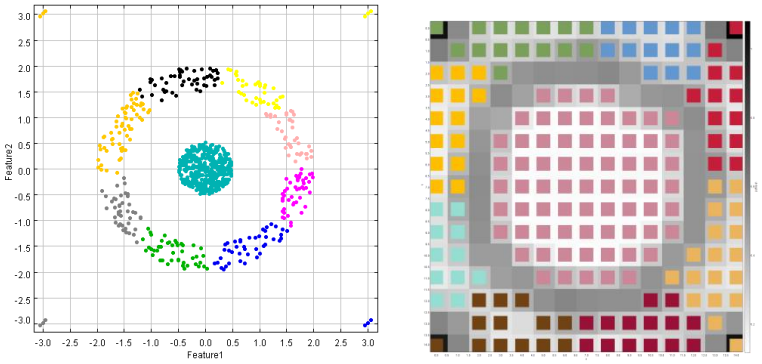


Figura 63 - SOM + Umat-CC applicato al dataset Target

9.4.4 SOM + TWL

Anche in questo caso il test mostra la superiorità del metodo SOM + TWL. Il riconoscimento dei due cluster centrali infatti avviene in maniera perfetta e il controllo sulle connessioni tra nodi troppo lontani, descritto nel paragrafo dedicato alla descrizione del modello, consente l'isolamento dei quattro gruppi di outlier.

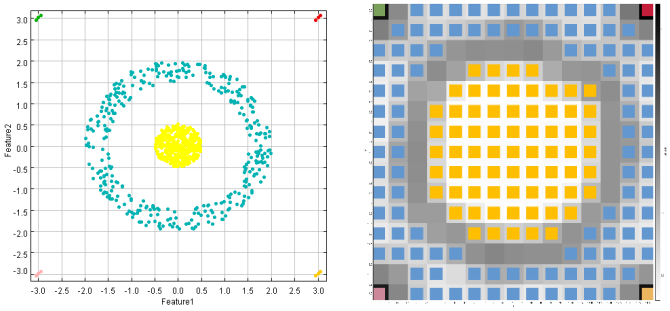


Figura 64 - SOM + TWL applicato al dataset Target (a destra la relativa U-Matrix)

9.4.5 E-SOM

Anche l'E-SOM è in grado di riconoscere sia il cluster centrale che l'anello esterno nella sua totalità. E' tuttavia interessante notare il riconoscimento di uno solo dei gruppi di outlier. Ciò può far intuire che il modello E-SOM è in teoria in grado di riconoscerli tramite il processo di taglio delle connessioni deboli. Tuttavia, il modo stocastico in cui quest'ultimo avviene, basandosi sui parametri immessi dall'utente ed essendo esso influenzato anche dall'ordine in cui i pattern sono presentati, può portare ai risultati mostrati.

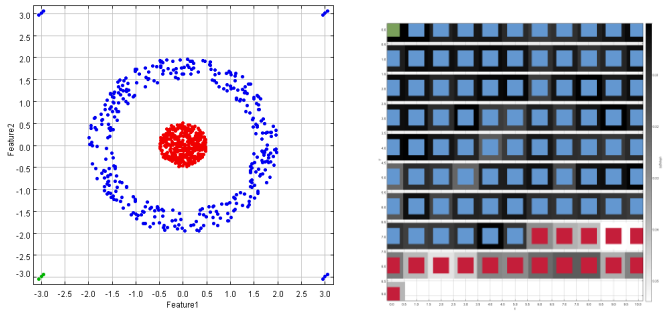


Figura 65 - E-SOM applicato al dataset Target

9.4.6 E-TREE

L'E-TREE si dimostra anche in questo in grado di superare tutte le difficoltà presentate nel dataset, riconoscendo perfettamente si i due cluster che gli outlier.

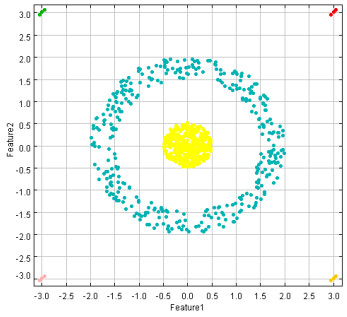


Figura 66 - E-TREE applicato al dataset Target

9.4.7 Confronto

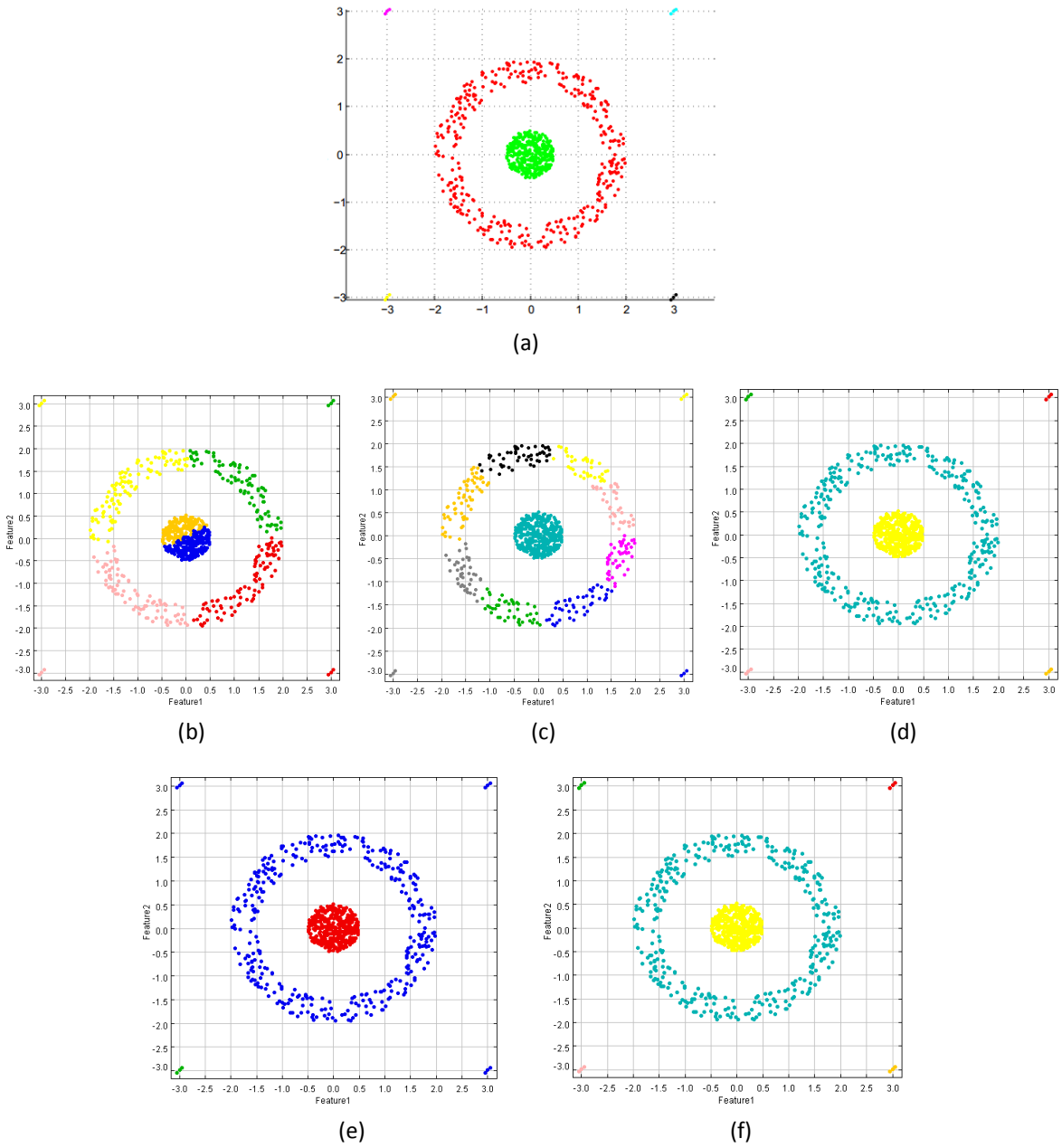


Figura 67 - Confronto del clustering del dataset Target; (a) teorico; (b) SOM+K-Means; (c) SOM+Umat-CC; (d) SOM+TWL; (e) E-SOM; (f) E-TREE

Di seguito gli indici statistici di valutazione dell'errore di clustering per tutti i modelli.

CONFRONTO STATISTICO SU PRESTAZIONI DI CLUSTERING (Target)						
MODELLO	E-TREE	SOM	SOM+K-means	SOM+UmatCC	SOM+TWL	E-SOM
errore di quantizzazione	-	0.07	0.07	0.07	0.07	0.09
errore topografico	-	0.07	0.07	0.07	0.07	-
indice di Davies-Bouldin	-	-	0.86	0.72	12.51	12.09
Accuratezza	0		0	0.2	0	0.33
Completezza	0		1	0.48	0	0.34

Tabella 31 – Confronto di prestazioni di clustering tra i vari modelli proposti

9.5 GOLFBALL

L'ultimo dataset estratto dalla FCPS è un dataset composto da 4002 pattern tridimensionali con la caratteristica di appartenenti tutti ad un unico cluster.

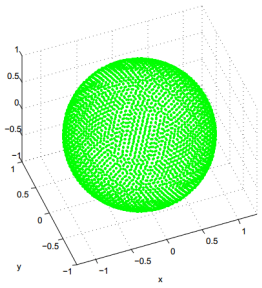


Figura 68 - Clustering teorico del dataset Golfball

9.5.1 SOM Single-Stage

La U-Matrix mostrata non evidenzia particolari zone identificabili come cluster. Ciò significa, quindi, che i nodi si sono disposti sulla superficie della sfera indicata dal dataset in maniera più o meno uniforme. Data la particolarità del test non sarà effettuato un post-processing basato sul K-Means che chiaramente, con un solo cluster atteso, riporterebbe i risultati desiderati.

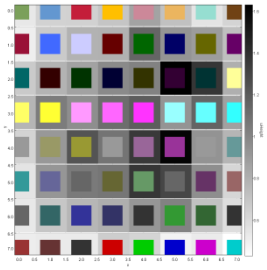


Figura 69 - U-Matrix della SOM Single-Stage applicata al dataset Golfball

9.5.2 SOM + Umat-CC

Poiché i nodi non si dispongono in maniera perfettamente uniforme, ci sono piccoli agglomerati che influenzano il processo di clusterizzazione. Si noti però, un particolare limite dell'Umat-CC. Se anche i nodi si fossero disposti in maniera uniforme sulla sfera, equidistanti l'uno dall'altro, la mappa mostrerebbe tutti i nodi con lo stesso gradiente di grigio. Ciò vuol dire che l'Umat-CC non avrebbe trovato, per nessuno dei nodi, un percorso lungo il gradiente più basso, portando al riconoscimento di ogni nodo come nodo interno alla CC e quindi rappresentante di un cluster.

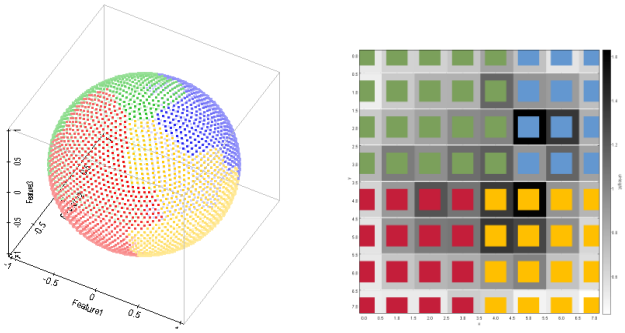


Figura 70 - SOM + Umat-CC applicato al dataset Golfball (a destra la relativa U-Matrix)

9.5.3 SOM + TWL

Anche nel riconoscimento di questa sfera, il metodo TWL come post-processing della SOM si dimostra migliore di quelli visti, poiché solo una piccolissima percentuale di pattern viene classificata in un cluster separato. Questo comportamento era prevedibile, ricordando che il metodo TWL tende a sperare i nodi esterni. Poiché su una U-Matrix esisterà sempre almeno un nodo esterno, il TWL è difficilmente in grado di risconoscere dataset in cui sia presente un solo cluster.

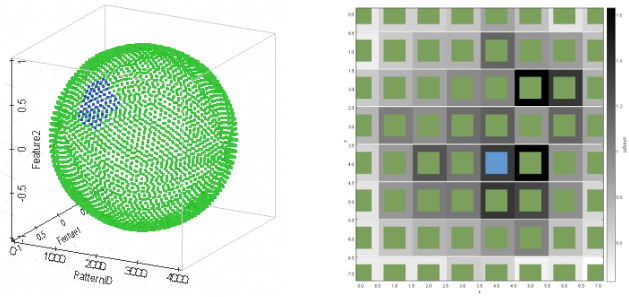


Figura 71 - SOM + TWL applicato al dataset Golfball (a destra la relativa U-Matrix)

9.5.4 E-SOM

Anche il modello E-SOM, come si può notare, riconosce l'unico cluster presente all'interno del dataset.

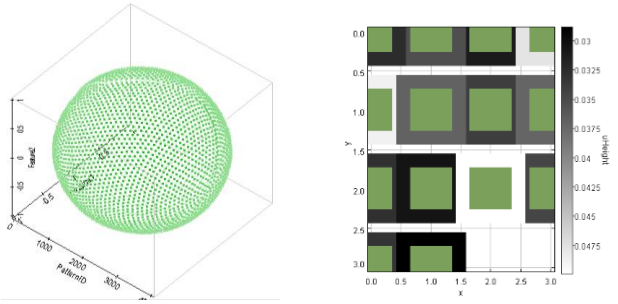


Figura 72 - E-SOM applicato al dataset Golfball (a destra la relativa U-Matrix)

9.5.5 E-TREE

L'immagine mostra il corretto riconoscimento del cluster da parte del modello E-TREE.

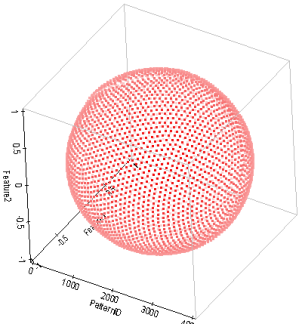
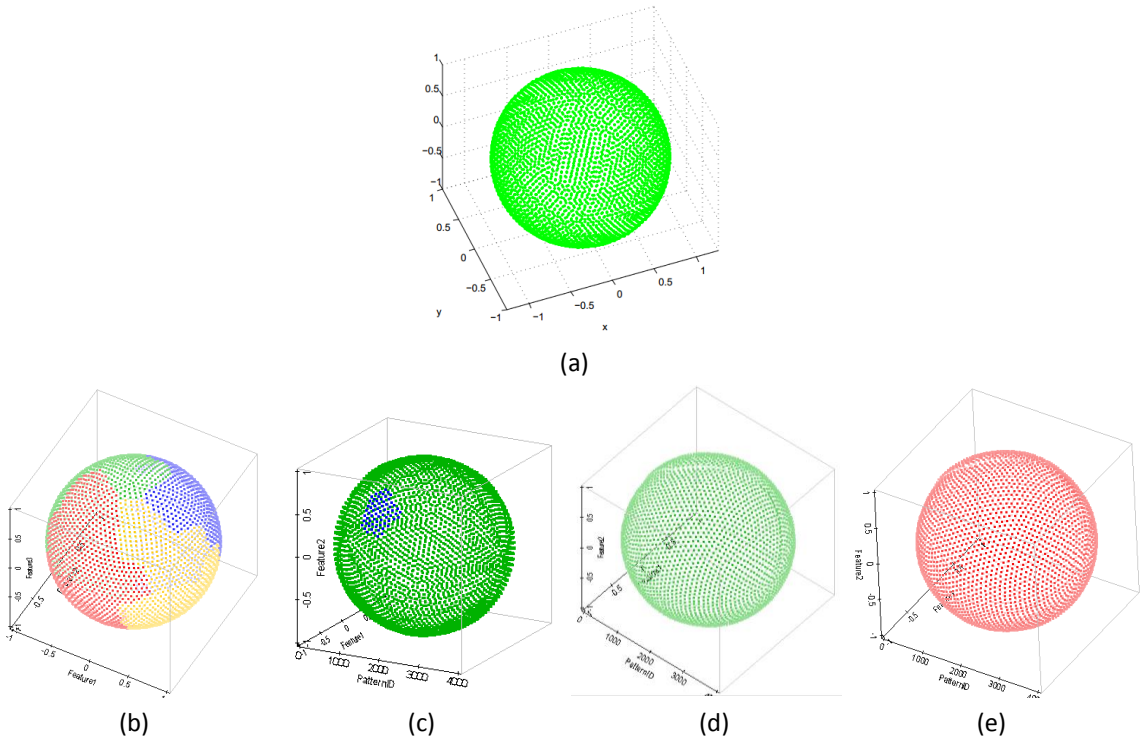


Figura 73 - E-TREE applicato al dataset Golfball

9.5.6 Confronto



(a)

(b) (c) (d) (e)

Figura 74 - Confronto clustering dataset Golfball; (a) teorico; (b) SOM+Umat-CC; (c) SOM+TWL; (d) E-SOM; (e) E-TREE

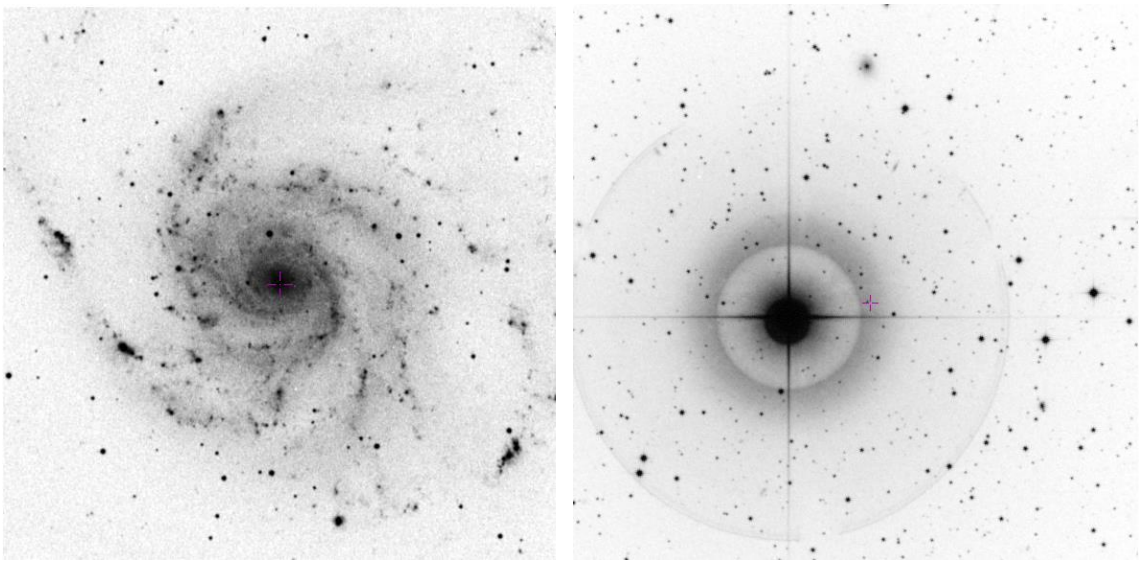
Di seguito gli indici statistici di valutazione dell'errore di clustering per tutti i modelli.

CONFRONTO STATISTICO SU PRESTAZIONI DI CLUSTERING (Golfball)						
MODELLO	E-TREE	SOM	SOM+K-means	SOM+UmatCC	SOM+TWL	E-SOM
errore di quantizzazione	-	0.17	-	0.17	0.17	1.0
errore topografico	-	0.76	-	0.76	0.76	-
indice di Davies-Bouldin	-	-	-	2.76	1.13	0.0
Accuratezza	0		0	0.6	0.33	0
Completezza	0		0	0	0	0

Tabella 32 – Confronto di prestazioni di clustering tra i vari modelli proposti

9.6 IMMAGINI ASTRONOMICHE MONOCROMATICHE

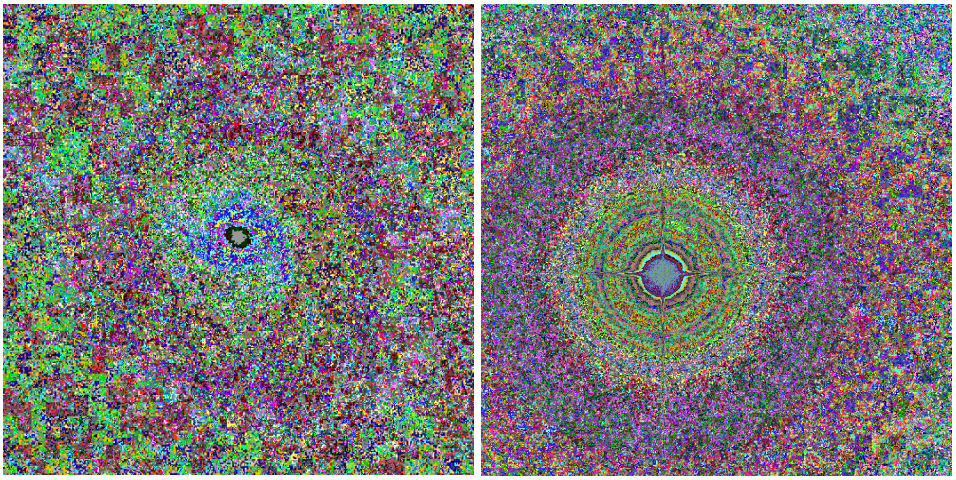
Una sessione di test è stata dedicata al clustering di immagini. Essendo il presente lavoro orientato ad applicazioni di tipo astrofisico, si è scelto di effettuare una serie di test di clustering su immagini astronomiche. Il formato tipicamente utilizzato in questo settore disciplinare è FITS. Notoriamente un'immagine FITS può rappresentare una vista in banda singola (monocromatica) o multi-banda (colori ottenuti come sovrapposizione di immagini monocromatiche). In questa prima sezione abbiamo utilizzato due immagini monocromatiche, rispettivamente una galassia (M101) ed una regione di osservazione POSSII influenzata dalla presenza di una stella in risalto rispetto agli altri oggetti lontani. M101 è un'immagine FITS 530x530 pixel, mentre POSSII-XP559 ha dimensioni 891x893 pixel. Data la complessità del dataset sottoposto, i dati sono sottoposti alla procedura di normalizzazione.



