Università degli Studi di Napoli Federico II

Facoltà di Scienze MM. FF. NN

Corso di Laurea in Informatica

Tesi di Laurea Triennale Sperimentale

Integrazione di due nuove tecniche di apprendimento automatico per il Progetto DAME

Relatori

Dott.ssa Anna Corazza

Dott. Massimo Brescia

Candidato

Marisa Guglielmo

50/524

Anno Accademico 2009-2010

Indice

INDICE 3

ELENCO DELLE FIGURE 5

ELENCO DELLE TABELLE 7

INTRODUZIONE 8

CAPITOLO 1. IL PROGETTO DAME (DATA MINING & EXPLORATION) 10

1.1 Descrizione generale 10
1.2 Architettura della suite 11
1.2.1 Front-End 12
1.2.2 FrameWork 13
1.2.3 Driver 14
1.2.4 Registry & DataBase 18
1.2.5 Data Mining Model 21
1.3 Funzionalità della suite 22
1.3.1 Classificazione 22
1.3.2 Regressione 23
1.4 Modelli della suite DAME 24

CAPITOLO 2.TECNOLOGIE DI BASE 27

2.1 La libreria NExTII (Neural ExTractor) 27
2.1.1 Reverse Engineering 28
2.1.2 La struttura di NExTII 29
2.1.2.1 Le classi Image e Pixel 31
2.1.2.2 La classe Pattern 32
2.1.2.3 La classe UnsupervisedNN 33
2.1.2.4 La classe ClusteringNN 35
2.1.2.5 La classe FrozenMorphologyClusteringNN 36
2.1.2.6 La classe SOMClusteringNN 37
2.1.2.7 La classe MultiLayerClusteringNN 38
2.2 Il componente Data Mining Model (DMM) 39
2.2.1 Il Bridge Pattern 40
2.2.2 La struttura del DMM 41
2.2.3 Il componente DmmParams 45
2.3 Il componente Data Mining Plugin (DMPlugin) 47

CAPITOLO 3. SVILUPPO DELLA SOLUZIONE ADOTTATA 48

3.1 Estensione di NExTII 48
3.1.1 La classe ImageTable 48
3.1.2 La classe NextControl 49
3.1.3 Il controllo di NExTII: NextMain 51
3.2 Estensione del DMM 56
3.2.1 Adattamento della soluzione al Bridge Pattern 57
3.2.2 Livello di astrazione: le classi Unsupervised e Clustering 57
3.2.3 Livello di implementazione: il modello SOM 59
3.2.3.1 Il modello GSOM 60
3.2.3.2 Il modello CSOM 66
3.3 Multi Layer Clustering Plugin (MLC Plugin) 67
3.3.1 Introduzione 67
3.3.2 Lo sviluppo del costruttore 67

CAPITOLO 4. IL TESTING 72

4.1 Testing dei componenti 72
4.2 Caratteristiche del testing 72
4.3 Classi da verificare durante il testing 73
4.3.1 Test Cases 73

CONCLUSIONI 80
RIFERIMENTI BIBLIOGRAFICI 82

Elenco delle Figure

Figura 1 – Architettura generale della suite DAME. 11
Figura 2 – Interfaccia Utente della Suite DAME. 12
Figura 3 – Esempio di comunicazione tra FE e FW. 13
Figura 4 – Funzioni del Driver. 15
Figura 5 – Schema di traduzione del Driver. 15
Figura 6 – Chioma della cometa Hale-Bopp. 16
Figura 7 – Header dell'immagine FITS della cometa Hale-Bopp in Figura 6. 17
Figura 8 – Schema generale di un file VOTable. 18
Figura 9 – Modello ER del REDB. 20
Figura 10 – Prinicipio del modello SVM 24
Figura 11 – Esempio di immagine per l'applicazione del modello SVM. 25
Figura 12 – Risultati dell'applicazione del modello SVM 25
Figura 13 – Struttura di una rete neurale MLP 26
Figura 14 – La gerarchia delle UnsupervisedNN. 29
Figura 15 – Relazione tra ClusteringNN e MultiLayerClusteringNN. 30
Figura 16 – Relazione tra Image e Pixel 31
Figura 17 – La classe Image 32
Figura 18 – La classe Pixel 32
Figura 19 – La classe Pattern 33
Figura 20 – Esempio di estrazione di un pattern di un'immagine multi-banda 33
Figura 21 – La classe UnsupervisedNN 35
Figura 22 – La classe ClusteringNN 36
Figura 23 – La classe FrozenMorphologyClusteringNN 36
Figura 24 – Tipi di ForzenMorphologyClusteringNN 37
Figura 25 – La classe SOMClusteringNN 38
Figura 26 – La classe MultiLayerClusteringNN 39
Figura 27 – Struttura del Bridge Pattern. 40
Figura 28 – Diagramma UML del DMM. 42
Figura 29 – Abstraction Layer del DMM. 43
Figura 30 – Implementation Layer del DMM. 44
Figura 31 – Le classi Statistics e Visualization. 45
Figura 32 – Il pacchetto DmmParams 46
Figura 33 – Classe ImageTable 49
Figura 34 – Classe NextControl 50
Figure 35 – Le classi Unsupervised e Clustering 57

Figura 36 – Diagramma UML del livello di implementazione del DMM integrato con i modelli CSOM e GSOM 59
Figura 37 – La classe GSOM 60
Figura 38 – Esempio:Dataset Originario 62
Figura 39 –Esempio: Dataset estratto S1 63
Figura 40 – Esempio:Secondo dataset estratto S2 63
Figura 41 – Esempio: Dataset di overlap inserito nel file “overlap” 63
Figura 42 – Esempio: File “overlap” dopo il train su S1 64
Figura 43 – Esempio: File “overlap” dopo il train su S2 64
Figura 44 – Esempio:tabella temporanea riguardante il dataset S1 64
Figura 45 – Esempio: tabella temporanea riguardante il dataset S2 65
Figura 46 – Esempio: il file “stat.log” al termine del caso d’uso test train 66
Figura 47 – La classe CSOM 66
Figura 48 – Interfaccia grafica del DMPluginWizard 68
Figura 49 – Esempio di riempimento dei campi della GUI del DMPlugin 69
Figura 50 – Finestra della voce “Fields” 70
Figura 51 – Finestra della voce “Input Files” 70
Figura 52 – Esempio di generazione codice del plugin 71

Elenco delle Tabelle

Tab. 1 – Parametri casi d’uso: train o run, un livello di rete 51
Tab. 2 – Paramatri caso d’uso “test”, un solo livello di rete 52
Tab. 3 - Parametri casi d’uso “test” ,due livello di rete 53
Tab. 4 - Parametri casi d’uso:train o run, tre livelli di rete 54
Tab. 5 - Parametri casi d’uso “test”, tre livelli di rete 55

Introduzione

Il presente lavoro riguarda la progettazione e implementazione di una serie di strumenti software in grado di estendere le funzionalità del progetto DAME (Data Mining & Exploration). Tale progetto nasce con lo scopo di ideare e sviluppare una suite di sistemi software, "*web-oriented*", server-side, basati sul calcolo distribuito (CLOUD, GRID) e sui paradigmi del *Machine Learning* e del *Soft Computing* per il data mining su grandi quantità di dati (*Massive Data Set*, MDS), prevalentemente di tipo astronomico.

Il problema affrontato riguarda l'estensione della suite DAME (in particolare del suo componente Data Mining Model, DMM), inserendovi ex novo quanto segue:

- una nuova categoria di algoritmi non supervisionati: **Unsupervised**;
- una nuova funzionalità: **Clustering**;
- un'infrastruttura per ospitare reti multi-livello per il clustering: **MultiLayerClustering**;
- un nuovo modello di rete non supervisionato general-purpose: **Self Organizing Map** (SOM);
- una versione specializzata del modello SOM per il clustering di immagini astronomiche, attraverso il reverse engineering e l'adattamento di librerie software, di proprietà del progetto DAME, che fanno riferimento al modello NExTII, un sistema di gestione di workflow per la segmentazione di immagini astronomiche multi-banda.

Il presente elaborato è organizzato come segue.

Le motivazioni sulla nascita del progetto DAME, le caratteristiche, l'architettura e i modelli di reti neurali supervisionati già implementati sono descritti nel Capitolo 1.