(a) (b)

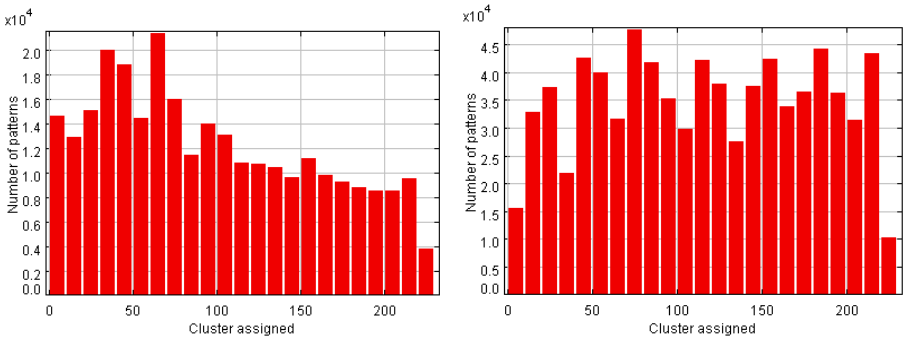
Figura 75 – Immagini mono-banda utilizzate per i test di clustering; (a) galassia M101, (b) regione POSSII-XP559

9.6.1 SOM Single-Stage



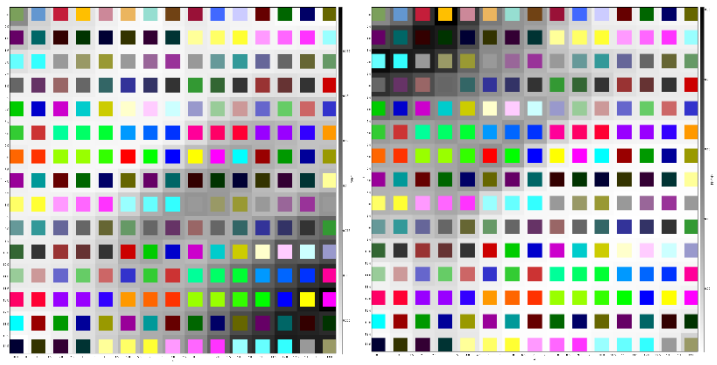
(a) (b)

Figura 76 – Immagine output del training della SOM Single-Stage; (a) galassia M101, (b) regione POSSII-XP559



(a) (b)

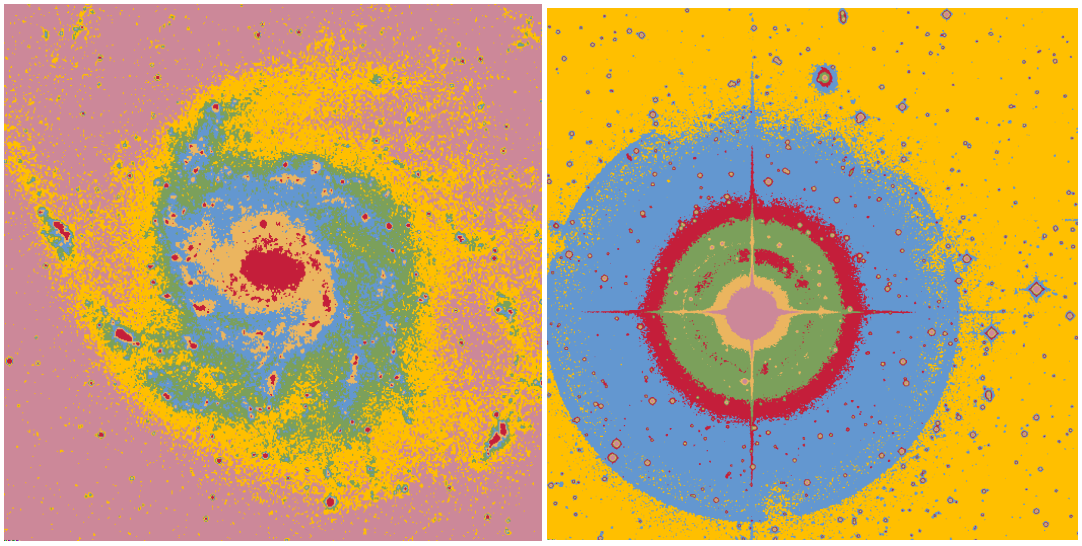
Figura 77 – Istogramma distribuzione cluster output training di SOM Single-Stage; (a) galassia M101, (b) regione POSSII-XP559



(a) (b)

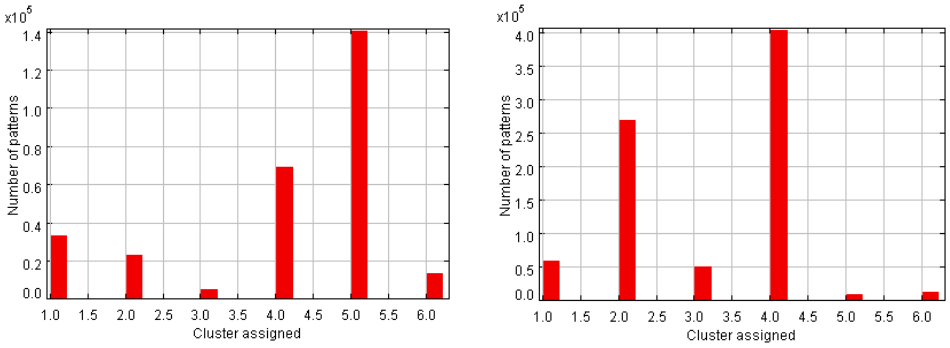
Figura 78 - U-Matrix della SOM Single-Stage; (a) galassia M101, (b) regione POSSII-XP559

9.6.2 SOM + K-Means



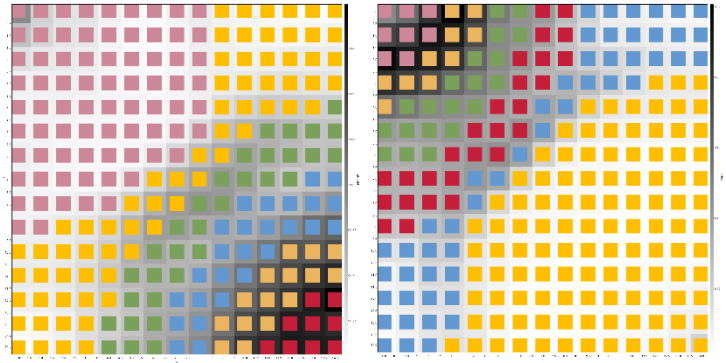
(a) (b)

Figura 79 – Immagine output del training della SOM+K-means; (a) galassia M101, (b) regione POSSII-XP559



(a) (b)

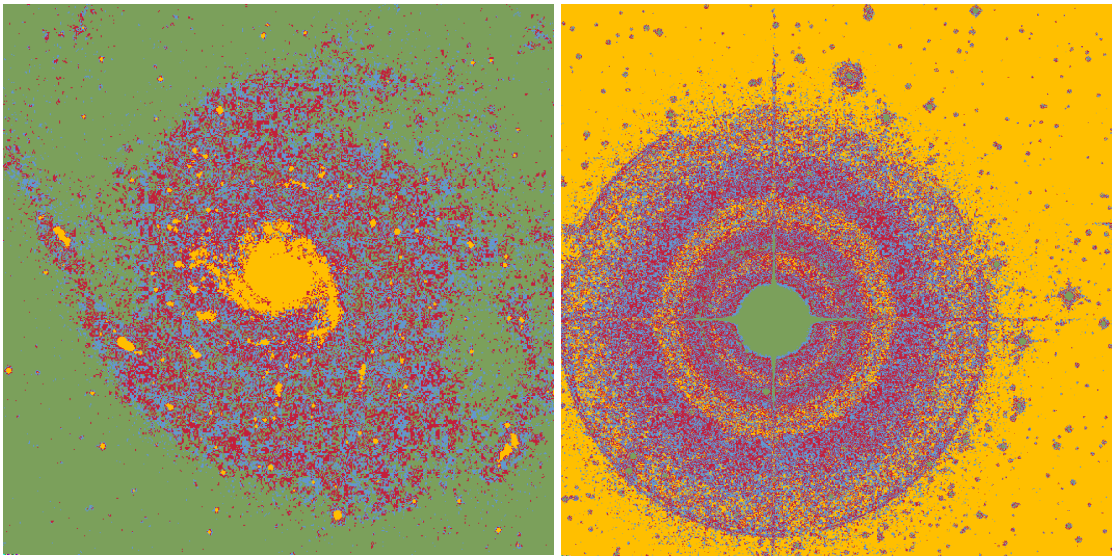
Figura 80 – Istogramma distribuzione cluster output del training di SOM+K-means; (a) galassia M101, (b) regione POSSII-XP559



(a) (b)

Figura 81 - U-Matrix della SOM+K-means; (a) galassia M101, (b) regione POSSII-XP559

9.6.3 SOM + Umat-CC



(a) (b)

Figura 82 – Immagine output del training della SOM+UmatCC; (a) galassia M101, (b) regione POSSII-XP559

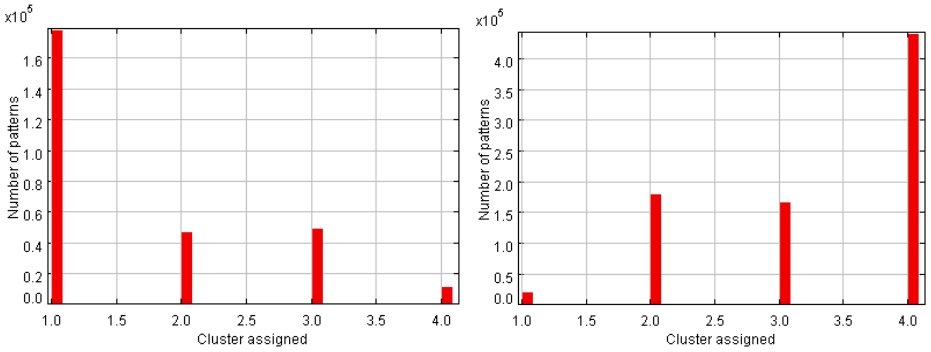


Figura 83 – Istogramma distribuzione cluster output training di SOM+UmatCC; (a) galassia M101, (b) regione POSSII-XP559

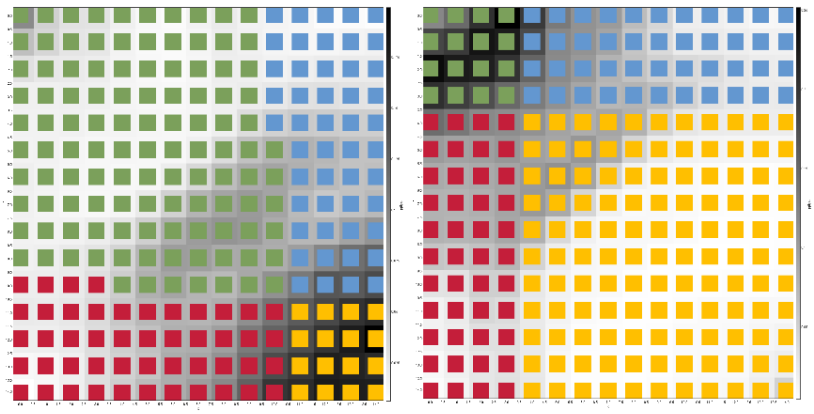


Figura 84 - U-Matrix della SOM+UmatCC; (a) galassia M101, (b) regione POSSII-XP559

9.6.4 SOM + TWL

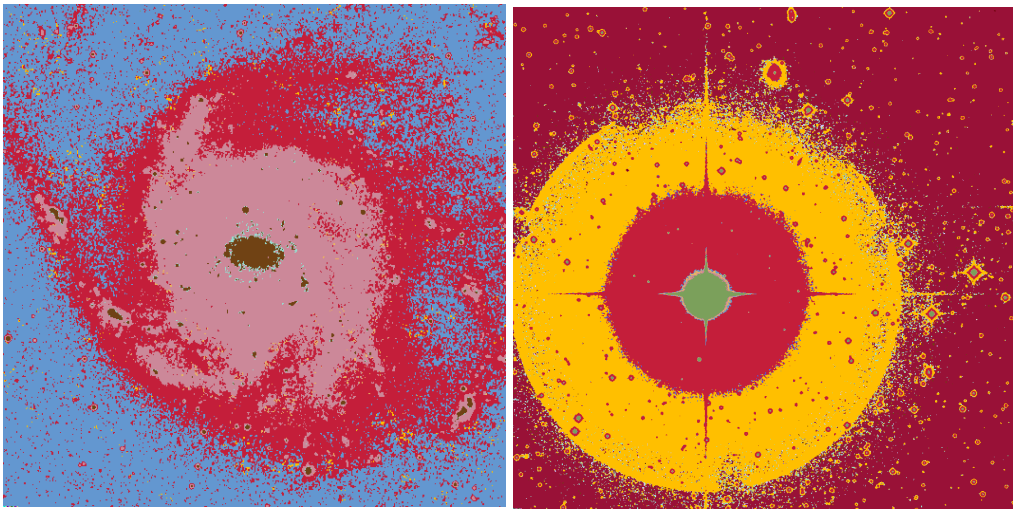


Figura 85 – Immagine output del training della SOM+TWL; (a) galassia M101, (b) regione POSSII-XP559

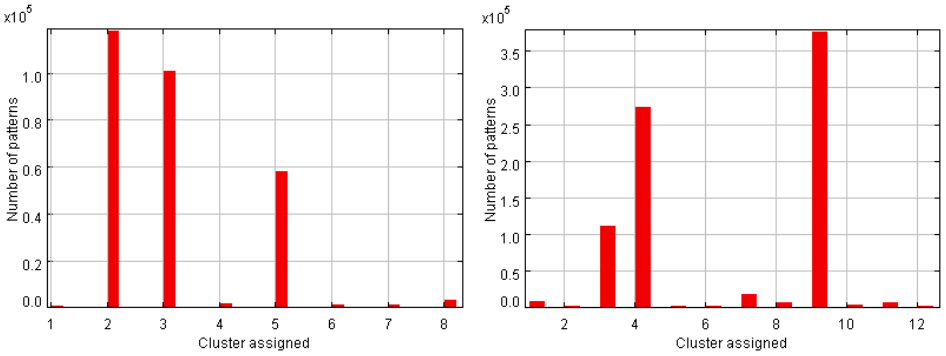


Figura 86 – Istogramma distribuzione cluster output del training di SOM+TWL; (a) galassia M101, (b) regione POSSII-XP559

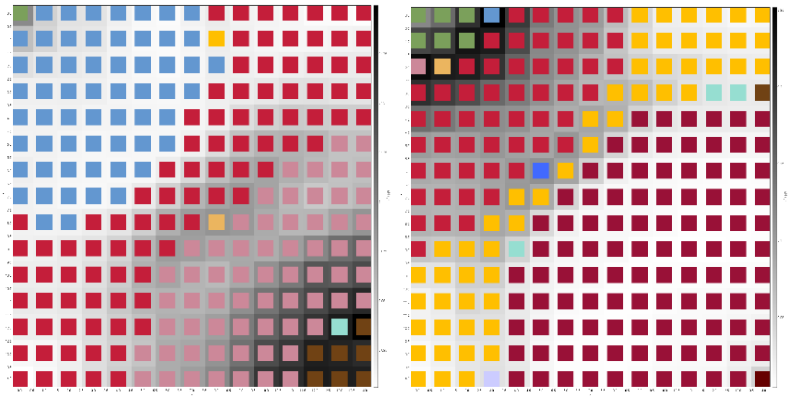
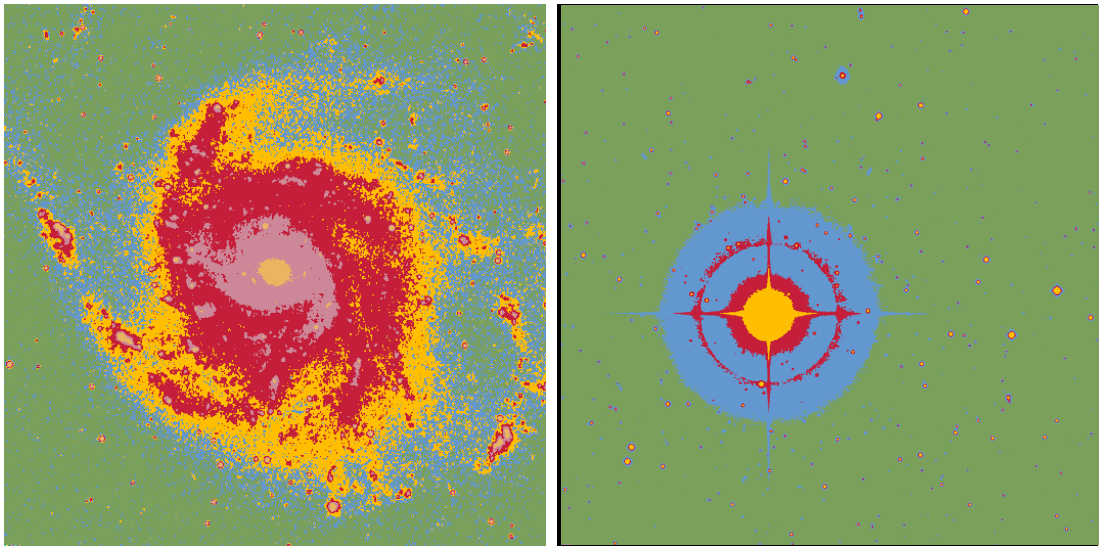


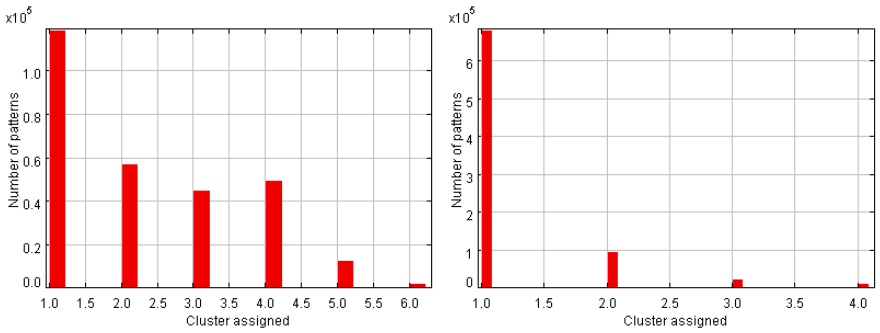
Figura 87 - U-Matrix della SOM+TWL; (a) galassia M101, (b) regione POSSII-XP559

9.6.5 E-SOM



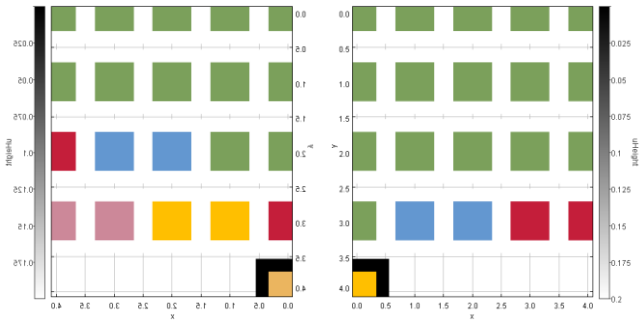
(a) (b)

Figura 88 – Immagine output del training di E-SOM; (a) galassia M101, (b) regione POSSII-XP559



(a) (b)

Figura 89 – Istogramma distribuzione cluster output del training di E-SOM; (a) galassia M101, (b) regione POSSII-XP559



(a) (b)

Figura 90 - U-Matrix della E-SOM; (a) galassia M101, (b) regione POSSII-XP559

9.6.6 Confronto

L'applicazione della SOM Single-Stage produce, come atteso, un eccessivo numero di raggruppamenti. Nell'immagine M101, le variazioni dinamiche del background sono indotte dagli aloni dei rami della spirale mentre, nell'immagine POSSII-XP559, l'alone indotto dalla stella in risalto condiziona fortemente il gradiente di fondo cosmico. I test evidenziano una certa similarità dei metodi SOM + K-Means ed E-SOM, similarità maggiormente accentuata nel test sull'immagine M101 nel quale il modello E-SOM individua lo stesso numero di cluster del K-Means, pur non avendo a priori alcuna conoscenza sul numero degli stessi. Nell'immagine POSSII-XP559 invece, i risultati si differenziano nell'alone esterno della stella centrale, che viene assimilata, nel modello E-SOM, allo sfondo, senza compromettere il riconoscimento degli oggetti sovrapposti all'alone stesso. Il modello TWL, se si escludono i piccoli cluster individuati da nodi separatori di classe o possibili outliers (si veda il paragrafo 5.4.3), individua un numero inferiore di raggruppamenti rispetto ai metodi sopraccitati, inglobando in cluster unici alcuni degli aloni che gli oggetti celesti presentano. Il comportamento dell'Umat-CC è per certi versi paragonabile al TWL ma, per sua natura, non è in grado di lavorare correttamente su una U-Matrix generata da una così particolare configurazione dei nodi della rete. Il metodo infatti riconosce i nuclei degli oggetti e gran parte dello sfondo, confondendo in maniera evidente però gli aloni degli stessi. Quanto appena detto trova riscontro nella valutazione di clustering ottenuta tramite l'indice DB.