Nel Capitolo 2 si descrive l'operazione di reverse engineering effettuata sia sulla libreria NExTII, sia sul componente DMM della suite DAME, al fine di ottenere tutti gli elementi e le informazioni utili per lo sviluppo della soluzione al problema che mi è stato proposto.

Nel capitolo 3, cuore di questo elaborato, si fornisce una descrizione dettagliata del lavoro svolto partendo dall'estensione della libreria NExTII e del componente DMM, fino alla creazione dei plugin che permettono di eseguire esperimenti di data mining e clustering tramite il modello di rete non supervisionato SOM.

Il lavoro è stato svolto in partnership con:

- Università degli Studi di Napoli Federico II, Dipartimento di Fisica;
- Istituto Nazionale di Astrofisica (INAF), sede regionale Osservatorio Astronomico di Capodimonte;
- California Institute of Technology, Pasadena-USA;

e in collaborazione con:

- Virtual Observatory Technological Infrastructure (VOTECH);
- S.Co.P.E;
- Ministero dell'Istruzione dell'università e della Ricerca (MIUR);
- European Virtual Observatory (EURO-VO).

Capitolo 1. Il progetto DAME (Data Mining & Exploration)

In questo capitolo sarà descritto il progetto DAME (Data Mining & Exploration) e, in particolare, sarà data una panoramica generale sulla struttura della suite, sulle funzionalità che essa offre agli utenti e sull'architettura su cui si basa l'interazione fra i vari componenti della suite stessa.

1.1 Descrizione generale

Il progetto DAME nasce dall'esigenza di offrire alla comunità scientifica (astrofisica in prima istanza) uno strumento di "*data mining*" (esplorazione dei dati)[1] "*web-based*" e "*service-oriented*", utile per la ricerca e per l'estrazione della conoscenza, non nota a priori, da grandi quantità di dati (*Massive Data Sets*, **MDS**), implementato su piattaforme di calcolo distribuito, sia "*resource-based*" (GRID) sia "*service-based*" (CLOUD), rispettando gli standard internazionali istituiti dall'**International Virtual Observatory Alliance** (IVOA), relativi alle modalità di rappresentazione ed estrazione dei dati in ambito astronomico ed astrofisico. Questa esigenza è legata all'impellente necessità di gestire enormi quantità di dati prodotti da strumenti di nuova generazione altamente tecnologici come i "**telescopi di survey**"[2], dotati di un grande campo di vista e, sul piano focale, di un rivelatore ad alta risoluzione in grado di osservare ampie regioni di cielo con estrema nitidezza in più bande fotometriche; ciò vuol dire che la stessa regione di cielo è osservata più volte attraverso diversi filtri, con diverse caratteristiche, portando alla produzione di oltre 10TB di immagini astronomiche di grandi dimensioni (fino a 32K x 32K pixel) per notte che devono essere acquisite, elaborate e catalogate.

[1] Per Data Mining si intende il processo di selezione, esplorazione e modellazione di grandi quantità di dati, al fine di scoprire regolarità o relazioni,
[2] Un esempio di telescopio di survey è il "**VLT Survey Telescope**" (VST), progettato e realizzato dall'Osservatorio Astronomico di Capodimonte e in fase di installazione presso l'Osservatorio di Cerro Panaral dell' European Southern Observatory (ESO) in Cile (**www.eso.org**).

1.2 Architettura della suite

L'architettura generale della suite DAME (Figura 1) si basa sulla suddivisione del sistema in diversi componenti software, ciascuno preposto ad un particolare compito o espletamento di servizi e comunicanti tra loro attraverso lo scambio di documenti in "*eXtendible Markup Language*" (**XML**).

In particolare la suite è suddivisa in 5 componenti principali:

- **Front-End** (FE)
- **FrameWork** (FW)
- **DRiver** (DR)
- **Registry & DataBase** (REDB)
- **Data Mining Models** (DMM)