9.6.6.1 M101

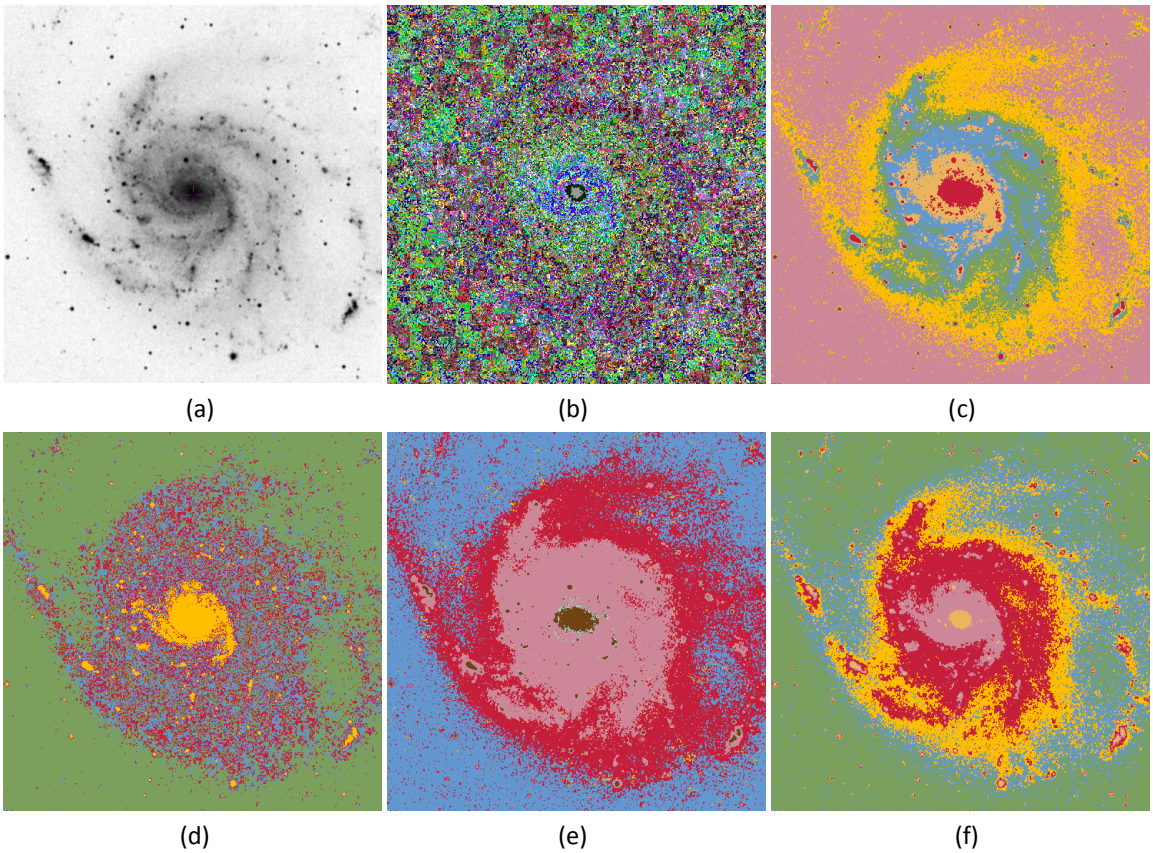


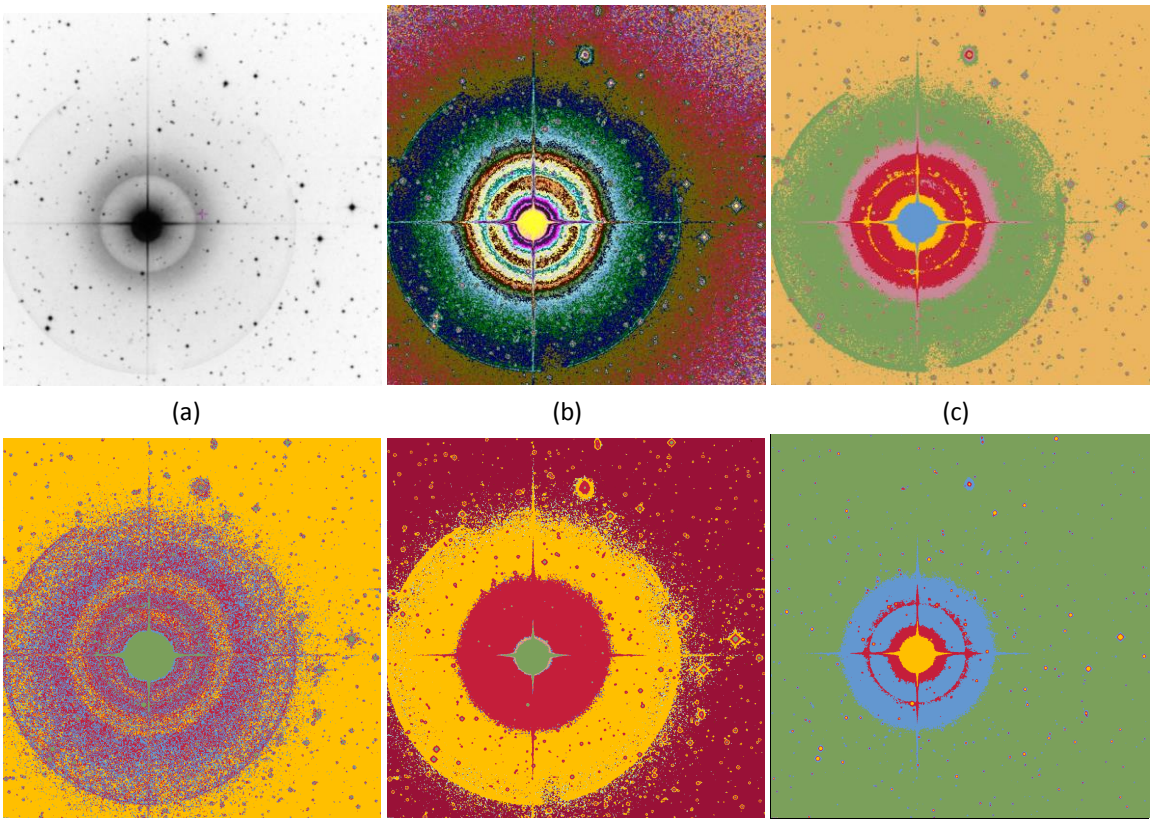
Figura 91 - Confronto clustering immagine M101 – (a) originale, (b) SOM, (c) SOM+K-means; (d) SOM+Umat-CC; (e) SOM+TWL; (f) E-SOM

Di seguito sono riportati gli indici staticisti dei test eseguiti. Naturalmente, non avendo a disposizione informazioni a priori sui cluster attesi per le due immagini, i due indici ICA e ICC non sono disponibili.

CONFRONTO STATISTICO SU PRESTAZIONI DI CLUSTERING (Img M101)					
MODELLO	SOM	SOM+K-means	SOM+UmatCC	SOM+TWL	E-SOM
errore di quantizzazione	0.001	0.001	0.001	0.001	0.03
errore topografico	0.90	0.90	0.90	0.90	-
indice di Davies-Bouldin	-	0.53	6.41	0.55	0.60

Tabella 33 – Confronto di prestazioni di clustering tra i vari modelli proposti per M101

9.6.6.2 POSSII-XP559



(a) (b) (c)

(d) (e) (f)

Figura 92 - Confronto clustering immagine POSSII – (a) originale, (b) SOM, (c) SOM+K-means; (d) SOM+Umat-CC; (e) SOM+TWL; (f) E-SOM

CONFRONTO STATISTICO SU PRESTAZIONI DI CLUSTERING (Img POSSII)					
MODELLO	SOM	SOM+K-means	SOM+UmatCC	SOM+TWL	E-SOM
errore di quantizzazione	0.001	0.001	0.001	0.001	0.04
errore topografico	0.91	0.91	0.91	0.91	-
indice di Davies-Bouldin	-	0.48	9.18	0.62	1.1

Tabella 34 – Confronto di prestazioni di clustering tra i vari modelli proposti per POSSII

9.7 IMMAGINI ASTRONOMICHE MULTI-BANDA

Per effettuare i test di clustering su immagini astronomiche multi-banda, si è scelta l'immagine mostrata in Figura 93. Un'immagine multi-banda è un file FITS composto da un insieme di immagini in singola banda dello spettro elettromagnetico, la cui sovrapposizione produce un'immagine a colori (relativi ai tipi di banda usati).

Per il test si è scelto un campo deep field osservato dallo strumento CFHT[1] nelle bande del visibile UGRIZ, in cui ogni singola immagine monocromatica ha dimensioni 700x700 pixel.

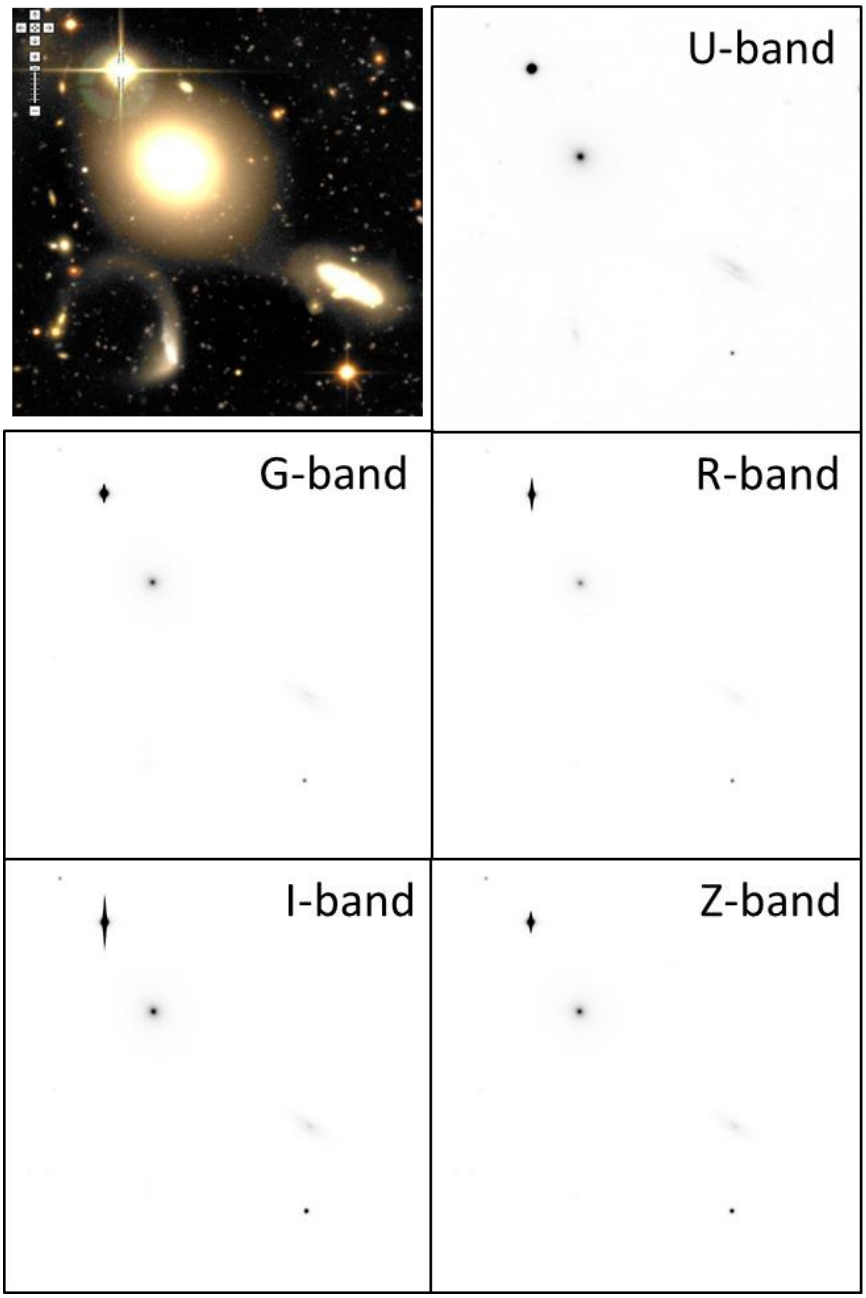


Figura 93 - Immagine multibanda (a colori) di riferimento e le 5 a singola banda per il campo CFHT-D1

[1] http://www3.cadc-ccda.hia-iha.nrc-cnrc.gc.ca/community/CFHTLS-SG/docs/cfhtlsD1.html

9.7.1 SOM Single-Stage

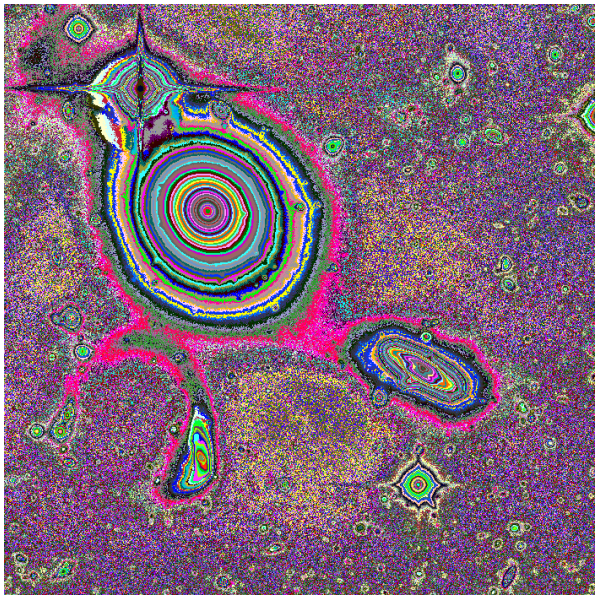


Figura 94 - Immagine di output del training della SOM Single-Stage su CFHT-D1

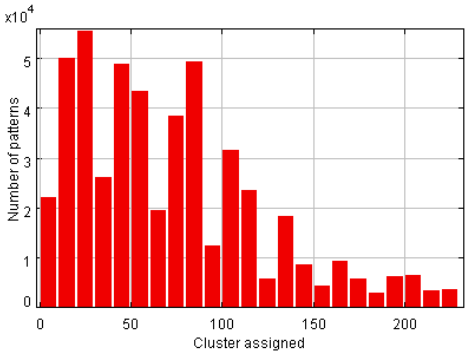


Figura 95 - Istogramma di distribuzione cluster output di training SOM Single-Stage CFHT-D1

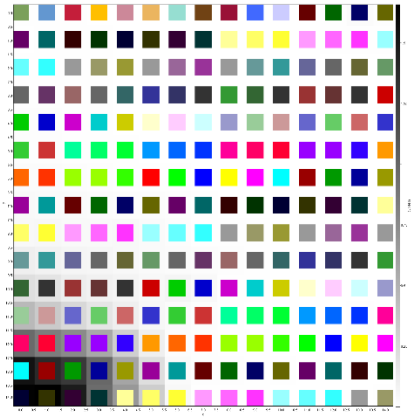


Figura 96 - U-Matrix della SOM Single-Stage su CFHT-D1

9.7.2 SOM + K-Means

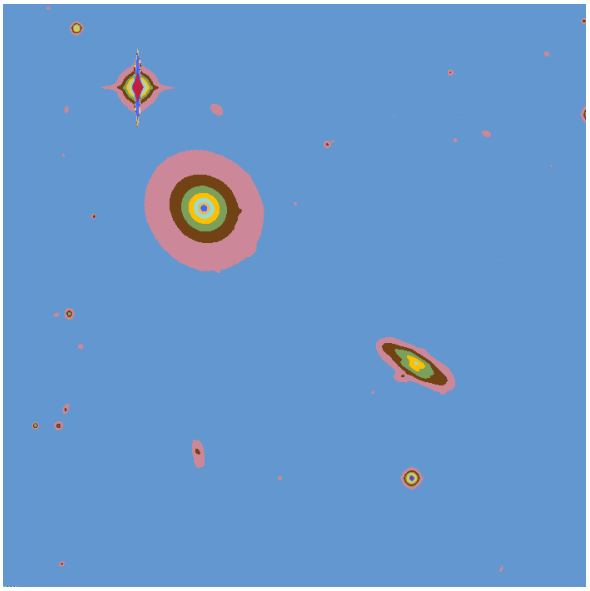


Figura 97 - Immagine di output del training della SOM + K-Means su CFHT-D1

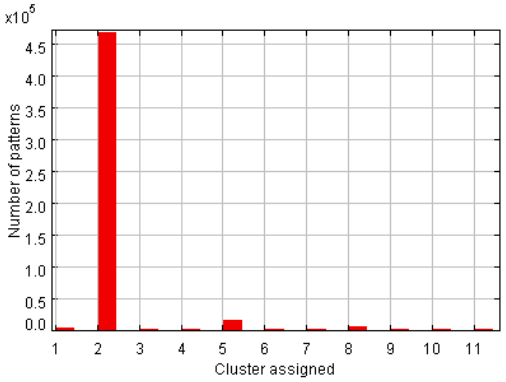


Figura 98 - Istogramma di distribuzione cluster output di training SOM + K-Means CFHT-D1

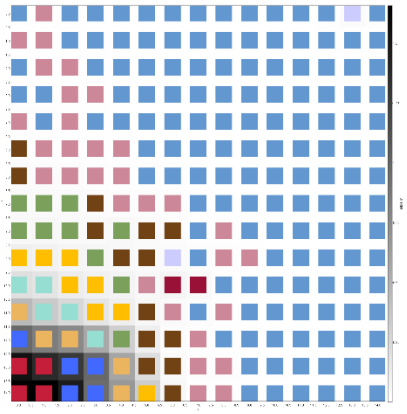


Figura 99 - U-Matrix della SOM + K-Means su CFHT-D1

9.7.3 SOM + Umat-CC

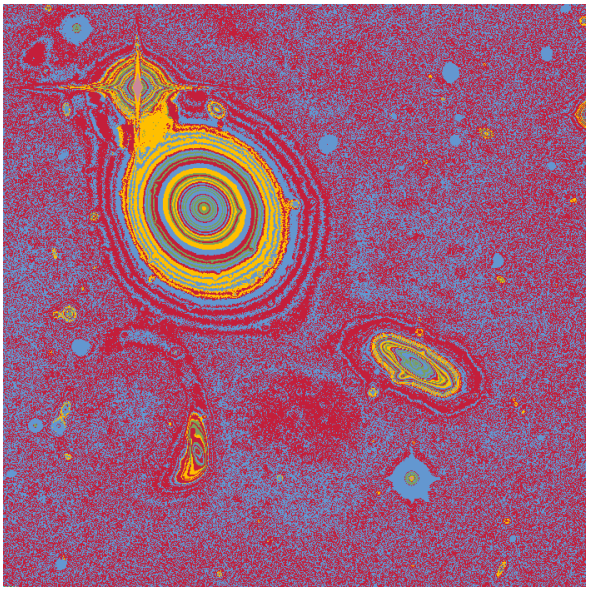


Figura 100 - Immagine di output del training della SOM + Umat-CC su CFHT-D1

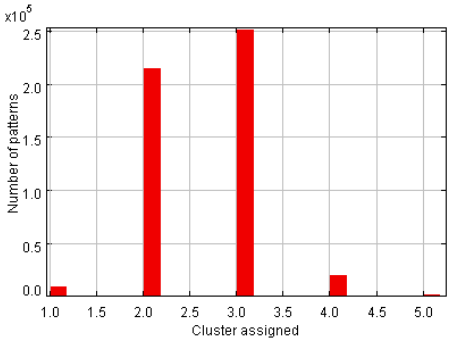


Figura 101 - Istogramma di distribuzione cluster output di training SOM + Umat-CC su CFHT-D1

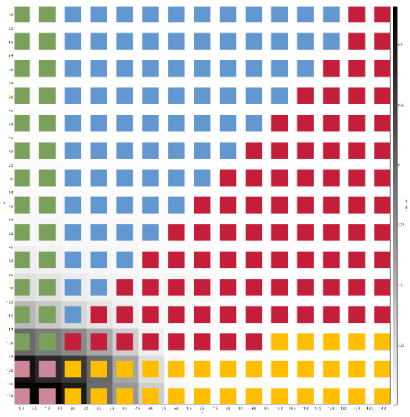


Figura 102 - U-Matrix della SOM + Umat-CC su CFHT-D1

9.7.4 SOM + TWL

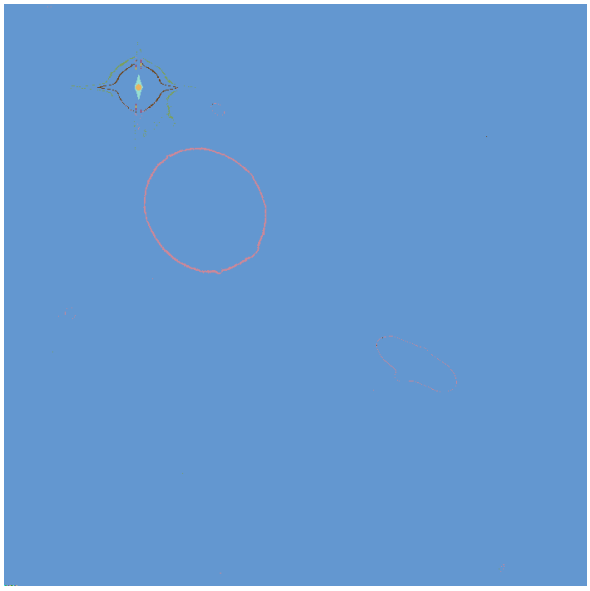


Figura 103 - Immagine di output del training di SOM + TWL su CFHT-D1

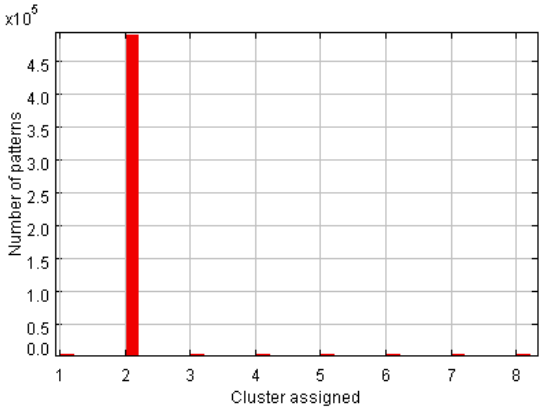


Figura 104 - Istogramma di distribuzione cluster output di training di SOM + TWL su CFHT-D1

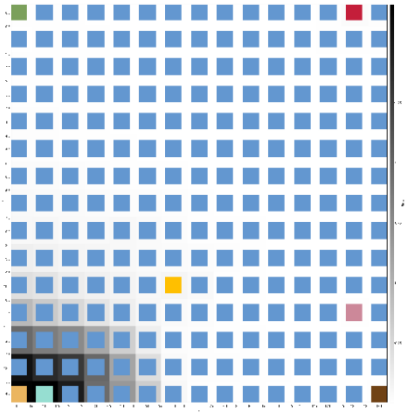


Figura 105 - U-Matrix della SOM + TWL su CFHT-D1

9.7.5 E-SOM

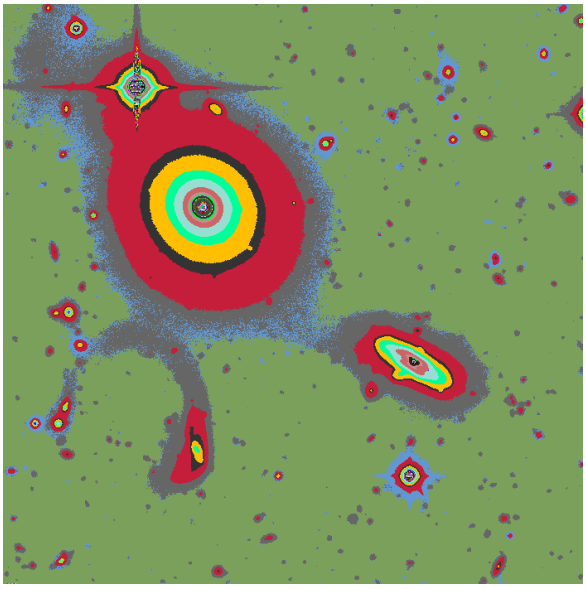


Figura 106 - Immagine di output del training dell'E-SOM su CFHT-D1

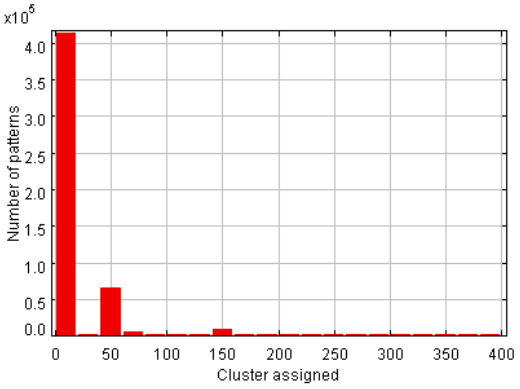


Figura 107 - Istogramma di distribuzione cluster output di training dell'E-SOM su CFHT-D1

Figura 108 - U-Matrix dell'E-SOM su CFHT-D1

9.7.6 Confronto

La Figura 109 mostra il confronto diretto tra i vari metodi di clustering sull'immagine multi-banda. Nonostante i riferimenti statistici mostrati in Figura 109, il migliore risultato è palesemente ottenuto dal modello E-SOM. Il valore del parametro DB tuttavia, indica che vi è una sovrastima del numero di cluster identificati. Ciò infatti deriva dal notevole numero di raggruppamenti evidenziati all'interno di alcuni degli oggetti celesti (Figura 110). Il modello SOM+Umat-CC rivela una parziale confusione di aloni e background, sebbene però sia l'unica, insieme all' E-SOM, ad individuare una variazione significativa del fondo in zone peculiari. Da notare inoltre lo scarso risultato da parte del modello SOM+TWL, probabilmente dovuto all'eccessiva complessità del dataset.