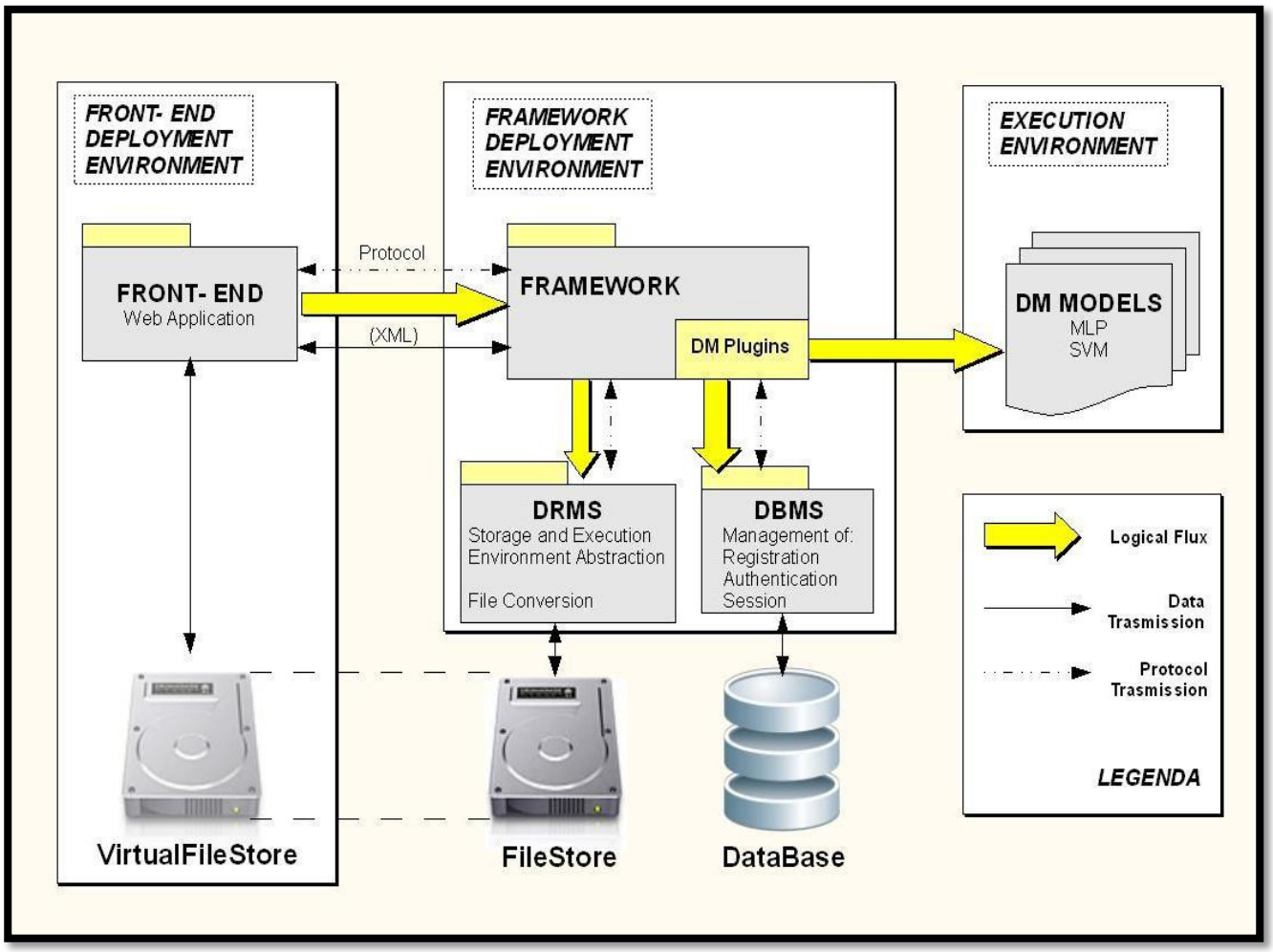

Figura 1 – Architettura generale della suite DAME.

1.2.1 Front-End

Il **Front-End** è l'elemento che si occupa direttamente delle interazioni tra la suite e l'utente finale tramite un'interfaccia grafica (o *Graphical User Interface*, **GUI**).