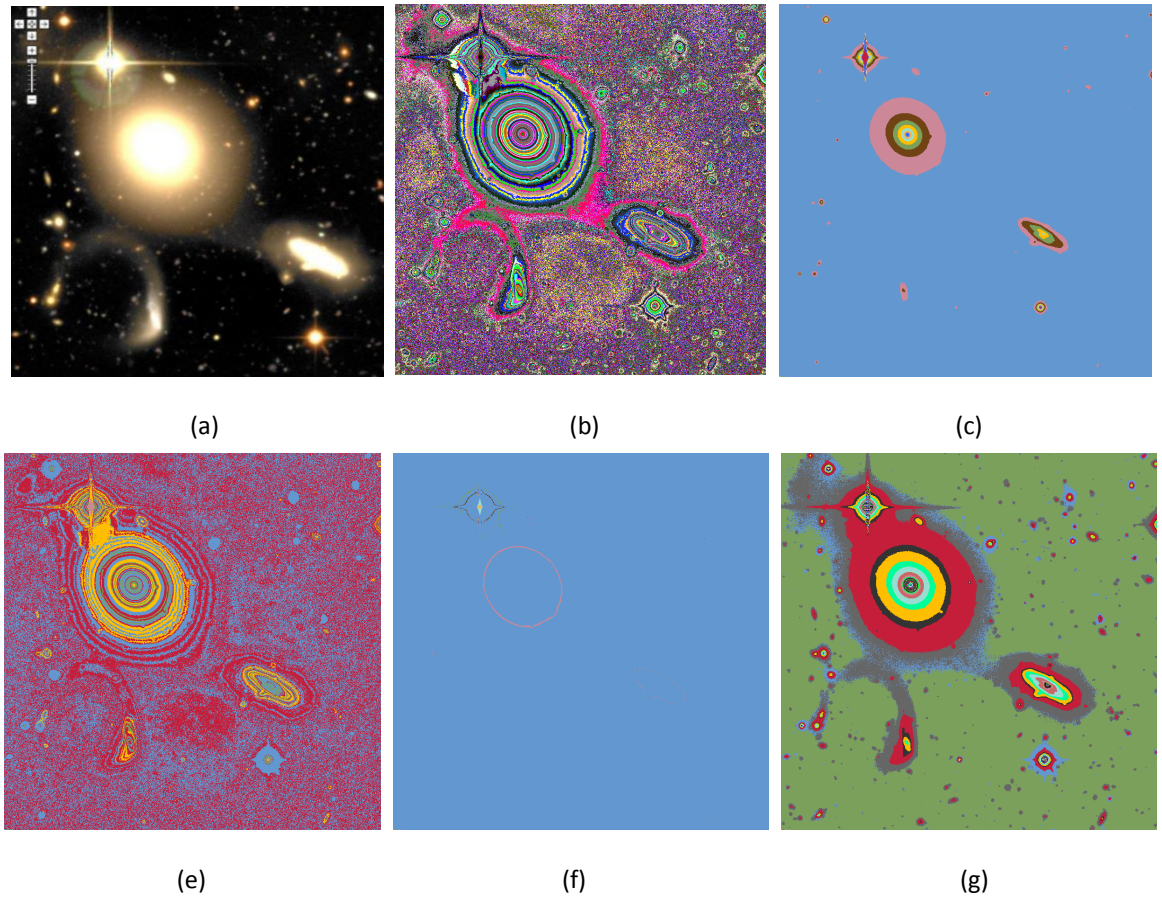


Figura 109 – Confronto di clustering sull'immagine multi-banda CFHT-D1: (a) originale; (b) SOM single-stage; (c) SOM+K-means; (d) SOM+Umat-CC; (e) SOM+TWL; (f) E-SOM

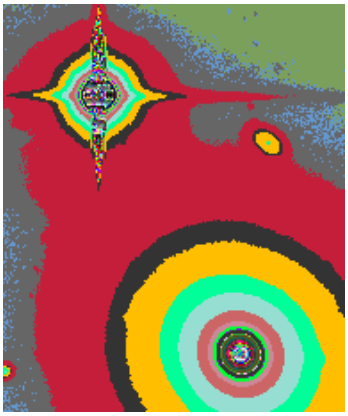


Figura 110 – Particolare dell'immagine di output del clustering relativo al modello E-SOM, in cui sono visibili gli innumerevoli cluster all'interno degli oggetti

CONFRONTO STATISTICO SU PRESTAZIONI DI CLUSTERING (Img D1)					
MODELLO	SOM	SOM+K-means	SOM+UmatCC	SOM+TWL	E-SOM
errore di quantizzazione	0.0005	0.0005	0.0005	0.0005	0.0007
errore topografico	0.88	0.88	0.88	0.88	-
indice di Davies-Bouldin	-	0.87	7.93	0.91	1.28

Tabella 35 – Confronto di prestazioni di clustering tra i vari modelli proposti per CFHT-D1

9.8 CONFRONTO CON MODELLI ESTERNI

La validazione della scelta dei metodi progettati e sviluppati in questo lavoro è stata compiuta selezionando opportunamente altri modelli esterni, fruibili in modo comune da internet, con cui confrontare direttamente i metodi proposti.

In particolare, per il clustering, il modello base più noto e comunemente utilizzato, è il K-means. Questo modello è stato confrontato con i modelli oggetto del presente lavoro (SOM, SOM+K-means, SOM+UmatCC, SOM+TWL, E-SOM).

Mentre per la classificazione si è scelto di confrontare il modello proposto E-TREE con due modelli supervisionati molto noti, nella fattispecie Multi Layer Perceptron con Back Propagation (MLPBP) e Support Vector Machine (SVM).

Nel seguito si descrivono gli esperimenti di confronto ed i relativi risultati.

9.8.1 Confronto di clustering (con K-means)

Per validare i modelli di clustering proposti, si è scelto il modello base K-means (Hartigan & Wong 1979) per il confronto, utilizzando il dataset IRIS come esempio di test.

9.8.1.1 Il dataset Iris

Il dataset Iris è un dataset multivariato introdotto da Fisher (1936) come esempio per spiegare l'analisi discriminante. Il dataset è composto da 50 pattern per ogni specie di Iris *(Setosa, Virginica e Versicolor)* per un totale di 150 pattern. Per ognuno di essi sono state misurate 4 feature: lunghezza e larghezza dei petali e dei sepali.

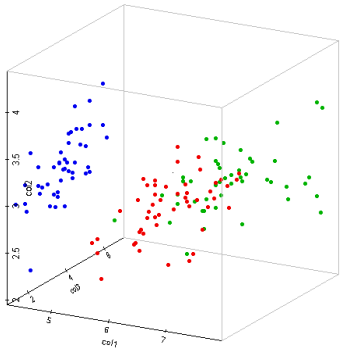


Figura 111 – Separazione in 3 classi delle specie nel dataset Iris

Essendo note a priori le classi di appartenenza di ogni pattern, il dataset è stato ampiamente utilizzato come test per algoritmi di classificazione mentre risulta meno comune per l'analisi dei cluster, in quanto solo due gruppi sono nettamente distinguibili, uno dei quali contiene i pattern relativi alla specie *Iris Setosa* mentre l'altro contiene quelli relativi alle specie *Iris Virginica* e *Iris Versicolor* (Figura 111).

9.8.1.2 Configurazione dei modelli

Di seguito si riporta la lista dei parametri utilizzati per il setup dei vari modelli impiegati per il confronto. Si è volontariamente omessa la lista dei parametri del modello base K-Means, in quanto costituito da un solo parametro richiesto, ossia il numero di cluster attesi, settato a 3.

PARAMETRI SOM	
Numero di nodi input	4
Righe dello strato di kohonen	7
Colonne dello strato di kohonen	7
Numero massimo di iterazioni	200
Learning rate iniziale	0.7
Soglia di learning rate decay	0.005
Diametro iniziale del vicinato	14
Normalizzazione dei dati input	No
PARAMETRI post-processing (K-means)	
Numero di cluster attesi	3
PARAMETRI post-processing (Umat-CC)	
Grado di sfocatura della U-matrix	1.5
PARAMETRI post-processing (TWL)	
Grado di sfocatura della U-matrix	0.5

Tabella 36 – Parametri del modello SOM e post-processing usati per l'esperimento di clustering

PARAMETRI E-SOM	
Numero di nodi input	4
Epsilon (soglia minima di distanza tra pattern e BMU)	0.5
Pruning time (Intervallo di tempo dopo il quale effettuare il taglio della connessione più debole)	10
Learning rate	0.05
Normalizzazione dei dati input	no

Tabella 37 – Parametri del modello E-SOM usati per l'esperimento di clustering

9.8.1.3 Risultati del confronto

L'applicazione di un generico metodo di clustering, quale il **K-Means**, nel caso del dataset Iris con 3 cluster attesi, porta ai seguenti risultati:

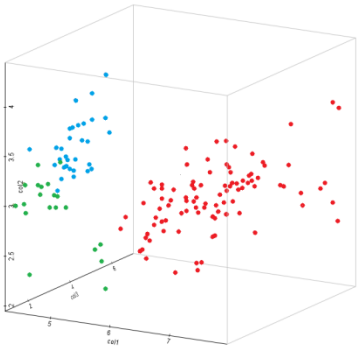


Figura 112 – Risultato prodotto da un generico metodo basato sul K-Means applicato al dataset Iris

Come è possibile notare, confrontando le immagini mostrate, il riconoscimento delle 3 classi non avviene in maniera corretta. I pattern appartenenti alla classe *Iris Setosa* vengono infatti divisi tra due cluster mentre i restanti vengono uniti in un unico raggruppamento.

Riportiamo di seguito i risultati dell'applicazione del modello **SOM** senza l'applicazione di un metodo di clustering a due stadi. Risulta evidente quindi quanto detto nel capitolo 5, ovvero la necessità, ai fini del clustering, dell'utilizzo di un metodo di post-processing che raggruppi i nodi dello strato di Kohonen.

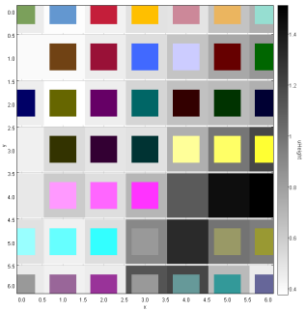


Figura 113 - U-Matrix della SOM Single-Stage applicata al dataset Iris

Abbiamo precedentemente mostrato cosa succede applicando un generico algoritmo partizionale come il K-Means, al dataset Iris. Mostriamo invece adesso cosa accade anteponendo il modello SOM all'esecuzione del K-Means (**SOM+K-means**), mantenendo invariato il parametro indicante il numero di cluster attesi pari a tre. Sono evidenti le elevate percentuali di associazione di ogni pattern alla giusta classe, mostrate nella Tabella 38.

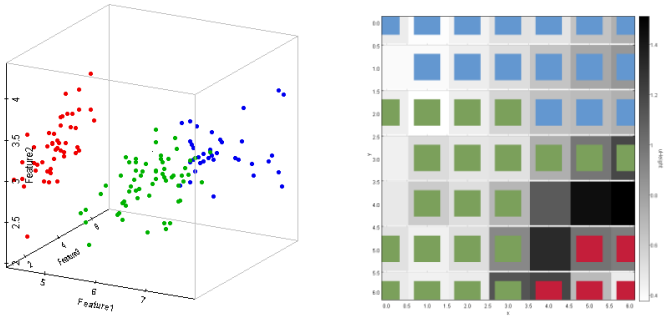


Figura 114 - SOM + K-Means applicato al dataset Iris (a destra la relativa U-Matrix)

CLASSE	trovati	persi	%
Iris Setosa	50	0	100%
Iris Virginica	48	2	96%
Iris Versicolor	36	14	72%

Tabella 38 - Percentuale di associazione dei pattern applicando SOM + K-Means al dataset Iris

Esiste comunque, quindi, una piccola frazione di pattern appartenenti alle classi *Iris Virginica* e *Iris Versicolor* che restano difficilmente distinguibili senza una conoscenza a priori delle classi di appartenenza. Tuttavia l'esempio proposto mostra chiaramente l'efficacia della combinazione di SOM e K-Means in una tecnica di clustering a due stadi. Nel caso di **SOM+UmatCC** l'assegnazione dei pattern alle giuste classi non avviene in maniera corretta. La classe *Iris Setosa,* ovvero quella rappresentata dal raggruppamento di pattern alla sinistra del grafico, viene riconosciuta perfettamente. Per quanto riguarda invece il raggruppamento mostrato nella parte destra del grafico, contenente i pattern appartenenti alle restanti classi, si può immaginare che l'assenza di una forzatura sul numero di cluster attesi, e l'intrinseco modo di lavorare del metodo, porti al riconoscimento, oltre delle due classi effettivamente presenti, di una terza classe intermedia, contenente quei pattern che non appartengono in maniera evidente né alla classe *Iris Viriginica* né alla classe *Iris Versicolor*.

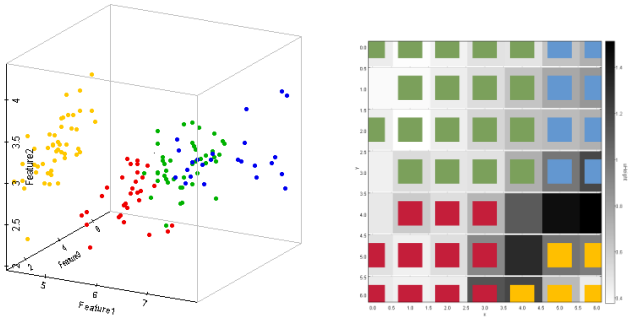


Figura 115 - SOM + Umat-CC applicato al dataset Iris (a destra la relativa U-matrix)

Nel caso di **SOM+TWL**, risulta lampante il ruolo dei nodi esterni, descritto nell' apposito paragrafo. Sebbene infatti il nodo azzurro rappresenti dei falsi outlier, ovvero pattern che sono più distanti dalla classa ma non per questo esclusi, il nodo arancione individua esattamente un punto in cui la differenza tra le classi diventa leggermente più accentuata. Questo porta ai notevoli risultati di seguito mostrati.

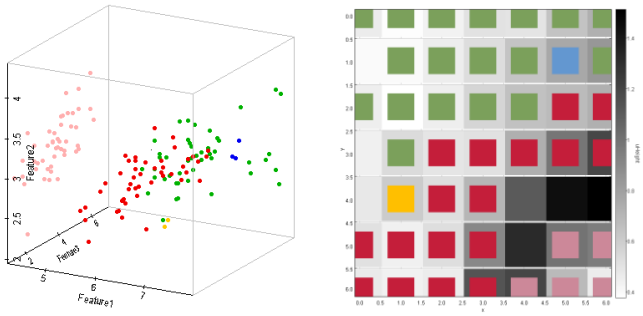


Figura 116 - SOM + TWL applicato al dataset Iris (nodo azzurro falso outlier; in arancione un nodo separatore di classi)

CLASSE	trovati	persi	%
Iris Setosa	50	0	100%
Iris Virginica	45	5	90%
Iris Versicolor	47	3	94%

Tabella 39 - Percentuale di associazione dei pattern applicando SOM + TWL al dataset Iris

Circa il modello **E-SOM**, sebbene le immagini possano indurre a pensare che il clustering effettuato sia sbagliato, in realtà è bene ricordare che, come anticipato, i cluster reali del dataset Iris sono due, quelli effettivamente individuati durante questo test. Ricordiamo infatti che il suddetto modello lavora esclusivamente basandosi sulla distribuzione dei nodi, i quali si andranno a disporre riproducendo in maniera più o meno fedele il dataset.

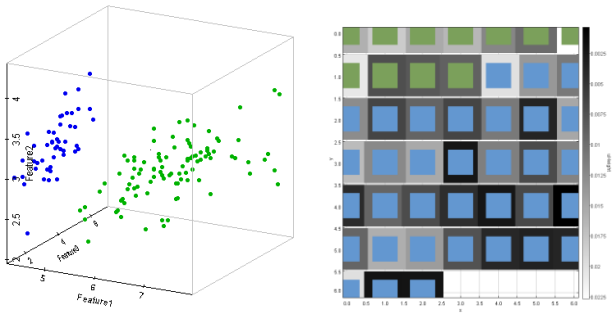


Figura 117 - E-SOM applicato al dataset Iris (a destra la relativa U-Matrix)

Per un più agevole confronto tra i risultati ottenuti, di seguito vengono riproposte le immagine riportanti l'esito del test, mentre nella tabella successiva sono riportati gli indici di valutazione.

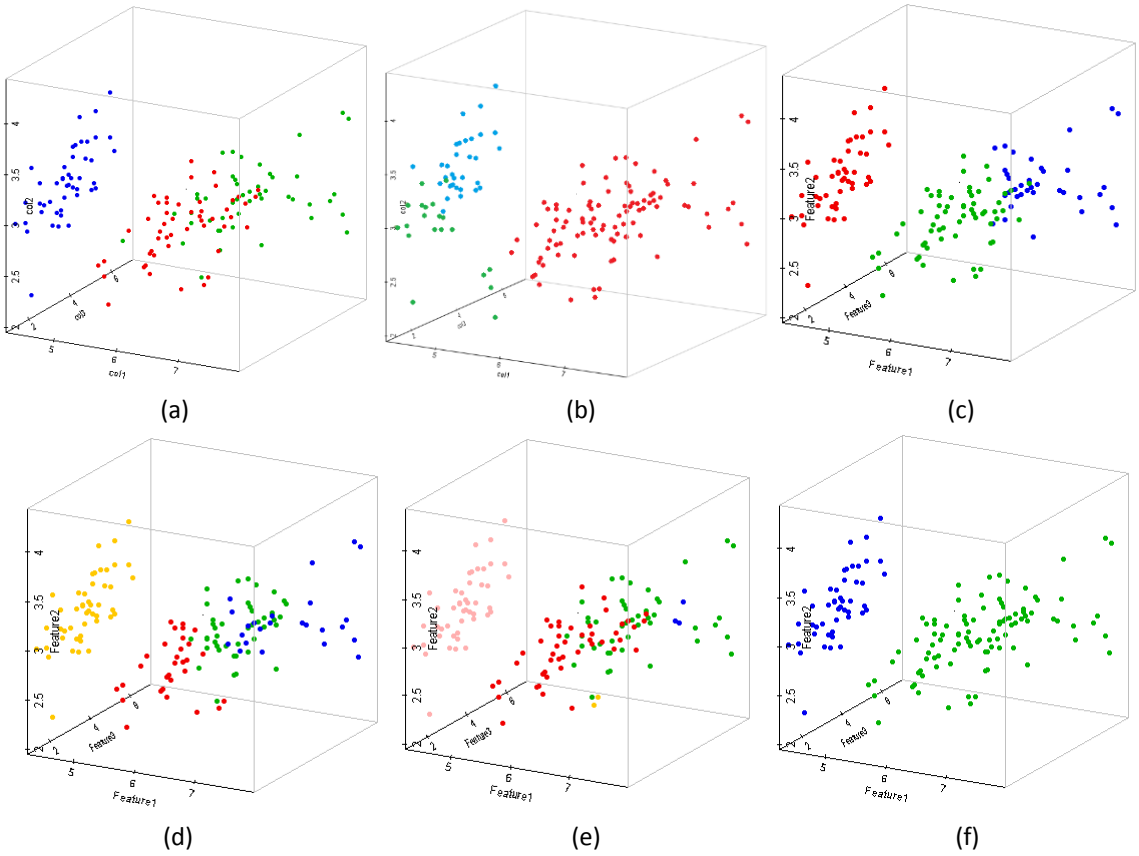


Figura 118 - Confronto di clustering del dataset Iris - Clustering teorico (a) K-Means (b) SOM + K-Means (c) SOM + Umat-CC (d) SOM + TWL (e) E-SOM (f)

CONFRONTO STATISTICO SU PRESTAZIONI DI CLUSTERING						
MODELLO	K-Means	SOM	SOM+K-means	SOM+UmatCC	SOM+TWL	E-SOM
errore di quantizzazione	5.85	0.24	0.24	0.24	0.24	0.24
errore topografico	-	0.23	0.23	0.23	0.23	-
indice di Davies-Bouldin	6.60	-	0.72	1.32	1.33	0.38

Tabella 40 – Confronto di prestazioni di clustering tra i vari modelli proposti ed il K-Means base

9.8.2 Confronto di classificazione (con MLP e SVM)

Per verificare l'efficienza sia computazionale che qualitativa del modello proposto E-TREE, nell'ambito della classificazione, abbiamo considerato il seguente problema reale, sul quale si è scelto di confrontare E-TREE con due fra i più noti e tradizionalmente impiegati classificatori supervisionati: Multi Layer Perceptron addestrato con la regola della Back Propagation (MLPBP; Rumelhart et al. 1986) e le Support Vector Machine (SVM; Chang & Lin 2011).

9.8.2.1 Il dataset GCSearch

Il dataset scelto come base dati per il test di classificazione del modello E-TREE è denominato GCSearch. Il relativo problema scientifico consiste nello studio di popolazioni di ammassi globulari (in inglese Globular Clusters, GC), in galassie esterne a quella terrestre (Brescia et al. 2012b). Questo argomento è di notevole interesse in molti settori astrofisici: dalla cosmologia, all'evoluzione dei sistemi stellari, fino alla formazione ed evoluzione di sistemi binari. In generale lo studio dei GC extra-galattici richiede l'uso di fotometria a grande campo e multi-banda. Infatti, i GC in galassie lontane (a diversi MegaParsec, Mpc, da noi), appaiono come sorgenti luminose non risolte nelle immagini astronomiche da Terra. Ciò li rende indistinguibili dalle galassie circostanti, causando un altissimo tasso di contaminazione da parte di oggetti spuri. Per tale motivo la tecnica tradizionale consiste nella selezione dei GC sulla base dei colori e delle magnitudini delle sorgenti nelle diverse bande osservate. Tuttavia, per cercare di minimizzare l'effetto di contaminazione e misurare le proprietà dei GC, come dimensioni e parametri strutturali (raggio di core, concentrazione, tasso di formazione di stelle binarie ecc.), si richiedono dati supplementari, ad alta risoluzione, ottenuti da osservazioni con telescopi spaziali (come l'Hubble Space telescope, HST). Il dataset GCSearch, utilizzato nel presente esperimento consiste quindi in dati (pattern di oggetti osservati) a grande campo (wide field) ottenuti da HST e relativi alla galassia NGC1399, una gigante ellittica distante circa 20 Mpc. Questa galassia rappresenta l'ambiente ideale di studio dei GC, a causa della sua relativa distanza e della conseguente possibilità di coprire una grande porzione del suo sistema di GC tramite un relativamente ridotto numero di osservazioni. Inoltre a quella distanza i GC sono solo marginalmente risolti da HST, permettendo quindi di verificare le prestazioni dell'algoritmo di classificazione nel caso peggiore (cioè più conservativo, dato che i dati sono parzialmente contaminati e rumorosi). I dati sono caratterizzati da pattern (cioè oggetti) costituiti ciascuno da un certo numero di parametri, ottici e strutturali, relativi alle sorgenti rivelate nelle immagini osservate. La sezione di cielo relativa alle osservazioni è evidenziata in Figura 119.

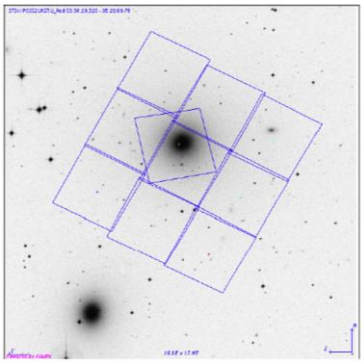


Figura 119 - Il campo di vista (FOV) coperto da HST nella banda F606W. Il campo centrale mostra la regione d'interesse per la detection di GC nelle bande G e Z

La base di conoscenza costruita per l'esperimento di classificazione supervisionata è stata ottenuta attraverso l'identificazione di GC con I metodi statistici tradizionali (Chi quadro) applicati al modello fisico degli oggetti in esame. Tali tecniche, sebbene affidabili scientificamente, sono particolarmente onerose analiticamente e richiedono un grande numero di costose campagne osservative con strumenti spaziali e da terra di grandi dimensioni. L'obiettivo ultimo dell'esperimento è dunque la possibilità di addestrare un modello empirico sulla base di poche osservazioni reali, mettendolo in condizioni di classificare correttamente i GC con un minimo numero di osservazioni ed in ambienti caratterizzati da rumore e contaminazione. Il dataset finale è costituito da 2100 pattern (oggetti), ciascuno costituito da 11 parametri (features, 7 ottiche e 4 strutturali), più la label associata (0 non GC, 1 GC), relativa alla classificazione effettuata con tecniche tradizionali. Questo dataset costituisce l'input alla fase di addestramento e di validazione dei modelli usati in questo contesto.

Di seguito si riporta una frazione del dataset, relativa ai primi 5 pattern (i parametri sono separati da virgole):

24.4753,26.7468,24.3789,0.0205,3.72,0.067,4.12,16.25,-0.1139,1.822,51.29,0,1
24.2342,26.5263,24.1632,0.0196,3.5,0.027,4.01,16.61,0.1321,1.856,35.38,0,1
23.1554,25.5964,23.1654,0.016,3.5,0.032,4.09,14.47,-0.3295,2.638,129.2,1,0
22.6316,25.3519,22.6808,0.0151,3.5,0.039,4.69,16.33,0.8065,5.002,80.45,1,0
22.4708,24.4951,22.4699,0.0216,3.5,0.066,3.45,12.81,-0.3912,-7.425,5.66,0,1

9.8.2.2 Configurazione dei modelli

In termini qualitativi, cioè di validazione scientifica dei risultati, il modello E-TREE è stato testato in classificazione sul dataset dei globular clusters, variando opportunamente i parametri di configurazione disponibili:

PARAMETRI E-TREE	
Soglia di splitting	180
Numero di nodi generati a seguito di uno splitting	3
η_0 tasso di apprendimento iniziale	0.6
σ_0 diametro iniziale del vicinato	0.5
τ_1 costante temporale	4
τ_2 costante temporale	60
Fattore di decadimento del contatore di occorrenze dei BMU	0.1
Numero di esecuzioni del K-Means per riordinare le foglie dell'albero	20
Cross Validation (k-fold leave-one-out)	10

Tabella 41 – Parametri del modello E-TREE usati per l'esperimento di classificazione

Di seguito riportiamo le informazioni relativa al settaggio dei parametri dei due modelli di classificazione scelti per il confronto.

Multilayer perceptron trained by back propagation (MLPBP)		
Input nodes	numero di features dei pattern input	11
Hidden nodes	numero di nodi dello strato interno	23
Output nodes	numero di nodi dello strato output	2
Activation functions	funzione di attivazione dei nodi della rete	tangente iperbolica
Error loop threshold	soglia di errore di training	0.001
Number of iterations	numero massimo di iterazioni	10000
Output error type	tipo di funzione di errore	cross entropy

Tabella 42 – Parametri del modello MLPBP usati per l'esperimento di classificazione

Support Vector Machine (SVM)		
Kernel	tipo di funzione di kernel	RBF
Gamma	gamma del kernel	[2 – 223]
Error tolerance	soglia di errore di training	0.001
Cross validation	k-fold leave-one-out	5
Cachesize	cache memory size	100MB

Tabella 43 – Parametri del modello SVM usati per l'esperimento di classificazione

9.8.2.3 Risultati del confronto

I tre modelli su citati sono stati addestrati e testati sul dataset descritto in precedenza, il cui scopo era la classificazione di una tipologia di oggetto esteso dell'Universo (ammassi globulari). Per poter confrontare opportunamente i tre metodi, si è scelto di utilizzare tre indicatori statistici che possono essere facilmente calcolati a partire dall'output del test di classificazione. Essi sono:

- CL_ACC : percentuale di accuratezza di classificazione, intesa come media del numero di pattern input correttamente classificati rispetto alla loro classe teorica su entrambe le classi;
- CONTAMINATION: percentuale di pattern input teoricamente appartenenti alla classe dei non GC, ma erroneamente classificati come GC, rispetto al totale di oggetti GC attesi;

CONFRONTO SU PRESTAZIONI DI CLASSIFICAZIONE			
MODELLO	MLPBP	SVM	E-TREE
CL_ACC [%]	59.9	90.5	87.1
CONTAMINATION [%]	42.2	7.7	9.4

Tabella 44 – Confronto di prestazioni di classificazione tra E-TREE ed i due modelli esterni selezionati

Dalla Tabella 44 si evince che il modello E-TREE risulta confrontabile con SVM, sia in termini di accuratezza che di contaminazione e molto meglio della classica rete neurale MLPBP. Tuttavia, il vantaggio del modello E-TREE proposto, rispetto al modello SVM, risulta oltremodo indubbio se si confrontano i rispettivi tempi di esecuzione del training. Infatti, a fronte di circa 2 ore per l'SVM (ossia 7010 secondi), il modello E-TREE impiega 3 secondi, a parità di condizioni (tipo di macchina e dataset). Quindi il minimo deficit qualitativo delle prestazioni del modello E-TREE, rispetto a SVM, è pienamente compensato dall'estrema rapidità di esecuzione, tre ordini di grandezza migliore.

10 CONCLUSIONI

Il presente lavoro si colloca all'interno del settore del data mining con metodi di machine learning, dedicato alle funzionalità di clustering, riduzione di dimensionalità e classificazione. Lo scopo principale del lavoro è estendere la web application suite di data mining DAMEWARE con una famiglia di modelli di machine learning dedicati in particolare al clustering di immagini e dati tabulari. I modelli proposti sono stati selezionati sulla base di caratteristiche prettamente tarate su problemi reali di tipo astrofisico, dove immagini (mono- e multi-banda) e grandi tabelle di dati presentano un elevato livello di difficoltà (basso rapporto signal-to-noise) ai fini del clustering e classificazione di oggetti celesti.

Una delle principali linee guida nella selezione e progettazione dei modelli proposti è stata l'evoluzione del classico modello SOM (Self Organizing Maps), in modo da superare l'intrinseca limitazione della topologia statica della griglia output di Kohonen. L'evoluzione del modello SOM storicamente ha proceduto in due direzioni: (i) mantenimento della topologia della griglia e modifica della legge di apprendimento; (ii) mantenimento della legge di apprendimento e modifica della topologia della griglia. Seguendo questi due filoni di ricerca, sono stati selezionati, progettati e sviluppati rispettivamente, il modello E-SOM (Evolving SOM), dedicato al clustering non supervisionato ed il modello E-TREE (Evolving Tree), dedicato alla classificazione supervisionata.

Relativamente al modello standard SOM, oltre allo sviluppo ed implementazione del modello base, si è proceduto ad elaborare e sviluppare una serie di metodi di clustering per il raffinamento dell'output della SOM (considerabile più come una riduzione di dimensioni dello spazio dei parametri). Tale sistema è stato definito come two-stage clustering, in cui il primo livello è costituito dalla SOM base, seguito dal livello di post-processing. Per quest'ultimo sono stati elaborati due metodi standard, K-means e UmatCC (U-matrix a Componenti Connesse), più un metodo completamente nuovo, denominato TWL (Two Winners Linkage), ispirato al metodo dinamico della E-SOM, applicata all'output della SOM.

Per tutti i modelli citati, si è proceduto alla progettazione e sviluppo del codice in linguaggio C, con particolare attenzione a rispettare tutti i vincoli strutturali e d'interfaccia richiesti dall'infrastruttura della web application DAMEWARE, in cui tali modelli dovevano essere innestati e resi fruibili all'utenza esterna.

Di particolare rilevanza risulta uno dei modelli di clustering two-stage, SOM+TWL, in quanto ideato e progettato ex novo in questo lavoro. Dai test effettuati, tale modello risulta perfettamente confrontabile con gli altri modelli proposti. In particolare il metodo TWL elaborato da zero, in qualità di post-processing della SOM, ha prestazioni equiparabili e in qualche caso superiori ai metodi più tradizionali di post-processing, quali K-means e UmatCC.

In futuro, si prevede una sessione esaustiva di test su varie tipologie di immagini e dataset tabulari per tutti i modelli proposti, in modo da evidenziare ed enucleare tutti gli aspetti e le caratteristiche specifiche rispetto alle peculiarità dei dati esaminati in vari contesti disciplinari (con particolare riferimento alle problematiche astrofisiche). Sebbene i test effettuati fin'ora mostrino risultati incoraggianti, si dovrà procedere comunque ad un'attenta analisi critica del nuovo modello proposto (TWL), eventualmente evolevondolo in una versione in grado di superare delle problematiche emerse.

11 BIBLIOGRAFIA

Brescia M. et al., 2010. *"DAME: A Web Oriented Infrastructure for Scientific Data Mining & Exploration".* arXiv:1010.4843, http://adsabs.harvard.edu/abs/2010arXiv1010.4843B;

Brescia M. , 2012. *ETREE: a machine learning model for scalable unsupervised classification*. Internal technical documentation, DAME Consortium, code: etree_DAMEWARE-SDD-NA-0021-Rel1.3

Brescia, M. et al., 2012b. *The detection of globular clusters in galaxies as a data mining problem*. Monthly Notices of the Royal Astronomical Society, Volume 421, Issue 2, pp. 1155-1165

Bruske J., Sommer G., 1995. *Dynamic cell structure learns perfectly topology preserving map*. Neural Comput. 7 845–865;

Caudill M., Butler C., 1992. "*Naturally intelligent systems".* Mit Press;

Chang C.C., Lin C.J., 2011. ACM Trans. Intelligent Syst. Technol., 2, 27

Chi S-C., Yang C-C., 2008. "*A Two-stage Clustering Method Combining Ant Colony SOM and K-means"*. Journal of Information Science and Engineering 24, 1445-1460;

Davies D.L., Bouldin D.W., 1979. "*A cluster separation measure"*. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 1, 224-227;

Deng D., Kabasov N., 2003. *On-line pattern analysis by evolving self-organizing maps*. Neurocomputing 51, Elsevier, 87-103;

Esposito F., 2013. *Clustering con Modelli Software Dinamici*. Seminario Dip. di Informatica, Università degli Studi di Napoli Federico II, http://dame.dsf.unina.it/documents.html

Fisher R.A., 1936. *"The use of multiple measurements in taxonomic problems".* Annals of Eugenics. Vol.7, 179-188;

Fritzke B., 1994. *Growing cell structures—a self-organizing network for unsupervised and supervised learning*. Neural Networks 7, 1441–1460;

Fritzke B., 1995. *A growing neural gas network learns topologies*. In: D. Touretzky, T.K. Keen (Eds.), Advances in Neural Information Processing Systems, Vol. 7, MIT Press, Cambridge, MA, pp. 625–632;

Hamel L., Brown C.W., 2011. "*Improved interpretability of the unified distance matrix with connected components".* Proceedings of the 2011 International Conference on Data Mining;

Hartigan, J. A., **Wong**, M. A., **1979**. "*A K-means clustering algorithm*". Applied Statistics, 28, 100–108;

Kiang M. Y., 2001. "*Extending the Kohonen self-organizing map netowrks for clustering analysis*". Computational Statistics & Data Analysis. Vol. 38, 161-180;

Kiviluoto K., 1996. "Topology preservation in self-organizing maps". *Proceedings of the International Conference on Neural Networks.* 294-299;

Kohavi R., 1995. *A study of cross-validation and bootstrap for accuracy estimation and model selection.* Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann Editions, San Mateo, Vol. 2, pp. 1137–1143

Kohonen T., 2001. "*Self-Organizing Maps".* 3rd ed., Springer;

McCulloch W.S., Pitts W.H, 1943. "*A logical calculus of the ideas immanent in nervous activity*". Bullettin of mathematical biophysics. Vol. 5, 115-133;

Moutarde F., Ultsch A., 2005. "U*F Clustering: A new performant cluster-mining method on segmentation of self-organizing map". Proceedings of WSOM '05, September 5-8, Paris, France, 25-32;

Murtagh F., 1985. "*Multidimensional clustering alghoritms*". Physica-Verlag, Wurzburg;

Pakkanen J., Livarinen J., Oja E., 2004. *The Evolving Tree—A Novel Self-Organizing Network for Data Analysis*. Neural Processing Letters 20, 199–211, Kluwer Academic Publishers;

Pakkanen, J., 2003. *The Evolving Tree, a new kind of self-organizing neural network*. Published in the Proceedings of the Workshop on Self-Organizing Maps.

Pakkanen, J., Iivarinen, J., 2003. *A Novel Self-Organizing Neural Network for Defect Image Classification*. Published in Proceedings of the International Joint Conference on Neural Networks.

Rumelhart D. E., Hinton G. E. and Williams R. J., 1986. *Learning Internal Representations by Error Propagation*. David E. Rumelhart, James L. McClelland, and the PDP research group. (editors), Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundations. MIT Press

Shapiro L. G., Stockman G. C, 2001. *Computer Vision*. Prentice Hall, page 137-150

Ultsch, A., 2005. *"Clustering with SOM: U*C.* Proc. Workshop on Self-Organizing Maps, Paris, France. 75-82.

Vesanto J., Alhoniemi E., 2000. "*Clustering of the Self-Organizing Map".* IEEE Transactions on neural networks. Vol. 11, No. 3, 586-600;

12 APPENDICE A: DESCRIZIONE DEL CODICE SORGENTE SOM

Il software è scritto in linguaggio C al fine di consentire una successiva parallelizzazione del codice.
Il progetto è composto dai seguenti file:

- *main.c*
- *som.h*
- *utils.h*
- *somVector.h*
- *somParams.h*
- *somOutput.h*
- *som.c*
- *utils.c*
- *somVector.c*
- *somParams.c*
- *somOutput.c*
- *fitsio.h(*)*
- *longnam.h(*)*
- *il.h(*)*
- *Palette.pal(*)*
- *Stilts.jar(*)*

(il file è relativo ad una libreria esterna o ad un file ausiliario)*

Il ***main*** è, ovviamente il file che contiene il flusso principale dell'esecuzione, mostrato nei relativi Flow Chart. Il file ***somParams*** contiene la definizione delle strutture contenenti i parametri dell'esperimento e i nomi dei file che dovranno essere genearti in output, oltre alle funzioni per la loro gestione. Le funzioni contenute nel file ***somVector*** si occupano invece della gestione della memoria necessaria all'esecuzione dell'esperimento configurato. Il file ***utils*** contiene funzioni di carattere generale utili durante l'esecuzione del programma. Le funzioni del file ***somOutput*** sono invece utili alla creazione dell'output sia testuale che grafico fornito dal software. Il file ***som*** invece contiente le funzioni specifiche relative all'omonimo modello di rete neurale.

Nei successivi paragrafi verrà mostrato in dettaglio il contenuto di ognuno dei file sopraccitati.

12.1 SOMPARAMS

STRUCT EXPERIMENT	
Int useCase	Parametro che indica lo use case selezionato dall'utente
Int nInputNodes	Parametro che indica il numero di nodi dello strato di input
Int nOutputNodes	Parametro che indica il numero di nodi dello strato di output
Int nPatterns	Parametro che indica il numero di pattern del dataset passato in input
Int maxIterations	Parametro che indica il numero massimo di iterazioni che l'algoritmo deve eseguire
Int execIterations	Parametro che tiene traccia delle iterazioni eseguite incrementando ogni qual volta un'iterazione viene portata a termine
Int neighborSize	Parametro che indica il diametro del vicinato iniziale
Double eta0	Parametro che indica il learning rate iniziale
Double eta1	Parametro che indica la soglia minima che il learning rate può raggiungere prima che si interrompa l'algoritmo
Double eta	Parametro che tiene traccia del learning rate attuale durante l'esecuzione dell'algoritmo
Int gridRows	Parametro che indica il numero di righe dello strato di output della rete
Int gridCols	Parametro che indica il numero di colonne dello strato di output della rete
Time_T startTime	Data di inizio dell'esperimento
Time_T endTime	Data di fine dell'esperimento
Int treD	Valore booleano che indica l'utilizzo o meno di uno strato di output tridimensionale
Int normalization	Valore booleano che indica la richiesta o meno di normalizzazione del dataset in input

Int nClusters	Parametro che indica il numero di cluster individuati
Int nBMU	Parametro che indica il numero di neuroni dello strato di output risultati BMU
Int expectClst	Numero di cluster attesi
Double sigmaGaussianBlur	Parametro che indica il grado di sfocatura della U-Matrix
Int postProcessCase	Use case selezionato per il l'applicazione di un metodo di post processing della SOM
Int postProcessMethod	Parametro che indica il metodo di clustering da applicare alla SOM ottenuta
Char* inputDataset	Stringa contenente il nome del dataset passato in input
Int datasetFormat	Parametro che indica il formato del dataset
Char* usedDataet	Stringa contenente il nome del dataset utilizzato durante l'esperimento. Nel caso in cui sia necessaria la conversione del dataset infatti, il nome del file passato in input non coinciderà più con quello utilizzato durante l'esperimento. Esempio: InputDataset[] = ***dataset.jpg*** → usedDataset[] = ***dataset.txt***
Char* configurationFile	Stringa contenente il nome dell'eventuale file di configurazione
Char* targetClusterFile	Stringa contenente il nome dell'eventuale file con i cluster attesi per ogni pattern
Int outImageWidth	Parametro che salva, nel caso il dataset sia un'immagine, la sua larghezza
Int outImageHeight	Parametro che salva, nel caso il dataset sia un'immagine, la sua altezza
Int slice	Parametro che indica il numero di slice dell'immagine. In caso di immagine bidimensionale, sarà pari ad 1
Char *commandLine	Stringa contenente la riga di comando immessa dall'utente
Int **Coordinates	Array bidimensionale con un numero di righe pari al numero di nodi che compongono lo strato di output e un numero di colonne pari a 2 o 3 a seconda che si utilizzi rispettivamente uno strato di Kohonen bidimensionale o tridimensionale. Scopo di questa struttura dati e tener traccia delle coordinate matriciali di ogni nodo di output
Int *Winners	Vettore di interi di dimensioni pari al numero di pattern che compongono il dataset. L'indice del vettore coinciderà quindi con l'ID del pattern e alla corrispondente locazione di memoria verrà memorizzato l'ID del neurone vincente per il suddetto pattern.
Int *bmuVictories	Vettore di interi di dimensioni pari al numero di nodi che compongono lo strato di output. L'indice del vettore coinciderà quindi con l'ID del nodo e alla corrispondente locazione di memoria verrà memorizzato il numero di pattern assegnati al suddetto BMU.
Int *ClustersCount	Vettore di interi di dimensioni par al numero di cluster individuati che, per ognuno di essi riporta il numero di pattern assegnati.
Double **Weights	Array bidimensionale con un numero di righe pari al numero di neuroni facenti parte dello strato di output ed un numero di colonne pari al numero di nodi facenti parte dello strato di input. Tale array serve a memorizzare il peso del collegamento tra ogni nodo di output ed ogni nodo di input
Double **Out0	Array bidimensionale con un numero di righe pari al numero di pattern che compongono il dataset. Il numero di colonne è invece pari al numero di nodi dello strato di input, al fine di memorizzare, per ogni pattern le relative features.
Int *ClusterAssigned	Vettore di interi di dimensioni par al numero di nodi che compongono lo strato di output. L'indice del vettore coinciderà con l'ID del nodo e alla corrispondente locazione di memoria verrà memorizzato la label indicante il cluster assegnato.
Double *Activations	Vettore con un numero di righe pari al numero di pattern che per ognuno di essi memorizza il valore di attivazione del relativo neurone vincente.