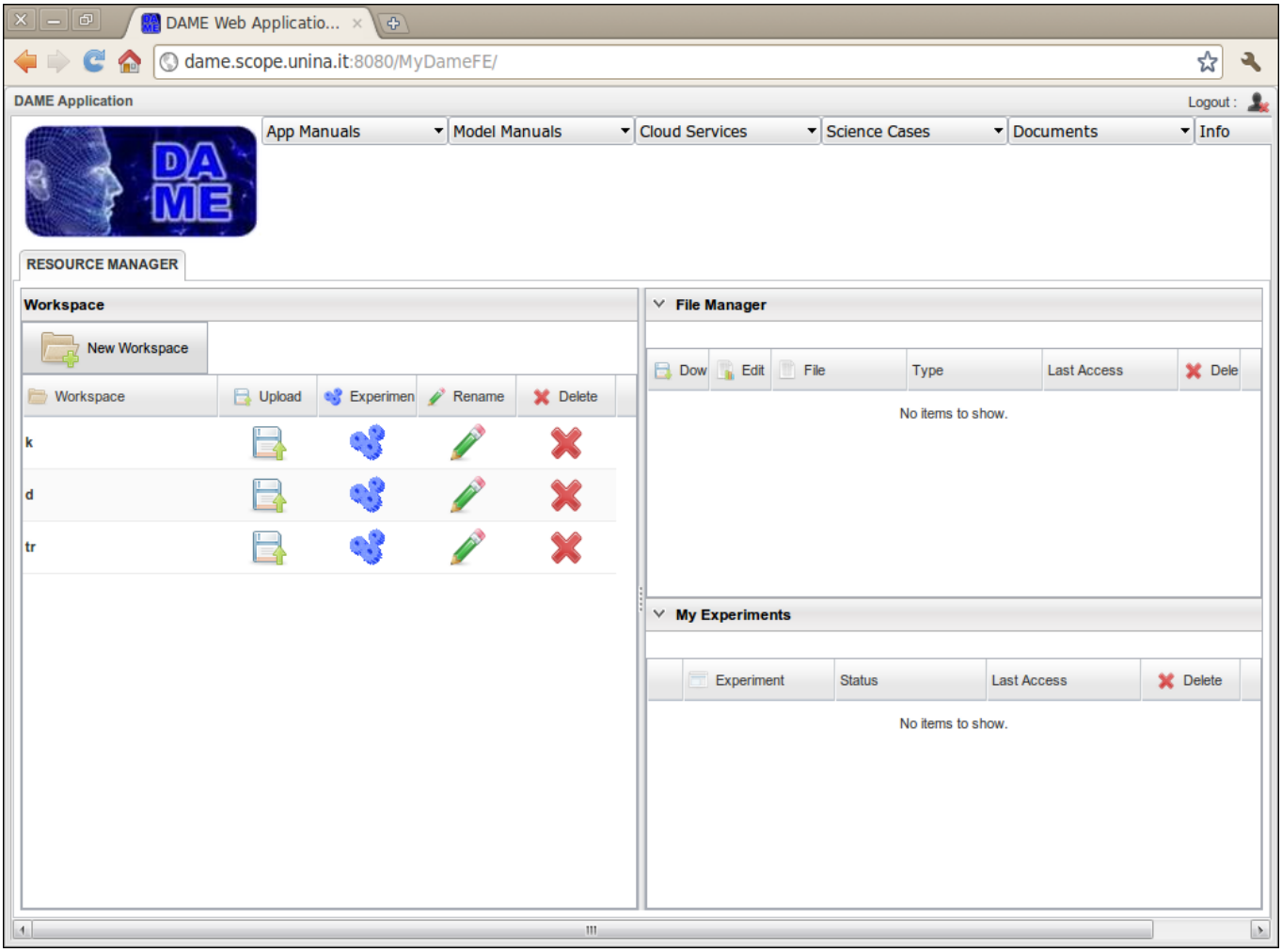

Figura 2 – Interfaccia Utente della Suite DAME.

L'utente, una volta registrato e 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, download e upload di file, configurazione dei propri dataset, monitoraggio e download dei risultati, sia grafici sia 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 la descrizione delle stesse.

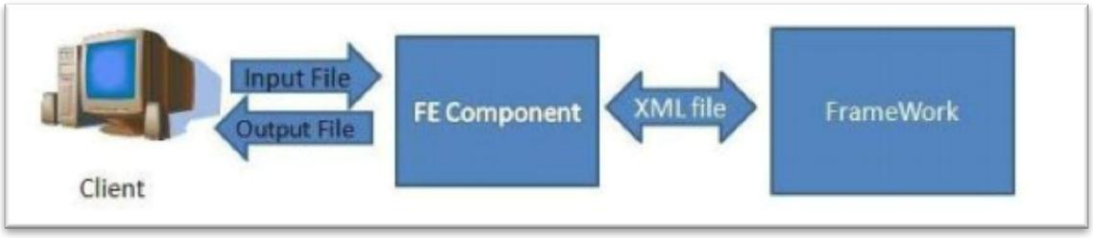

Figura 3 – Esempio di comunicazione tra FE e FW.