Tabella 45 – Struct Experiment del modello SOM

STRUCT FILENAME	
Char* RES_FILE	Nome del file ***SOM_XXX_results.txt***
Char* CLST_FILE	Nome del file ***SOM_XXX_clusters_count.txt***
Char* LOG_FILE	Nome del file ***SOM_XXX_status.log***
Char* NORM_RES_FILE	Nome del file ***SOM_XXX_normalized_results.txt***
Char* HISTO_IMAGE	Nome del file ***SOM_XXX_histogram.png***

Char* NET_CONF_FILE	Nome del file ***SOM_Train_network_configuration.txt***
Char* CLSTRD_IMAGE	Nome del file ***SOM_XXX_clustered_image.png***
Char* CLSTRD_IMAGE_TXT	Nome del file **SOM_XXX_clustered_image.txt**
Char* VALIDITY_INDX	Nome del file ***SOM_XXX_validity_indices.txt***
Char* UMATRIX	Nome del file ***SOM_XXX_U_Matrix.png***
Char* KOHONEN_LYR	Nome del file ***SOM_XXX_output_layer.txt***
Char* DATACUBE_ZIP	Nome del file ***SOMl_XXX_clustered_datacube_image.zip***

Tabella 46 – Struct FileName del modello E-SOM

char* **getTextUseCase** (int UseCase)

Parametri in input	• **UseCase:** use case selezionato dall'utente
Parametro restituito	• Stringa contente lo use case selezionato dall'utente in formato testuale
Descrizione	La funzione prende in input lo use case selezionato in formato numerico e lo converte in formato testuale restituendo tale stringa. Tale funzione risulta utile nella generazione dinamica dei nomi dei file di output.

int **isTable** (char *extension)

Parametri in input	• **extension:** estensione del dataset in input al programma
Parametro restituito	• Valore booleano
Descrizione	La funzione restituisce TRUE se l'estensione del dataset coincide con una di quelle ammesse dal formato *table*.

int **isImage** (char *extension)

Parametri in input	• **extension:** estensione del dataset in input al programma
Parametro restituito	• Valore booleano
Descrizione	La funzione restituisce TRUE se l'estensione del dataset coincide con una di quelle ammesse dal formato *image*.

int **isFitsImage** (char *extension)

Parametri in input	• **extension:** estensione del dataset in input al programma
Parametro restituito	• Valore booleano
Descrizione	La funzione restituisce TRUE se l'estensione del dataset coincide con una di quelle ammesse dal formato *fits_image*.

int **isAscii** (char *extension)

Parametri in input	• **extension:** estensione del dataset in input al programma
Parametro restituito	• Valore booleano
Descrizione	La funzione restituisce TRUE se l'estensione del dataset coincide con una di quelle ammesse dal formato *ASCII*.

void **fileNameGenerator** (struct FileName *Files, int UseCase)

Parametri in input	• **Files:** struttura in cui sono memorizzati i nomi dei file di output • **UseCase:** use case selezionato
Parametro restituito	• -
Descrizione	La funzione si occupa di generare dinamicamente i nomi dei file di output. Come descritto nel capitolo apposito infatti, i nomi dei file variano in base ai parametri dell'esperimento. Il parametro in input **UseCase,** una volta convertito in formato testuale, andrà posto all'interno del nome del file.

void **checkParameters** (struct Experiment *Exp, int *givenParams)

Parametri in input	• **Exp:** struttura contente i parametri dell'esperimento • **givenParmas:** vettore con valori booleani riguardanti i parametri passati in input
Parametro restituito	•
Descrizione	La funzione controlla che i parametri in input rispettino i vincoli imposti in fase di progettazione. Il controllo sul passaggio del file di configurazione viene effettuato al di fuori della funzione. Il motivo risiede nel fatto che la suddetta funzione viene richiamata a seguito di un'eventuale lettura di tale file, mentre il controllo sullo stesso deve essere effettuato evidentemente prima di una sua apertura.

void **initialize** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione inizializza i parametri dell'esperimento con valori di default.

void **errorReport** (int errorCode, struct Experiment *Exp)

Parametri in input	• **errorCode:** codice dell'errore che si è verificato • **Exp**: struttura con i parametri relativi all'esperimento
Parametro restituito	-
Descrizione	La funzione viene chiamata quando si verifica un'interruzione inaspettata del programma. Ad ogni condizione di uscita forzata è associato un codice che la funzione utilizza per stampare, nel file di output, il relativo messaggio di errore insieme alla riga di comando immessa dall'utente. Successivamente alla creazione del file, viene richiamata la funzione adibita alla deallocazione della memoria occupata.

void **patternCount** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione analizza il dataset contando il numero di pattern contenuti al suo interno. Controlla inoltre che la dimensione dei pattern coincida con il numero di nodi che costituiscono lo strato di input.

void **randomWeights** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione si occupa di generare in maniera casuale i pesi iniziali dei neuroni dello strato di output, normalizzati tra -1 ed 1.

long int **readpatterns** (long int startFile, struct Experiment *Exp)

Parametri in input	• **startFile:** posizione della quale iniziare a leggere il file • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Posizione del cursore nel file al termine della lettura
Descrizione	La funzione si occupa di acquisire e memorizzare i pattern letti all'interno del dataset.

void **readConfigFile** (struct Experiment *Exp, long int *weightsPosition)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Posizione del cursore nel file al termine della lettura
Descrizione	La funzione serve per l'acquisizione dei parametri da un file di configurazione, anziché da linea di comando. Si utilizza quindi solitamente nei casi di Test e Run, oppure in caso di Resume Training. A seconda del valore assunto dal parametro **weightsPosition,** la funzione agirà in maniera differente. Se uguale a 0, allora la funzione leggerà, partendo dall'inizio del file, i parametri dell'esperimento. Viceversa, il valore indicherà la posizione nel file dei vettori dei pesi che la funzione dovrà leggere.

void **coordinatesComputation** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione ricava, a partire dall'ID del nodo, le coordinate matriciali dello stesso.

void **normalizeDataset** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura con i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione normalizza le features del dataset in input nell'intervallo [-1, 1].

void **targetCount** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione assicura che il numero di cluster attesi contenuti, all'interno dell'apposito file, sia uguale al numero di pattern del dataset.

12.2 SOMVECTOR

int** **IntMatrixAllocate** (int rows, int cols)

Parametri in input	• **rows:** numero di righe della matrice • **cols:** numero di colonne della matrice
Parametro restituito	• Matrice di interi
Descrizione	La funzione si occupa di allocare la memoria per una matrice di interi delle dimensioni specificate dai parametri in input

double** **DoubleMatrixAllocate** (int rows, int cols)

Parametri in input	• **rows:** numero di righe della matrice • **cols:** numero di colonne della matrice
Parametro restituito	• Matrice di double
Descrizione	La funzione si occupa di allocare la memoria per una matrice di double delle dimensioni specificate dai parametri in input

void **DoubleMatrixFree** (double **matrix, int rows)

Parametri in input	• **matrix:** matrice da deallocare • **rows:** numero di righe della matrice
Parametro restituito	• -
Descrizione	La funzione si occupa di liberare la memoria allocata per una matrice di double il cui numero di righe è specificato tra i parametri in input

void **IntMatrixFree** (int **matrix, int rows)

Parametri in input	• **matrix:** matrice da deallocare • **rows:** numero di righe della matrice
Parametro restituito	• -
Descrizione	La funzione si occupa di liberare la memoria allocata per una matrice di double il cui numero di righe è specificato tra i parametri in input

void **freeMemory** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione si occupa di deallocare la memoria allocata dinamicamente.

void **allocMemory** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione si occupa di allocare dinamicamente la memoria necessaria.

12.3 UTILS

int **fileExist** (char *filename)

Parametri in input	• **filename:** nome del file di cui verificare l'esistenza
Parametro restituito	• valore booleno
Descrizione	La funzione verifica l'esistenza del file passato in input.

int **toAsciiHex** (int x)

Parametri in input	• **x:** valore da convertire
Parametro restituito	• Valore convertito
Descrizione	La funzione converte un numero compreso tra 0 e 15 nel suo corrispettivo esadecimale restituendo il codice ASCII del valore convertito.

void **getColor** (int id, char *color)

Parametri in input	• **id:** indice del colore nella palette • **color:** stringa contenente il colore
Parametro restituito	• -
Descrizione	La funzione converte un colore in formato RGB letto all'interno di una palette, in formato esadecimale. Il colore da convertire è indicato dal parametro **id,** ovvero l'indice dello stesso nella palette.

char* **getExtension** (char *File)

Parametri in input	• **File:** nome del file in input
Parametro restituito	• Estensione del file
Descrizione	La funzione analizza il nome del file restituendone l'estensione.

void **jumpHeader** (FILE *File)

Parametri in input	• **File:** file sul quale la funzione dovrà lavorare
Parametro restituito	• -
Descrizione	La funzione muove il cursore, del file passato in input, nel punto immediatamente successivo ad eventuali header lines.

double **doubleRangeRand** (const double a, const double b)

Parametri in input	• **a:** estremo inferiore dell'intervallo entro il quale generare un numero casuale • **b:** estremo superiore dell'intervallo entro il quale generare un numero casuale
Parametro restituito	• Numero casuale compreso nell'intervallo specificato
Descrizione	La funzione serve a generare un numero casuale all'interno di un intervallo specificato dai parametri in input.

int **max** (double x, double y)

Parametri in input	• **x:** primo valore • **y:** secondo valore
Parametro restituito	• Massimo tra i due valori
Descrizione	La funzione calcola e restituisce il massimo tra i due valori passati in ingresso.

int **min** (double x, double y)

Parametri in input	• **x:** primo valore • **y:** secondo valore
Parametro restituito	• Minimo tra i due valori
Descrizione	La funzione calcola e restituisce il minimo tra i due valori passati in ingresso.

void **help** ()

Parametri in input	• **-**
Parametro restituito	• -
Descrizione	Funzione che mostra a video una guida sull'uso del software.

void **debugFitsImage** (char *inputFits, char *outputFits)

Parametri in input	• **inputFits:** nome del file FITS in input • **outputFits:** nome del file FITS in output
Parametro restituito	• -
Descrizione	La funzione legge l'immagine FITS indicata dal primo parametro e copia il contenuto all'interno del secondo. Tale procedimento ha lo scopo di verificare la corretta lettura del file da parte del programma.

12.4 SOMOUTPUT

void **writeConfig** (char *configFile, struct Experiment Exp)

Parametri in input	• **configFile:** nome del file di configurazione da scrivere • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione si occupa di scrivere il file ***SOM_Train_network_configuration.txt.***

void **writeResults** (char *clusterFile, struct Experiment Exp, int normClusters)

Parametri in input	• **clusterFile:** nome del file da generare • **Exp:** struttura contenente i parametri dell'esperimento • **normClusters:** flag che indica il tipo di file da generare
Parametro restituito	• -
Descrizione	La funzione scrive, a seconda del valore assunto dal parametro **normClusters,** il file ***SOM_XXX_results.txt*** o il file ***SOM_XXX_normalized_results.txt.***

void **writeStatus** (char *statusFile, struct Experiment Exp)

Parametri in input	• **statusFile:** nome del file da generare • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	Funzione che si occupa di scrivere il file ***SOM_XXX_status.log.***

void **writeClusters** (char *countFile, struct Experiment Exp)

Parametri in input	• **countFile:** nome del file da generare • **Exp:** struttura con i parametri dell'esperimento
Parametro restituito	• -
Descrizione	Funzione che si occupa di scrivere il file ***SOM_XXX_clusters_count.txt.*** La percentuale di associazione tra il numero di pattern assegnati ad ogni cluster e il numero di pattern totali, è calcolato all'interno della funzione stessa.

void **writeOutputLayer** (char *kohonenFile, struct Experiment Exp)

Parametri in input	• **kohonenFile:** nome del file da generare • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	Funzione che scrive il file ***SOM_XXX_output_layer.txt.***

void **writeValidityIndx** (char *validityFile, struct Experiment Exp)

Parametri in input	• **validityFile:** nome del file da generare • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	Funzione che scrive il file ***SOM_XXX_validity_indices.txt.***

void **drawUmatrix** (char *inFileName, char *outFileName, struct Experiment Exp)

Parametri in input	• **inFileName:** nome del file da passare in input al comando STILTS • **outFileName:** nome del file generato dal comando STILTS • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione utilizza il comando ***plot2d*** di STILTS per disegnare la U-Matrix relativa alla SOM. Maggiori informazioni sul comando sono disponibili al Paragrafo 15.3.2.

void **drawClusteredImage**(char *outFileName,struct Experiment Exp, int slice)

Parametri in input	• **outFileName:** nome dell'immagine generata • **Exp:** struttura contenente i parametri dell'esperimento • **slice:** slice dell'immagine cui la funzione fa riferimento
Parametro restituito	• -
Descrizione	La funzione genera, tramite l'ausilio delle routine facenti parte della libreria DevIL, l'immagine ***SOM_XXX_clustered_image.png.*** In base al parametro **slice,** la funzione sarà in grado di calcolare il pattern da cui iniziare la scrittura dell'immagine, al fine di generare l'output relativa alla slice passata in input.

void **drawHistogram** (struct Experiment Exp, char *inFileName, char *outFileName)

Parametri in input	• **Exp:** struttura con i parametri dell'esperimento • **inFileName:** nome del file da passare in input al comando STILTS • **outFileName:** nome del file generato dal comando STILTS
Parametro restituito	• -
Descrizione	La funzione utilizza il comando ***plothist*** per disegnare un istogramma che riporta il numero di pattern assegnati ad ogni cluster. Maggiori informazioni riguardo il comando sono disponibili al Paragrafo 15.3.2.

void **writeClusteredImage**(char *outFileName,struct Experiment Exp, int slice)

Parametri in input	• **outFileName:** nome dell'immagine generata • **Exp:** struttura contenente i parametri dell'esperimento • **slice:** slice dell'immagine cui la funzione fa riferimento
Parametro restituito	• -
Descrizione	La funzione genera, tramite l'ausilio delle routine facenti parte della libreria DevIL, l'immagine ***SOM_XXX_clustered_image.png.*** In base al parametro **slice,** la funzione sarà in grado di calcolare il pattern da cui iniziare la scrittura dell'immagine, al fine di generare l'output relativa alla slice passata in input, generando inoltre il corretto prefisso per il nome. In caso di immagini bidimensionali, la slice sarà chiaramente unica e non ci sarà bisogno di alcun prefisso.

12.5 SOM

void **getQuantizationError** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione restituisce l'errore di quantizzazione relativo alla SOM.

void **getTopographicalError** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione restituisce l'errore topografico relativo alla SOM.

void **getDaviesBouldin** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione restituisce l'indice di Davies-Bouldin.

double **distPatterns** (int pattern1, int pattern2, struct Experiment Exp)

Parametri in input	• **pattern1:** primo pattern da cui misurare la distanza • **pattern2:** secondo pattern da cui misurare la distanza • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Distanza euclidea tra i due pattern
Descrizione	La funzione si occupa di calcolare la distanza euclidea tra due pattern utilizzando, come coordinate, le relative features.

double **distPatternNode** (int pattern, int node, struct Experiment Exp)

Parametri in input	• **pattern:** pattern da cui misurare la distanza • **node:** nodi di output da cui misurare la distanza • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Distanza euclidea tra il nodo e pattern in input
Descrizione	La funzione si occupa di calcolare la distanza euclidea tra un pattern ed un nodo dello strato di output utilizzando, come coordinate, rispettivamente le features e i pesi.

double **gridDistance** (int node1, int node2, struct Experiment Exp)

Parametri in input	• **node1:** primo nodo da cui misurare la distanza • **node2:** secondo nodo da cui misurare la distanza • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Distanza euclidea tra i due nodi
Descrizione	La funzione si occupa di calcolare la distanza euclidea tra due nodi, sul reticolo dello strato di output, utilizzando quindi, come coordinate, le rispettive coordinate matriciali.

double **distNodes** (int node1, int node2, struct Experiment Exp)

Parametri in input	• **node1:** primo nodo da cui misurare la distanza • **node2:** secondo nodo da cui misurare la distanza • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Distanza euclidea tra i due nodi
Descrizione	La funzione si occupa di calcolare la distanza euclidea tra due nodi dello strato di output utilizzando, come coordinate, i relativi pesi.

double **distNodeCentroids** (int node, int centroid, struct Experiment Exp)

Parametri in input	• **node:** nodo da cui calcolare la distanza • **centroid:** centroide da cui calcolare la distanza • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Distanza euclidea tra un nodo ed un centroide.
Descrizione	La funzione calcola la distanza tra un nodo ed un centroide.

double **distPatternCentroids** (int pattern, int centroid, struct Experiment Exp)

Parametri in input	• **pattern:** pattern da cui calcolare la distanza • **centroid:** centroide da cui calcolare la distanza • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Distanza euclidea tra un nodo ed un centroide.
Descrizione	La funzione calcola la distanza tra un pattern ed un centroide.

double **distCentroids** (int centroid1, int centroid2, struct Experiment Exp)

Parametri in input	• **centroid1:** primo centroide da cui calcolare la distanza • **centroid2:** secondo centroide da cui calcolare la distanza • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Distanza euclidea tra due centroidi.
Descrizione	La funzione calcola la distanza tra due centroidi.

void **clustersAnalysis** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione, in base ai parametri dell'esperimento, configura l'appropriato metodo di clustering da effettuare sulla SOM ottenuta.

void **uMatrixGraph** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione attua il metodo di clustering delle SOM descritto al paragrafo 5.4.2;

void **findInternalNode** (int node, struct Experiment *Exp, double *umatrix)

Parametri in input	• **node:** nodo da cui partire per cercare il nodo interno • **Exp:** struttura contenente i parametri dell'esperimento • **umatrix:** vettore contenente i valori contenuti nella U-Matrix
Parametro restituito	• -
Descrizione	La funzione trova, a partire da un nodo dello strato di output, il relativo nodo interno.

void **som_K_means** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione applica l'algoritmo del K-Means all'output generato dalla SOM.

void **findCentroid** (int bmu, struct Experiment Exp, double **Centroids)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento • **bmu:** id del nodo di cui individuare il centroide appropriato • **Centroids:** matrice contenente i centroidi
Parametro restituito	• -
Descrizione	La funzione, a partire dai centroidi contenuti nella matrice indicata dal parametro **Centroids**, trova quello più vicino al punto indicato dal parametro **bmu.**

void **repositioningCentroids** (struct Experiment Exp, double **Centroids, int *BMUs)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento • **Centroids:** matrice contenente i centroidi • **BMUs:** vettore contenente gli indici dei nodi risultati BMU
Parametro restituito	• -
Descrizione	La funzione ricalcola la posizione dei centroidi contenuti nella matrice **Centroids**.

int **findTwoWinners** (int pattern, int *BMUs, struct Experiment Exp)

Parametri in input	• **pattern:** indice di un pattern • **BMUs:** array per il salvataggio dei risultati • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• ID dei primi due BMU
Descrizione	La funzione si occupa di calcolare e restituire l'ID dei due nodi vincenti per il pattern indicato.

int **assignCluster** (int node, int **Connections, struct Experiment *Exp)

Parametri in input	• **node:** indice di un nodo dello strato di output • **int **Connections:** matrice delle connessioni tra nodi • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Label del cluster assegnato al nodo
Descrizione	La funzione, partendo da un nodo, segue le sue connessioni, assegna le relative label del cluster.

int **twoWinnersLnk** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• ID del neurone vincente
Descrizione	La funzione applica il metodo di post-processing TWL.

int **getExternalNode** (int node, double *umatrix, struct Experiment *Exp)

Parametri in input	• **node:** indice di un nodo dello strato di output • **double *umatrix:** vettore contenente i valori sulla U-Matrix di ogni nodo • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Id del nodo esterno
Descrizione	La funzione, partendo da un nodo esamina i suoi vicini e restituisce l'ID del nodo esterno. La funzione è utilizzata nell'ambito del metodo di post-processing TWL.

void **convertFitsImage** (char *convertedFile, struct Experiment *Exp)

Parametri in input	• **convertedFile:** nome del file di testo generato dalla conversione • **Exp:** struttura con i parametri dell'esperimento
Parametro restituito	• -
Descrizione	Tale funzione si occupa di convertire l'immagine FITS in input in formato testuale riportando, in un file di testo specificato dal parametro **convertedFile**, i valori dei pixel letti. La funzione si occupa inoltre di controllare il corretto utilizzo di uno strato di Kohonen tridimensionale. Qualora l'utente abbia deciso di utilizzarlo su un'immagine bidimensionale, la funzione genererà un'interruzione. Viceversa, imporrà l'uso di tale tipo di strato di output in caso di immagini tridimensionale, a prescindere dal parametro impostato dall'utente.

void **findNeighborhood** (int *NeighborhoodBoundaries, struct Experiment *Exp, double neighborhoodSize, int winner)

Parametri in input	• **NeighborhoodBoundaries:** vettore contenente gli indici per scorrere il vicinato • **Exp:** struttura contente i parametri dell'esperimento • **neighborhoodSize:** diametro attuale del vicinato • **winner:** ID del neurone vincente
Parametro restituito	• -
Descrizione	La funzione si occupa di calcolare gli indici che serviranno a scorrere il vicinato del BMU e a computare gli ID dei neuroni che vi si trovano all'interno.

int **setWinner** (int p, struct Experiment Exp)

Parametri in input	• **p:** indice di un pattern • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• ID del neurone vincente
Descrizione	La funzione si occupa di calcolare le distanze tra i nodi dello strato di output e il pattern **p.** Il neurone a distanza minore sarà considerato vincente e ne verrà restituito l'ID.

void **test_run** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	Funzione che esegue il ciclo di clusterizzazione, ovvero assegna ad ogni pattern il giusto cluster basandosi sulle distanze tra di essi.

void **train** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	Funzione che esegue il ciclo di addestramento. Oltre ad assegnare quindi ad ogni pattern il relativo cluster, modifica i pesi secondo la regola di Kohonen descritta nel capitolo 3.

void **updateNetwork** (int pattern, struct Experiment *Exp, double sigma)

Parametri in input	• **pattern:** pattern attualmente sottoposto alla rete • **Exp:** struttura contente i parametri dell'esperimento • **sigma:** diametro attuale del vicinato
Parametro restituito	• -
Descrizione	La funzione si occupa di eseguire il consueto aggiornamento della rete ad ogni pattern presentato. Calcola il vicinato del neurone vincente e tutti i parametri per l'aggiornamento dei pesi secondo la regola di Kohonen.

void **getUmatrix** (struct Experiment *Exp, double *umatrix)

Parametri in input	• **double *umatrix:** vettore contenente i valori sulla U-Matrix di ogni nodo • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione calcola i valori dei nodi da riportare sulla U-Matrix come gradiente su scala di grigi.

void **gaussianBlur** (struct Experiment *Exp, double *umatrix)

Parametri in input	• **double *umatrix:** vettore contenente i valori sulla U-Matrix di ogni nodo • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione applica il filtro di sfocatura alla U-Matrix

13 APPENDICE B: DESCRIZIONE DEL CODICE SORGENTE E-SOM

Il software è scritto in linguaggio C al fine di consentire una successiva parallelizzazione del codice.
Il progetto è composto dai seguenti file:

- *main.c*
- *esom.h*
- *utils.h*
- *esomVector.h*
- *esomParams.h*
- *esomOutput.h*
- *esom.c*
- *utils.c*
- *esomVector.c*
- *esomParams.c*
- *esomOutput.c*
- *fitsio.h(*)*
- *longnam.h(*)*
- *il.h(*)*
- *Palette.pal(*)*
- *Stilts.jar(*)*

(il file è relativo ad una libreria esterna o ad un file ausiliario)*

Il ***main*** è, ovviamente il file che contiene il flusso principale dell'esecuzione, mostrato nei relativi Flow Chart. Il file ***esomParams*** contiene la definizione delle strutture contenenti i parametri dell'esperimento e i nomi dei file che dovranno essere generati in output. Le funzioni contenute nel file ***esomVector*** si occupano invece della gestione della memoria necessaria all'esecuzione dell'esperimento configurato. Il file ***utils*** contiene funzioni di carattere generale utili durante l'esecuzione del programma. Le funzioni del file ***esomOutput*** sono invece utili alla creazione dell'output sia testuale che grafico fornito dal software. Il file ***esom*** invece contiene le funzioni specifiche relative all'omonimo modello di rete neurale

Nei successivi paragrafi verrà mostrato in dettaglio il contenuto di ognuno dei file sopraccitati.

13.1 ESOMPARAMS

STRUCT EXPERIMENT	
Int useCase	Parametro che indica lo use case selezionato dall'utente
Int nInputNodes	Parametro che indica il numero di nodi dello strato di input
Int nOutputNodes	Parametro che indica il numero di nodi dello strato di output
Int nPatterns	Parametro che indica il numero di pattern del dataset passato in input
Int maxIterations	Parametro che indica il numero massimo di iterazioni che l'algoritmo deve eseguire
Int execIterations	Parametro che tiene traccia delle iterazioni eseguite incrementando ogni qual volta un'iterazione viene portata a termine
Int neighborSize	Parametro che indica il diametro del vicinato iniziale
Double eta0	Parametro che indica il learning rate iniziale
Double eta1	Parametro che indica la soglia minima che il learning rate può raggiungere prima che si interrompa l'algoritmo
Double eta	Parametro che tiene traccia del learning rate attuale durante l'esecuzione dell'algoritmo
Int gridRows	Parametro che indica il numero di righe dello strato di output della rete
Int gridCols	Parametro che indica il numero di colonne dello strato di output della rete
Time_T startTime	Data di inizio dell'esperimento
Time_T endTime	Data di fine dell'esperimento
Int treD	Valore booleano che indica l'utilizzo o meno di uno strato di output tridimensionale
Int normalization	Valore booleano che indica la richiesta o meno di normalizzazione del dataset in

	input
Int nClusters	Parametro che indica il numero di cluster individuati
Int nBMU	Parametro che indica il numero di neuroni dello strato di output risultati BMU
Int expectClst	Numero di cluster attesi
Double sigmaGaussianBlur	Parametro che indica il grado di sfocatura della U-Matrix
Int postProcessCase	Use case selezionato per il l'applicazione di un metodo di post processing della SOM
Int postProcessMethod	Parametro che indica il metodo di clustering da applicare alla SOM ottenuta
Char* inputDataset	Stringa contenente il nome del dataset passato in input
Int datasetFormat	Parametro che indica il formato del dataset
Char* usedDataet	Stringa contenente il nome del dataset utilizzato durante l'esperimento. Nel caso in cui sia necessaria la conversione del dataset infatti, il nome del file passato in input non coinciderà più con quello utilizzato durante l'esperimento. Esempio: InputDataset[] = ***dataset.jpg*** → usedDataset[] = ***dataset.txt***
Char* configurationFile	Stringa contenente il nome dell'eventuale file di configurazione
Char* targetClusterFile	Stringa contenente il nome dell'eventuale file con i cluster attesi per ogni pattern
Int outImageWidth	Parametro che salva, nel caso il dataset sia un'immagine, la sua larghezza
Int outImageHeight	Parametro che salva, nel caso il dataset sia un'immagine, la sua altezza
Int slice	Parametro che indica il numero di slice dell'immagine. In caso di immagine bidimensionale, sarà pari ad 1
Char *commandLine	Stringa contenente la riga di comando immessa dall'utente
Int **Coordinates	Array bidimensionale con un numero di righe pari al numero di nodi che compongono lo strato di output e un numero di colonne pari a 2 o 3 a seconda che si utilizzi rispettivamente uno strato di Kohonen bidimensionale o tridimensionale. Scopo di questa struttura dati e tener traccia delle coordinate matriciali di ogni nodo di output
Int *Winners	Vettore di interi di dimensioni pari al numero di pattern che compongono il dataset. L'indice del vettore coinciderà quindi con l'ID del pattern e alla corrispondente locazione di memoria verrà memorizzato l'ID del neurone vincente per il suddetto pattern.
Int *bmuVictories	Vettore di interi di dimensioni pari al numero di nodi che compongono lo strato di output. L'indice del vettore coinciderà quindi con l'ID del nodo e alla corrispondente locazione di memoria verrà memorizzato il numero di pattern assegnati al suddetto BMU.
Int *ClustersCount	Vettore di interi di dimensioni par al numero di cluster individuati che, per ognuno di essi riporta il numero di pattern assegnati.
Double **Weights	Array bidimensionale con un numero di righe pari al numero di neuroni facenti parte dello strato di output ed un numero di colonne pari al numero di nodi facenti parte dello strato di input. Tale array serve a memorizzare il peso del collegamento tra ogni nodo di output ed ogni nodo di input
Double **Out0	Array bidimensionale con un numero di righe pari al numero di pattern che compongono il dataset. Il numero di colonne è invece pari al numero di nodi dello strato di input, al fine di memorizzare, per ogni pattern le relative features.
Int *ClusterAssigned	Vettore di interi di dimensioni par al numero di nodi che compongono lo strato di output. L'indice del vettore coinciderà con l'ID del nodo e alla corrispondente locazione di memoria verrà memorizzato la label indicante il cluster assegnato.
Double *Activations	Vettore con un numero di righe pari al numero di pattern che per ognuno di essi memorizza il valore di attivazione del relativo neurone vincente.

Tabella 47 – Struct Experiment del modello E-SOM

STRUCT FILENAME	
Char* RES_FILE	Nome del file E***SOM_XXX_results.txt***
Char* CLST_FILE	Nome del file E***SOM_XXX_clusters_count.txt***
Char* LOG_FILE	Nome del file E***SOM_XXX_status.log***
Char* NORM_RES_FILE	Nome del file E***SOM_XXX_normalized_results.txt***

Char* HISTO_IMAGE	Nome del file E***SOM_XXX_histogram.png***
Char* NET_CONF_FILE	Nome del file E***SOM_Train_network_configuration.txt***
Char* CLSTRD_IMAGE	Nome del file E***SOM_XXX_clustered_image.png***
Char* CLSTRD_IMAGE_TXT	Nome del file **ESOM_XXX_clustered_image.txt**
Char* VALIDITY_INDX	Nome del file E***SOM_XXX_validity_indices.txt***
Char* UMATRIX	Nome del file E***SOM_XXX_U_Matrix.png***
Char* KOHONEN_LYR	Nome del file E***SOM_XXX_output_layer.txt***
Char* DATACUBE_ZIP	Nome del file *E**SOMl_XXX_clustered_datacube_image.zip***

Tabella 48 – Struct FileName del modello E-SOM

char* **getTextUseCase** (int UseCase)

Parametri in input	• **UseCase:** use case selezionato dall'utente
Parametro restituito	• Stringa contente lo use case selezionato dall'utente in formato testuale
Descrizione	La funzione prende in input lo use case selezionato in formato numerico e lo converte in formato testuale restituendo tale stringa. Tale funzione risulta utile nella generazione dinamica dei nomi dei file di output.

int **isTable** (char *extension)

Parametri in input	• **extension:** estensione del dataset in input al programma
Parametro restituito	• Valore booleano
Descrizione	La funzione restituisce TRUE se l'estensione del dataset coincide con una di quelle ammesse dal formato *table*.

int **isImage** (char *extension)

Parametri in input	• **extension:** estensione del dataset in input al programma
Parametro restituito	• Valore booleano
Descrizione	La funzione restituisce TRUE se l'estensione del dataset coincide con una di quelle ammesse dal formato *image*.

int **isFitsImage** (char *extension)

Parametri in input	• **extension:** estensione del dataset in input al programma
Parametro restituito	• Valore booleano
Descrizione	La funzione restituisce TRUE se l'estensione del dataset coincide con una di quelle ammesse dal formato *fits_image*.

int **isAscii** (char *extension)

Parametri in input	• **extension:** estensione del dataset in input al programma
Parametro restituito	• Valore booleano
Descrizione	La funzione restituisce TRUE se l'estensione del dataset coincide con una di quelle ammesse dal formato *ASCII*.

void **fileNameGenerator** (struct FileName *Files, int UseCase)

Parametri in input	• **Files:** struttura in cui sono memorizzati i nomi dei file di output • **UseCase:** use case selezionato
Parametro restituito	• -
Descrizione	La funzione si occupa di generare dinamicamente i nomi dei file di output. Come descritto nel capitolo apposito infatti, i nomi dei suddetti file variano in base ai parametri dell'esperimento. Il parametro in input **UseCase,** una volta convertito in formato testuale, andrà posto all'interno del nome del file.

void **checkParameters** (struct Experiment *Exp, int *givenParams)

Parametri in input	• **Exp:** struttura contente i parametri dell'esperimento • **givenParmas:** vettore con valori booleani riguardanti i parametri passati in input
Parametro restituito	•
Descrizione	La funzione controlla che i parametri passati in input al software rispettino i vincoli imposti in fase di progettazione. Il controllo sul passaggio in input del file di configurazione viene effettuato al di fuori della funzione. Il motivo risiede nel fatto che la suddetta funzione viene richiamato a seguito di un'eventuale lettura di tale file, mentre il controllo sullo stesso deve essere effettuato evidentemente prima di una sua apertura.

void **initialize** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione inizializza i parametri dell'esperimento con valori di default.

void **errorReport** (int errorCode, struct Experiment *Exp)

Parametri in input	• **errorCode:** codice dell'errore che si è verificato • **Exp**: struttura con i parametri relativi all'esperimento
Parametro restituito	-
Descrizione	La funzione viene chiamata quando si verifica un'interruzione inaspettata del programma. Ad ogni condizione di uscita forzata è associato un codice che la funzione utilizza per stampare, nel file di output, il relativo messaggio di errore insieme alla riga di comando immessa dall'utente. Successivamente alla creazione del file, viene richiamata la funzione adibita alla deallocazione della memoria occupata.

void **patternCount** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione analizza il dataset contando il numero di pattern contenuti al suo interno. Controlla inoltre che la dimensione dei pattern coincida con il numero di nodi che costituiscono lo strato di input.

void **randomWeights** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione si occupa di generare in maniera casuale i pesi iniziali dei neuroni dello strato di output, normalizzati tra -1 ed 1.

long int **readpatterns** (long int startFile, struct Experiment *Exp)

Parametri in input	• **startFile:** posizione della quale iniziare a leggere il file • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Posizione del cursore nel file al termine della lettura
Descrizione	La funzione si occupa di acquisire e memorizzare i pattern letti all'interno del dataset.

void **readConfigFile** (struct Experiment *Exp, long int *weightsPosition)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Posizione del cursore nel file al termine della lettura
Descrizione	La funzione serve per l'acquisizione dei parametri da un file di configurazione, anziché da linea di comando. Si utilizza quindi solitamente nei casi di Test e Run, oppure in caso si voglia addestrare una rete a partire dai parametri ottenuti da un addestramento precedente (Resume Training). A seconda del valore assunto dal parametro **weightsPosition,** la funzione agirà in maniera differente. Se tale valore è uguale a 0, allora la funzione leggerà, partendo dall'inizio del file, i parametri dell'esperimento. Viceversa, il valore indicherà la posizione nel file dei vettori dei pesi che la funzione dovrà leggere.

void **coordinatesComputation** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione ricava, a partire dall'ID del nodo, le coordinate matriciali dello stesso.

void **normalizeDataset** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura con i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione normalizza le features del dataset in input nell'intervallo [-1, 1].

void **targetCount** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione assicura che il numero di cluster attesi contenuti, all'interno dell'apposito file, sia uguale al numero di pattern del dataset.

13.2 ESOMVECTOR

int** **IntMatrixAllocate** (int rows, int cols)

Parametri in input	• **rows:** numero di righe della matrice • **cols:** numero di colonne della matrice
Parametro restituito	• Matrice di interi
Descrizione	La funzione si occupa di allocare la memoria per una matrice di interi delle dimensioni specificate dai parametri in input

double** **DoubleMatrixAllocate** (int rows, int cols)

Parametri in input	• **rows:** numero di righe della matrice • **cols:** numero di colonne della matrice
Parametro restituito	• Matrice di double
Descrizione	La funzione si occupa di allocare la memoria per una matrice di double delle dimensioni specificate dai parametri in input

void **DoubleMatrixFree** (double **matrix, int rows)

Parametri in input	• **matrix:** matrice da deallocare • **rows:** numero di righe della matrice
Parametro restituito	• -
Descrizione	La funzione si occupa di liberare la memoria allocata per una matrice di double il cui numero di righe è specificato tra i parametri in input

void **IntMatrixFree** (int **matrix, int rows)

Parametri in input	• **matrix:** matrice da deallocare • **rows:** numero di righe della matrice
Parametro restituito	• -
Descrizione	La funzione si occupa di liberare la memoria allocata per una matrice di double il cui numero di righe è specificato tra i parametri in input

void **freeMemory** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione si occupa di deallocare la memoria allocata dinamicamente .

void **allocMemory** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione si occupa di allocare dinamicamente la memoria necessaria all'esecuzione dell'esperimento.

void **newNodeRealloc** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione si occupa di riallocare le strutture dati che lo necessitano quando un nuovo nodo viene aggiunto allo strato di output.

13.3 UTILS

int **fileExist** (char *filename)

Parametri in input	• **filename:** nome del file di cui verificare l'esistenza
Parametro restituito	• valore booleno
Descrizione	La funzione verifica l'esistenza del file passato in input.

int **toAsciiHex** (int x)

Parametri in input	• **x:** valore da convertire
Parametro restituito	• Valore convertito
Descrizione	La funzione converte un numero compreso tra 0 e 15 nel suo corrispettivo esadecimale restituendo il codice ASCII del valore convertito.

void **getColor** (int id, char *color)

Parametri in input	• **id:** indice del colore nella palette • **color:** stringa contenente il colore
Parametro restituito	• -
Descrizione	La funzione converte un colore in formato RGB letto all'interno di una palette, in formato esadecimale. Il colore da convertire è indicato dal parametro **id,** ovvero l'indice dello stesso nella palette.

char* **getExtension** (char *File)

Parametri in input	• **File:** nome del file in input
Parametro restituito	• Estensione del file
Descrizione	La funzione analizza il nome del file restituendone l'estensione.

void **jumpHeader** (FILE *File)

Parametri in input	• **File:** file sul quale la funzione dovrà lavorare
Parametro restituito	• -
Descrizione	La funzione muove il cursore, del file passato in input, nel punto immediatamente successivo ad eventuali header lines.

double **doubleRangeRand** (const double a, const double b)

Parametri in input	• **a:** estremo inferiore dell'intervallo entro il quale generare un numero casuale • **b:** estremo superiore dell'intervallo entro il quale generare un numero casuale
Parametro restituito	• Numero casuale compreso nell'intervallo specificato
Descrizione	La funzione serve a generare un numero casuale all'interno di un intervallo specificato dai parametri in input.

int **max** (double x, double y)

Parametri in input	• **x:** primo valore • **y:** secondo valore
Parametro restituito	• Massimo tra i due valori
Descrizione	La funzione calcola e restituisce il massimo tra i due valori passati in ingresso.

int **min** (double x, double y)

Parametri in input	• **x:** primo valore • **y:** secondo valore
Parametro restituito	• Minimo tra i due valori
Descrizione	La funzione calcola e restituisce il minimo tra i due valori passati in ingresso.

void **help** ()

Parametri in input	• **-**
Parametro restituito	• -
Descrizione	Funzione che mostra a video una guida sull'uso del software.

void **debugFitsImage** (char *inputFits, char *outputFits)

Parametri in input	• **inputFits:** nome del file FITS in input • **outputFits:** nome del file FITS in output
Parametro restituito	• -
Descrizione	La funzione legge l'immagine FITS indicata dal primo parametro e copia il contenuto all'interno del secondo. Tale procedimento ha lo scopo di verificare la corretta lettura del file da parte del programma.

13.4 SOMOUTPUT

void **writeConfig** (char *configFile, struct Experiment Exp)

Parametri in input	• **configFile:** nome del file di configurazione da scrivere • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione si occupa di scrivere il file E***SOM_Train_network_configuration.txt.***

void **writeResults** (char *clusterFile, struct Experiment Exp, int normClusters)

Parametri in input	• **clusterFile:** nome del file da generare • **Exp:** struttura contenente i parametri dell'esperimento • **normClusters:** flag che indica il tipo di file da generare
Parametro restituito	• -
Descrizione	La funzione scrive, a seconda del valore assunto dal parametro **normClusters,** il file ***ESOM_XXX_clusters.txt*** o il file ***ESOM_XXX_normalized_clusters.txt.***

void **writeStatus** (char *statusFile, struct Experiment Exp)

Parametri in input	• **statusFile:** nome del file da generare • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	Funzione che si occupa di scrivere il file E***SOM_XXX_status.log.***

void **writeClusters** (char *countFile, struct Experiment Exp)

Parametri in input	• **countFile:** nome del file da generare • **Exp:** struttura con i parametri dell'esperimento
Parametro restituito	• -
Descrizione	Funzione che si occupa di scrivere il file E***SOM_XXX_clusters_count.txt.*** La percentuale di associazione tra il numero di pattern assegnati ad ogni cluster e il numero di pattern totali, è calcolato all'interno della funzione stessa.

void **writeOutputLayer** (char *kohonenFile, struct Experiment Exp)

Parametri in input	• **kohonenFile:** nome del file da generare • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	Funzione che scrive il file E***SOM_XXX_output_layer.txt.***

void **writeValidityIndx** (char *validityFile, struct Experiment Exp)

Parametri in input	• **validityFile:** nome del file da generare • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	Funzione che scrive il file E***SOM_XXX_validity_indices.txt.***

void **drawUmatrix** (char *inFileName, char *outFileName, struct Experiment Exp)

Parametri in input	• **inFileName:** nome del file da passare in input al comando STILTS • **outFileName:** nome del file generato dal comando STILTS • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione utilizza il comando ***plot2d*** di STILTS per disegnare la U-Matrix relativa alla SOM. Maggiori informazioni sul comando sono disponibili al Paragrafo 15.3.2.

void **drawClusteredImage**(char *outFileName,struct Experiment Exp, int slice)

Parametri in input	• **outFileName:** nome dell'immagine generata • **Exp:** struttura contenente i parametri dell'esperimento • **slice:** slice dell'immagine cui la funzione fa riferimento
Parametro restituito	• -
Descrizione	La funzione genera, tramite le routine della libreria DevIL, l'immagine ***SOM_XXX_clustered_image.png.*** In base al parametro **slice,** la funzione sarà in grado di calcolare il pattern da cui iniziare la scrittura dell'immagine, al fine di generare l'output relativa alla slice passata in input, generando inoltre il corretto prefisso per il nome.

void **drawHistogram** (struct Experiment Exp, char *inFileName, char *outFileName)

Parametri in input	• **Exp:** struttura con i parametri dell'esperimento • **inFileName:** nome del file da passare in input al comando STILTS • **outFileName:** nome del file generato dal comando STILTS
Parametro restituito	• -
Descrizione	La funzione utilizza il comando ***plothist*** per disegnare un istogramma che riporta il numero di pattern assegnati ad ogni cluster. Maggiori informazioni riguardo il comando sono disponibili al Paragrafo 15.3.2.

void **writeClusteredImage**(char *outFileName,struct Experiment Exp)

Parametri in input	• **outFileName:** nome dell'immagine generata • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione genera il file E***SOM_XXX_clustered_image.txt*.**

13.5 ESOM

void **getQuantizationError** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione restituisce l'errore di quantizzazione relativo alla SOM.

void **getDaviesBouldin** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione restituisce l'indice di Davies-Bouldin.

double **distPatterns** (int pattern1, int pattern2, struct Experiment Exp)

Parametri in input	• **pattern1:** primo pattern da cui misurare la distanza • **pattern2:** secondo pattern da cui misurare la distanza • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Distanza euclidea tra i due pattern
Descrizione	La funzione si occupa di calcolare la distanza euclidea tra due pattern utilizzando, come coordinate, le relative features.

double **distPatternNode** (int pattern, int node, struct Experiment Exp)

Parametri in input	• **pattern:** pattern da cui misurare la distanza • **node:** nodi di output da cui misurare la distanza • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Distanza euclidea tra il nodo e pattern in input
Descrizione	La funzione si occupa di calcolare la distanza euclidea tra un pattern ed un nodo dello strato di output utilizzando, come coordinate, rispettivamente le features e i pesi.

double **distNodes** (int node1, int node2, struct Experiment Exp)

Parametri in input	• **node1:** primo nodo da cui misurare la distanza • **node2:** secondo nodo da cui misurare la distanza • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Distanza euclidea tra i due nodi
Descrizione	La funzione si occupa di calcolare la distanza euclidea tra due nodi dello strato di output utilizzando, come coordinate, i relativi pesi.

double **distNodeCentroids** (int node, int centroid, struct Experiment Exp)

Parametri in input	• **node:** nodo da cui calcolare la distanza • **centroid:** centroide da cui calcolare la distanza • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Distanza euclidea tra un nodo ed un centroide.
Descrizione	La funzione calcola la distanza tra un nodo ed un centroide.

double **distPatternCentroids** (int pattern, int centroid, struct Experiment Exp)

Parametri in input	• **pattern:** pattern da cui calcolare la distanza • **centroid:** centroide da cui calcolare la distanza • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Distanza euclidea tra un nodo ed un centroide.
Descrizione	La funzione calcola la distanza tra un pattern ed un centroide.

double **distCentroids** (int centroid1, int centroid2, struct Experiment Exp)

Parametri in input	• **centroid1:** primo centroide da cui calcolare la distanza • **centroid2:** secondo centroide da cui calcolare la distanza • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Distanza euclidea tra due centroidi.
Descrizione	La funzione calcola la distanza tra due centroidi.

void **convertFitsImage** (char *convertedFile, struct Experiment *Exp)

Parametri in input	• **convertedFile:** nome del file di testo generato dalla conversione • **Exp:** struttura con i parametri dell'esperimento
Parametro restituito	• -
Descrizione	Tale funzione si occupa di convertire l'immagine FITS in input in formato testuale riportando, in un file di testo specificato dal parametro **convertedFile**, i valori dei pixel letti. La funzione si occupa inoltre di controllare il corretto utilizzo di uno strato di Kohonen tridimensionale. Qualora l'utente abbia deciso di utilizzarlo su un'immagine bidimensionale, la funzione genererà un'interruzione. Viceversa, imporrà l'uso di tale tipo di strato di output in caso di immagini tridimensionale, a prescindere dal parametro impostato dall'utente.

int **setWinner** (int p, struct Experiment Exp)

Parametri in input	• **p:** indice di un pattern • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• ID del neurone vincente
Descrizione	La funzione si occupa di calcolare le distanze tra i nodi dello strato di output e il pattern **p.** Il neurone a distanza minore sarà considerato vincente e ne verrà restituito l'ID.

void **test_run** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	Funzione che esegue il ciclo di clusterizzazione, ovvero assegna ad ogni pattern il giusto cluster basandosi sulle distanze tra di essi.

int **findTwoWinners** (int pattern, int *BMUs, struct Experiment Exp)

Parametri in input	• **pattern:** indice di un pattern • **BMUs:** array per il salvataggio dei risultati • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• ID dei primi due BMU
Descrizione	La funzione si occupa di calcolare le distanze tra i nodi dello strato di output e il pattern **p.** I due neuroni a distanza minore saranno considerati vincenti e ne verrà restituito l'ID.

void **setNewNode** (int node, int pattern, struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento • **node:** ID del nodo di cui settare i pesi • **pattern:** pattern sui cui valori inizializzare il nodo
Parametro restituito	• -
Descrizione	La funzione inizializza i pesi di un nodo basandosi sulle feature del pattern passato in input.

void **getActivations** (int node, int pattern, struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento • **node:** id del nodo di cui calcolare l'attivazione • **pattern:** pattern per cui calcolare l'attivazione del nodo passato in input
Parametro restituito	• -
Descrizione	La funzione calcola, dati un nodo ed un pattern, l'attivazione del primo nei confronti del secondo.

void **setConnection** (int node1, int node2, int pattern, struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento • **node1:** primo nodo con cui instaurare la connessione • **node2:** secondo nodo con cui instaurare la connessione • **pattern:** pattern correntemente analizzato
Parametro restituito	• -
Descrizione	La funzione setta la connessione tra i due nodi passati in input. Il calcolo del peso della suddetta connessione dipende dalle attivazione dei nodi per il pattern passato in input.

void **getSumOfAct** (int node, int pattern, struct Experiment Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento • **node:** ID del nodo di cui ricavare le attivazione del vicinato • **pattern:** pattern correntemente analizzato
Parametro restituito	• -
Descrizione	La funzione calcola la somma delle attivazione di tutto il vicinato del nodo passato in input.

int **assignCluster** (int node, int **Connections, struct Experiment *Exp)

Parametri in input	• **node:** indice di un nodo dello strato di output • **int **Connections:** matrice delle connessioni tra nodi • **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• Label del cluster assegnato al nodo
Descrizione	La funzione, partendo da un nodo e seguendo le sue connessioni, assegna le relative label del cluster di appartenenza

void **getCentroids** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione, a partire dai cluster individuati, calcola i relativi centroidi.

void **train** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	Funzione che esegue il ciclo di addestramento. Oltre ad assegnare quindi ad ogni pattern il relativo cluster, modifica i pesi secondo la regola di Kohonen descritta nel capitolo 3.

void **pruneWeakestConn** (struct Experiment *Exp)

Parametri in input	• **Exp:** struttura contenente i parametri dell'esperimento
Parametro restituito	• -
Descrizione	La funzione taglia la connessione più debole della rete.