1.2.2 FrameWork

Il **FrameWork** è il cuore dell'applicazione ed è stato creato per funzionare su diverse piattaforme: Cloud, Grid e Stand-Alone, quest'ultimo indicante una piattaforma *single-device*. 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'hardware sottostante e lo strato di software della suite DAME.

Il FrameWork è principalmente basato su:

- **RESTful, stateless Web-service: REST** è l'acronimo di *Representational Transfer State*, ed è un paradigma per la realizzazione di applicazioni Web che permette la manipolazione delle risorse tramite i metodi del protocollo HTTP; i servizi web sono identificati come risorse utilizzabili dall'utente finale tramite un "*Uniform Resource Locator*" (**URL**) che identifica ciascuna risorsa; un'architettura client/server basata su REST non ha meccanismi per la persistenza dello stato di una risorsa (*stateless*), ossia ogni richiesta è indipendente l'una dall'altra e, se si vuole che i dati persistano tra richieste successive, l'utente finale dovrà inviare a ogni richiesta i dati.
- **Interfaccia di amministrazione**: permette all'amministratore di installare e disinstallare funzionalità e di eseguire analisi statistiche sulla suite;
- **XML Generator**: utilizzato per generare file XML per interagire con il Front-End;

- **Data Types**: oggetti che rappresentano entità fondamentali della suite (ad esempio utente, esperimento, ecc);
- **Security**: classi utilizzate per la sicurezza della suite;
- **Error Management**: gestore degli errori che possono verificarsi durante l'utilizzo della suite;
- **Data Mining Plugin (DMPlugin):** componente dotato di interfaccia utente utilizzabile da sviluppatori interni ed esterni per aggiungere all'interno della suite nuove funzionalità riguardanti tecniche di data mining. Tali funzionalità non sono altro che nuovi modelli oppure varianti dei modelli già presenti nella suite DAME.

1.2.3 Driver

Il **Driver** ha il compito di astrarre, come già accennato, lo strato di software del FrameWork dalla piattaforma su cui è eseguito. Esso è stato realizzato per funzionare sia su piattaforme **Grid** (infrastruttura di calcolo distribuita) che su piattaforma **Stand-Alone** (singola macchina) (Figura 4).

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.

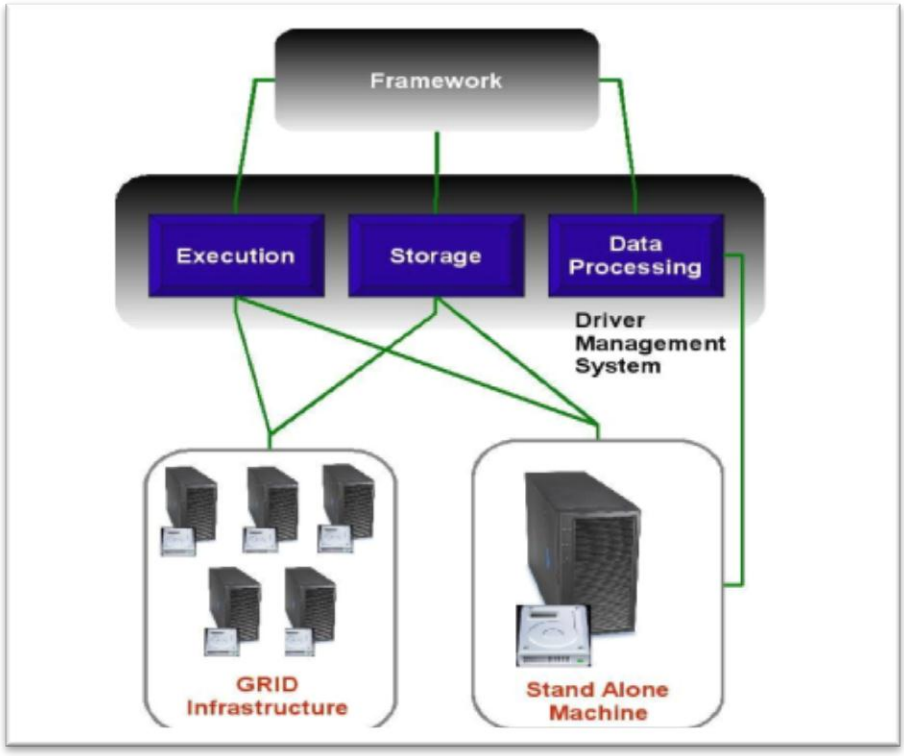

Figura 4 – Funzioni del Driver.

Per evitare la ridondanza dei dati, la suite DAME è dotata di un *FileStore* (**FS**) per la memorizzazione dei file fisici, gestito direttamente dal FrameWork tramite il Driver. Questo componente prevede inoltre una libreria di codice per la conversione di vari tipi di dati, permettendo così all'utente di utilizzare diversi tipi di file per i propri esperimenti.

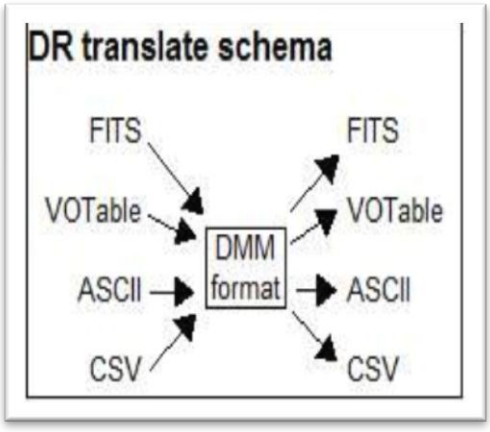

Figura 5 – Schema di traduzione del Driver.

I tipi di file attualmente presenti e utilizzabili all'interno della suite sono i seguenti:

- **Flexible Image Transport System (FITS)**: è il formato di file più utilizzato in ambito astronomico e progettato per dati di tipo scientifico; la struttura dei file di tipo Fits è molto particolare e comprende uno o più HDU (*Header* + *Data Unit*), metadati riguardanti l'immagine, come il tipo di estensione o la calibrazione fotometrica, e altri dettagli tecnici che riguardano soprattutto gli strumenti utilizzati per rilevare l'immagine.

  In basso possiamo osservare un'immagine della cometa Hale-Bopp in formato FITS e il relativo header.