14 APPENDICE C: DESCRIZIONE DEL CODICE SORGENTE E-TREE

14.1 FLUSSO DI TRAINING

1. **DataCache data -** istanza della classe dell'input in fase di Train;
2. **data.ReadFromDisk(data_file) –** carica gli input pattern per la fase di Train da un file ASCII;
3. **data.CalcClassStat() –** calcolo delle statistiche sull'input: numero di classi, occorrenze per ogni classe
4. **data.Normalize() –** gli input pattern sono normalizzati come una distribuzione con media 0 e varianza 1;
5. **Etree etree(&data) –** istanza dell'albero;
6. **etree.SetParameters(divisionThreshold, divisionFactor, eta0, sigma0, tau1, tau2, bdecay) –** inizializza i parametri come indicato dal file esterno caricato;
7. **etree.Initialize() –** inizializza l'albero con un vettore nel punto medio del dataset
8. **while (etree.HasTreeGrown()) –** tale funzione rappresenta il criterio d'arresto. Se l'albero è cresciuto meno di una soglia prefissata (5%) rispetto all'ultima iterazione, l'addestramento viene fermato;
    a. **etree.SingleRound() –** calcola un round dell'algoritmo di training che consiste dei seguenti passi:
        i. **datavecs->Shuffle() –** mescola gli input pattern;
        ii. **while(!last)**
            1. **const double *dv = datavecs->GetNextVector(last) –** carica il pattern successivo
            2. **SingleIteration(dv) –** addestra l'albero con un singolo pattern tramite i seguenti passi:
                a. **int bmu = FindBMU(datavec) –** trova il BMU
                b. **UpdateNodes(bmu,datavec) –** aggiornamento dei nodi
                    i. **weightbase = η_0*exp(-round/ τ_2);**
                    ii. **weight = weightbase * CalculateUpdateFactor(0)**. La funzione CalculateUpdateFactor(distance) calcola:
                        1. **$\sigma = \sigma_0$* exp(-(double)round/ τ_1);**
                        2. **foo = exp(-pow(distance, 2) / (2*pow(sigma, 2)));**
                    iii. **nodes[BMU].UpdateTowards(targetvector,weight):**
                        1. **for(int j=0; j<dim; j++)**
                            a. **proto[j] += weight*(targetvector[j] - proto[j]);** Rappresenta l'aggiornamento dei pesi secondo il modello SOM;
                    iv. **if (nodes[BMU].GetParent()!=ROOT_LOOP) –** aggiorna ricorsivamente il vicinato del BMU
                        1. **UpdateNode(nodes[BMU].GetParent(),BMU,BMU,datavec,1);**
            3. **if(nodes[bmu].IncrementBMU() >= divisionThreshold**) – splitting del nodo
        iii. **DecayBMUcounts()** - moltipilica tutti I contatori di occorrenze delle foglie come fattore di regolarizzazione;
9. **for(int i=0; i<kmeanscount; i++)**
    a. **etree.KmeansAdjust() –** rifinitura della posizione delle foglie tramite K-Means, piazzando I nodi foglia nel loro centro di massa:
        i. **CreateIndex() –** costruisce l'indice delle informazioni trovando tutti I BMU come risultato del Train;
        ii. Inizializza le nuove locazioni a 0
        iii. Aggiorna le nuove locazioni con I nodi figlio;
        iv. Dividi ogni nuova locazione per il numero di nodi per ottenere il centro di massa;
        v. Setta la nuova locazione per tutti I nodi, basata sulla normalizzazione con il centro di massa;
        vi.
10. **treefile << etree –** salva l'albero ottenuto dall'addestramento su un file esterno che sarà usato per la fase di Test;
11. **etree.PrintNewickTree(newickoutfile) –** stampa albero e dati nel format Newick

14.2 FLUSSO DI TEST

1. **DataCache train,test –** istanze delle classi dell'input per le fasi di Train e Test;
2. **train.ReadFromDisk(data_file), test.ReadFromDisk(test_input) -** carica gli input pattern per le fasi di Train e Test da un file ASCII;
3. **train.CalcClassStat(), test.CalcClassStat() -** calcolo delle statistiche sull'input (numero di classi, occorrenze per ogni classe) per le fasi di Train e test;
4. **train.Normalize(), test.Normalize() –** gli input pattern, per le fasi di Train e Test, sono normalizzati come una distribuzione con media 0 e varianza 1;
5. **treefile >> etree –** carica in memoria la struttura dell'albero da usare per il Test;
6. **testOut.open(test_out) –** aperture del file di output del Test
7. **etree.CreateIndex() –** prepara l'indice di classificazione da usare durante la classificazione;
    a. **ClearIndex() –** resetta le etichette dell'albero;
    b. Su tutti i nodi dell'albero:
        i. **vector<int>drEvil;**
        ii. **labels.push_back(drEvil);**
    c. Per tutti i pattern di addestramento
        i. **int ind = FindBMU(datavecs->getVectorNumber(i)) –** trova il BMU del pattern
        ii. **labels[ind].push_back(i) –** aggiungi il BMU alla classe del vettore associate al nodo foglia
    d. **testStatistics test_statM**
    e. **test_stat_patNumber = test.GetSize();**
    f. **test_stat_totalClassNumber = test.GetLabelInfoSize();**
    g. Per tutti i **cl** in **train.GetLabelInfoSize()**
        i. **test_stat_class_labels_push_back(train.GetLabelInfo(cl));**
        ii. **test_stat_occurrences.push_back(train.GetOccurrences(cl));**
    h. **patternStatistics *pat_stat;**
    i. **pat_stat = new patternStatistics[test.GetSize()];**
    j. Per tutti i k pattern di Test:
        i. **pat_stat[k].targetLabel = test.GetLabel(k); -** labal della classe del pattern corrente
        ii. **result = etree.GetClassification(test.GetVectorNumber(k));**
        iii. **BMU = FindBMU(queryvec)**
        iv. **result = GetIndexLabels(BMU);**
        v. **pat_stat[k].classResult = result –** salva la classifiicazione del pattern corrente
        vi. **DataCache classOut –** istanzia la classe dell'output della fase di classificazione
        vii. Per tutti I j in result.size() **classOut.SetLabel(train.GetLabel(result[j])**
        viii. **classOut.calcClassStat();**
        ix. **pat_stat[k].BMUnumber = classOut.GetLabelSize();**
        x. **pat_stat[k].classNumber = classOut.GetLabelInfoSize() –** salva le statistiche di output per il pattern corrente (numero di BMU e classe assegnata)
        xi. Per tutti I c in **classOut.GetLabelInfoSize()**
            1. **pat_stat[k].class_labels.push_back(classOut.GetLabelInfo(c));**
            2. **pat_stat[k].occurrences.push_back(classOut.GetOccurrences(c));**
            3. **testOut << "\nclass " << classOut.GetLabelInfo(c);**
            4. **testOut << classOut.GetOccurrences(c);**

15 APPENDICE D: LIBRERIE ESTERNE

Di seguito vengono descritte le librerie esterne utilizzate nella progettazione del codice:

15.1 CFITSIO

I file sorgenti relative a questa libreria sono:

- ***fitsio.h***
- ***longnam.h***

Le routine utilizzate sono:

int **fits_open_image** (fitsfile **fptr, char *filename, int iomode, int *status)

Parametri in input	• **fptr:** puntatore al file FITS • **filename:** nome del file da aprire • **iomode:** modalità in cui aprire il file • **status:** flag per la corretta esecuzione della funzione
Parametro restituito	• Errore in caso di esecuzione non corretta
Descrizione	Funzione che apre il file specificato dal parametro **filename** nella modalità specificata dal parametro **iomode** (READONLY o READWRITE) creando un puntatore al file fits specificato dal parametro **fptr**. A differenza delle altre funzioni dedite all'apertura di un file, questa sposta il cursore direttamente alla parte del file relativa ad un'immagine, se esiste. Il parametro **status** indica la corretta esecuzione della funzione, viene infatti settato ad 1 nel momento in cui si verifica un errore.

int **fits_get_img_dim** (fitsfile *fptr, int *naxis, int *status)

Parametri in input	• **fptr:** puntatore al file FITS • **naxis:** numero di assi dell'immagine • **status:** flag per la corretta esecuzione della funzione
Parametro restituito	• Errore in caso di esecuzione non corretta.
Descrizione	Funzione che restituisce il numero di assi (dimensioni) dell'immagine settando il parametro **naxis**. Il parametro **status** indica la corretta esecuzione della funzione, viene infatti settato ad 1 nel momento in cui si verifica un errore.

int **fits_get_img_size** (fitsfile *fptr, int maxdim, long *naxes, int *status)

Parametri in input	• **fptr:** puntatore al file FITS • **maxdim:** numero massimo di assi • **naxes:** vettore contenente le dimensioni degli assi • **stauts:** flag per la corretta esecuzione della funzione
Parametro restituito	• Errore in caso di esecuzione non corretta.
Descrizione	Funzione che resituisce la dimensione di ognuno degli assi inserendo tali valori nel vettore **naxes**. Il parametro **maxdim** serve ad indicare alla funzione il numero massimo di assi di cui deve restituire la dimensione. Il parametro **status** indica la corretta esecuzione della funzione, viene infatti settato ad 1 nel momento in cui si verifica un errore.

fits_read_pix (fitsfile *fptr, int datatype, long *fpixel, LONGLONG nelements, DTYPE *nulval, DTYPE *array, int *anynul, int *status)

Parametri in input	• **fptr:** puntatore al file FITS • **datatype:** tipo di dato letto • **fpixel:** vettore contenente i pixel da cui partire con la lettura • **nelements:** numero totale di pixel da leggere • **nulval:** valore da considerare come pixel nullo • **array:** vettore contenente i valori dei pixel letti • **anynul:** vettore che indica i pixel considerati nulli • **status:** flag per la corretta esecuzione della funzione
Parametro restituito	• -
Descrizione	La funzione si occupa di leggere i pixel dell'immagine inserendoli nel vettore **array** al fine poi di trattarli successivamente come si ritiene opportuno. I parametri in ingresso alla funzione determinano quanti e quali pixel saranno letti. Il numero totale di pixel da leggere è determinato dal parametro in ingresso **nelements** mentre i pixel da cui partire con la lettura sono indicati dai valori contenuti nel vettore **fpixel**. Tale vettore ha infatti dimensione pari al numero di assi ed il valore contenuto in ognuna delle locazioni indica da quale pixel iniziare a leggere per ciascuno degli assi. I parametri **nulval** e **anynul** riguardano la trattazione dei pixel indefiniti. In particolare, il primo parametro può essere usato per imporre alla funzione il valore da considerare per stabilire se un pixel è indefinito o meno. Il secondo è un vettore di dimensione pari al numero di pixel che, per ognuno di essi indica se è indefinito (valore uguale ad 1) o meno (valore uguale a 0). Infine il parametro **datatype** indica il tipo di dato che si andrà a leggere.

fits_close_file (fitsfile *fptr, int *status)

Parametri in input	• **fptr:** puntatore al file FITS • **status:** flag per la corretta esecuzione della funzione
Parametro restituito	• -
Descrizione	Funzione che chiude il file FITS precedentemente aperto.

15.2 DEVIL

I file sorgenti relativi a questa libreria sono:

- ***il.h***

Prima di passare alla trattazione delle routine utilizzate è bene specificare che la libreria DevIL definisce un sistema di tipi di dato interno al fine di garantire la massima portabilità. Il nome dei nuovi tipi di dato definiti viene ottenuto semplicemente anteponendo il prefisso "IL" al tipo di dato.

Le routine utilizzate sono:

ilInit ()

Parametri in input	• -
Parametro restituito	• -
Descrizione	Funzione che inizializza la libreria IL

ILboolean **ilLoadImage** (const char *FileName)

Parametri in input	• **FileName:** nome dell'immagine da caricare
Parametro restituito	• Valore booleano relativo all'esito della funzione
Descrizione	Funzione che carica l'immagine specificata dal parametro in input

Ilint **ilGetInteger** (ILenum Mode)

Parametri in input	• **Mode:** tipo di informazione richiesta
Parametro restituito	• Valore relativo all'informazione richiesta
Descrizione	La funzione è utilizzata per estrarre informazioni da un'immagine caricata precedentemente. Il tipo di informazione è decretata dal parametro in input **Mode**. Di seguito è mostrato in maniera più specifica, l'uso di questa funzione fatto all' interno del programma: • ***ilGetInteger*** *(IL_IMAGE_WIDTH)* : larghezza dell' immagine; • ***ilGetInteger*** *(IL_IMAGE_HEIGHT)* : lunghezza dell' immagine; • ***ilGetInteger*** *(IL_IMAGE_FORMAT)*: formato dell'immagine, dove per formato intendiamo lo spazio dei colori, i cui possibili valori sono: - ***IL_LUMINANCE:*** *bianco e nero* - ***IL_RGB:*** tre canali per ogni pixel nell'ordine rosso-verde-blu - ***IL_BGR:*** tre canali per ogni pixel nell'ordine blu-verde-rosso - ***IL_RGBA:*** come ***IL_RGB*** ma con l'aggiunta del canale per l'opacità - ***IL_BGRA:*** come **IL_BGR** ma con l'aggiunta del canale per l'opacità

ILubyte* **ilGetData** ()

Parametri in input	• -
Parametro restituito	• Vettore dei valori dei pixel letti
Descrizione	Legge i pixel di un'immagine precedentemente caricata e li salva all'interno di un vettore restituito come parametro di output

ILboolean **ilSetData** (ILvoid *Data)

Parametri in input	• **Data:** vettore dei valori dei pixel da scrivere
Parametro restituito	• Valore booleano relativo all'esito della funzione
Descrizione	Funzione speculare a ***ilGetData.*** Prende infatti in input un vettore e lo scrive all'interno dell'immagine attualmente caricata.

ILboolean **ilTexImage** (ILuint Width, ILuint Height, ILuint Depth, ILubyte Bpp, ILenum Format, ILenum Type, ILvoid *Data)

Parametri in input	• **Width:** larghezza dell'immagine • **Height:** altezza dell'immagine • **Depth:** profondità • **Bpp:** numero di byte per pixel • **Format:** formato dell'immagine • **Type:** tipo di immagine • **Data:** vettore dei valori dei pixel
Parametro restituito	• Valore booleano relativo all'esito della funzione
Descrizione	E' utilizzata per settare alcuni parametri dell'immagine attualmente caricata.

ILvoid **ilGenImages** (ILsizei Num, ILuint *Images)

Parametri in input	• **Num:** numero di identificativi di immagini da generare • **Images:** vettore contenente gli identificativi delle immagini generate
Parametro restituito	• -
Descrizione	Funzione utilizzata per generare uno o più identificativi da utilizzare per riferirsi ad una o più immagini

ILvoid **ilBindImage** (ILuint Image)

Parametri in input	• **Image:** identificativo di un'immagine
Parametro restituito	• -
Descrizione	Funzione collega l'identificativo passato in input all'immagine attualmente caricata

ILboolean **ilSaveImage** (const char *FileName)

Parametri in input	• **FileName:** nome dell'immagine in output
Parametro restituito	• Valore booleano relativo all'esito della funzione
Descrizione	Funzione che salva l'immagine caricata con il nome specificato dal parametro in input

ILvoid **ilDeleteImages** (ILsizei Num, ILuint *Images)

Parametri in input	• **Num:** numero di identificativi di immagini da cancellare • **Images:** vettore contenente gli identificativi delle immagini da cancellare
Parametro restituito	• -
Descrizione	Funzione utilizzata per cancellare uno o più identificativi di immagini e tutti i dati ad essi associati

ILboolean **ilEnable** (ILenum Mode)

Parametri in input	• **Mode:** tipo operazione da abilitare
Parametro restituito	• Valore booleano relativo all'esito della funzione
Descrizione	La funzione è utilizzata per abilitare alcune operazione a seconda del parametro fornito in input. In particolare, all'interno del programma, è utilizzata per abilitare la sovrascrittura fi file esistenti: ***IlEnable*** (IL_FILE_OVERWRITE)

ILboolean **ilLoadPal** (char * FileName)

Parametri in input	• **FileName:** nome del file da caricare
Parametro restituito	• Valore booleano relativo all'esito della funzione
Descrizione	La funzione è utilizzata per caricare una palette di colori da un file esterno

15.3 FILE AUSILIARI

15.3.1 Palette.pal

E' una palette di colori utilizza nella generazione di immagini clusterizzate e plot. Nella palette ogni colore è indicizzato e il colore con cui verrà rappresentato ogni cluster sarà quello il cui indice nella palette coincide con l'etichetta del cluster stesso. La palette è costruita in modo da non contenere ripetizioni di colori, ciò assicura che ogni cluster sarà rappresentato con un colore differente. Questo meccanismo consente di migliorare le prestazioni del programma consentendo un notevole risparmio computazionale a discapito però della gamma di colori utilizzabile.

15.3.2 Stilts.jar

Di seguito sono mostrati i moduli della libreria Stilts utilizzati all' interno del programma.

tcopy

E' uno strumento che permette di copiare dati tabulari permettendo quindi la conversione dei dati stessi, scopo per cui è utilizzato all' interno del programma. Il comando è utilizzato specificando i seguenti parametri:

PARAMETRI TCOPY	
in	Nome del file in input
out	Nome del file in output
ifmt	Formato file in input

Tabella 49 – Parametri comando tcopy

plothist

E' uno strumento che permette di generare istogrammi a partire dai dati contenuti in una colonna di un file passato in ingresso. Il comando è utilizzato specificando i seguenti parametri:

PARAMETRI PLOTHIST	
in	Nome del file in input
out	Nome del file in output
ifmt	Formato del file in input
xdata	Coordinata sull'asse delle x
xhi	Valore massimo dell'asse delle x
xlo	Valore minimo dell'asse delle x
xlabel	Etichetta assegnata all'asse delle x
ylabel	Etichetta assegnata all'asse delle y

Tabella 50 – Parametri comando plothist

plot2d / plot3d

Sono strumenti che permettono di generare dei grafici, rispettivamente i due e tre dimensioni, posizionando dei punti su di essi in base alle coordinate lette all'interno di un file. Il comando è utilizzato specificando i seguenti parametri:

PARAMETRI PLOT2D / PLOT3D	
In	Nome del file in input. All'interno dello stesso comando, è possibile, specificando un suffisso al parametro leggere da file differenti i dati che poi andranno a far parte dello stesso file di output.
Out	Nome del file in output
ifmt	Formato del file in input
xdata	Coordinata sull'asse delle x
ydata	Coordinata sull'asse delle y
zdata	Coordinate sull'asse delle z (solo nel comando ***plot3d***)
auxdata	Dati da plottare sull'asse ausiliario
auxshader	Metodo di visualizzazione dei valori dell'asse ausiliario
Xhi	Valore massimo dell'asse delle x
Xlo	Valore minimo dell'asse delle x
yhi	Valore massimo dell'asse delle y
ylo	Valore minimo dell'asse delle y
zhi	Valore massimo dell'asse delle z (solo nel comando ***plot3d***)
zlo	Valore minimo dell'asse delle z (solo nel comando ***plot3d***)
xlabel	Etichetta assegnata all'asse delle x
xpix	Larghezza dell'immagine in output
ypix	Altezza dell'immagine in output
legend	Valore booleano che indica l'affiancamento o meno di una legenda al grafico
title	Titolo da porre al di sopra del grafico
shape	Forma da utilizzare per i punti da posizionare sul grafico
subset	Permette di definire un'espressione che consente di selezionare solo alcuni punti

Tabella 51 – Parametri comando plot2d / plot3d

16 RINGRAZIAMENTI

Ringrazio tutte le persone che mi hanno aiutato nella stesura di questo lavoro. Tutte le persone che mi sono state vicino e che mi hanno sostenuto nei momenti di difficoltà.

Un ringraziamento particolare per il Dott. Massimo Brescia, il Prof. Giuseppe Longo e tutti i ragazzi del progetto DAME per la splendida opportunità di lavoro che mi hanno offerto e per ciò che grazie a loro ho imparato in questi mesi.

Ringrazio inoltre il Prof. Roberto Prevete per gli insegmenti ricevuti durante il percorso accademico e per l'aiuto fornito nella stesura dell'eleborato di tesi.

Grazie anche alla mia famiglia e alle persone a me più care, senza le quali non sarei dove sono ora.