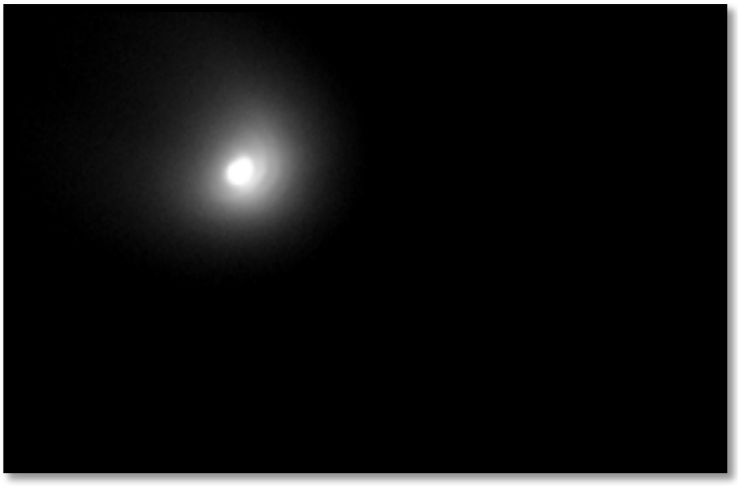

Figura 6 – Chioma della cometa Hale-Bopp.

```
=== FITS header ===
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                   16 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                  378 / length of data axis 1
NAXIS2  =                  242 / length of data axis 2
EXTEND  =                    T / FITS dataset may contain extensions
COMMENT   FITS (Flexible Image Transport System) format defined in Astronomy and
COMMENT   Astrophysics Supplement Series v44/p363, v44/p371, v73/p359, v73/p365.
COMMENT   Contact the NASA Science Office of Standards and Technology for the
COMMENT   FITS Definition document #100 and other FITS information.
BZERO   =                32768 / offset data range to that of unsigned short
BSCALE  =                    1 / default scaling factor
COMMENT =                      /  Astrophysics Supplement Series v44/p363, v44/p
COMMENT =                      /  Contact the NASA Science Office of Standards a
COMMENT =                      /  FITS Definition document #100 and other FITS i
BZERO   =                32768 /offset data range to that of unsigned short
BSCALE  =                    1 /default scaling factor
OBJECT  = Hale-Bopp          /
DATE-OBS= '18/4/1997'        /
UT      = '19:18:12,22'      /End of exposure
TELESCOP= 'ap.=200 focal=2000' /
EXPOSURE=                1.000 /Exposure in seconds
TEMPCASE=                 11.0 /Case Temperature in degree
TEMPCCD =                  9.0 /CCD Temperature in degree
COMMENT =                      /= 'Image created by IPX 00.02' /
INSTRUME= 'SETI_245'         /_
ORIGIN  = 'Cassinelle_Observatory' /_
OBSERVER= 'Guido_Conte'      /_
DATAMIN = 0.000              /Low pixel value
DATAMAX =             5820.000 /High pixel value
DATAAVE =               54.800 /Average pixel value
COMMENT =                      /= From 11.img IPX v. 00.20 Beta 6 seti245 /
END
```

Figura 7 – Header dell'immagine FITS della cometa Hale-Bopp in Figura 6.

- **VOTable**: formato dei file utilizzato all'interno della suite DAME. Rappresenta uno schema XML standard per lo scambio di dati rappresentati sotto forma di tabelle. Tali tabelle sono formate da un insieme non ordinato di righe, ognuna delle quali è una sequenza di celle che contiene un dato primitivo oppure un array.

  In figura si può osservare lo schema XML del formato VOTable;

```
<?xml version="1.0"?>
<VOTABLE version="1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.ivoa.net/xml/VOTable/VOTable/v1.1">
   <RESOURCE name="resourceName">
        <TABLE name="Parameters">
           <DESCRIPTION>Description (optional tag)</DESCRIPTION>
           <PARAM name="nomePar1" datatype="tipoPar1" value="valore1"/>
           <PARAM name="nomePar2" datatype="tipoPar2" value="valore2"/>
        </TABLE>
        <TABLE name=""InputFile">
           <DATA>
              <TABLEDATA>
                <STREAM encoding="gzip" href="ftp://server.com/mydata.tar.gz"/>
              </TABLEDATA>
           </DATA>
        </TABLE>
   </RESOURCE>
</VOTABLE>
```

Figura 8 – Schema generale di un file VOTable.

- **American Standard Code for Information Interchange (ASCII):** sistema di codifica dei caratteri a 7 bit, comunemente utilizzato nei calcolatori.
- **Comma-Separated Values (CSV):** formato di file basato su testo comunemente utilizzato per importare o esportare tabelle di dati (come fogli elettronici o database); in questo formato ogni riga della tabella (o del record del database) è rappresentato da una singola linea di testo, suddivisa in campi (colonne) separati da un apposito carattere divisore.

1.2.4 Registry & DataBase

Tutte le informazioni riguardanti gli utenti, le loro sessioni di lavoro, i vari dati di input/output di ogni esperimento e dati temporanei e finali di processi lanciati dall'utente ancora in fase di esecuzione, sono gestite dal componente **Registry & DataBase (REDB)**. Esso è primariamente composto di un *DataBase Management System* (**DBMS**) relazionale (Figura 9), basato su tecnologie MySQL DBMS Server, un connettore "*Java DataBase Connectivity* " (**JDBC)** e una " *Application Program Interface*" (**API**) di accesso ai dati.

Tali tecnologie permettono di

- Gestire le informazioni sugli account degli utenti;
- Memorizzare e gestire tre tipi di files:
    - *Supported*: tipi di files utilizzabili dall'utente (dataset);
    - *Exotic*: files di configurazione dei modelli della suite;
    - *Custom*: files e dati intermedi.

Il componente REDB comunica continuamente con il FrameWork per garantire l'integrità e la consistenza dei dati durante l'interazione con l'utente.

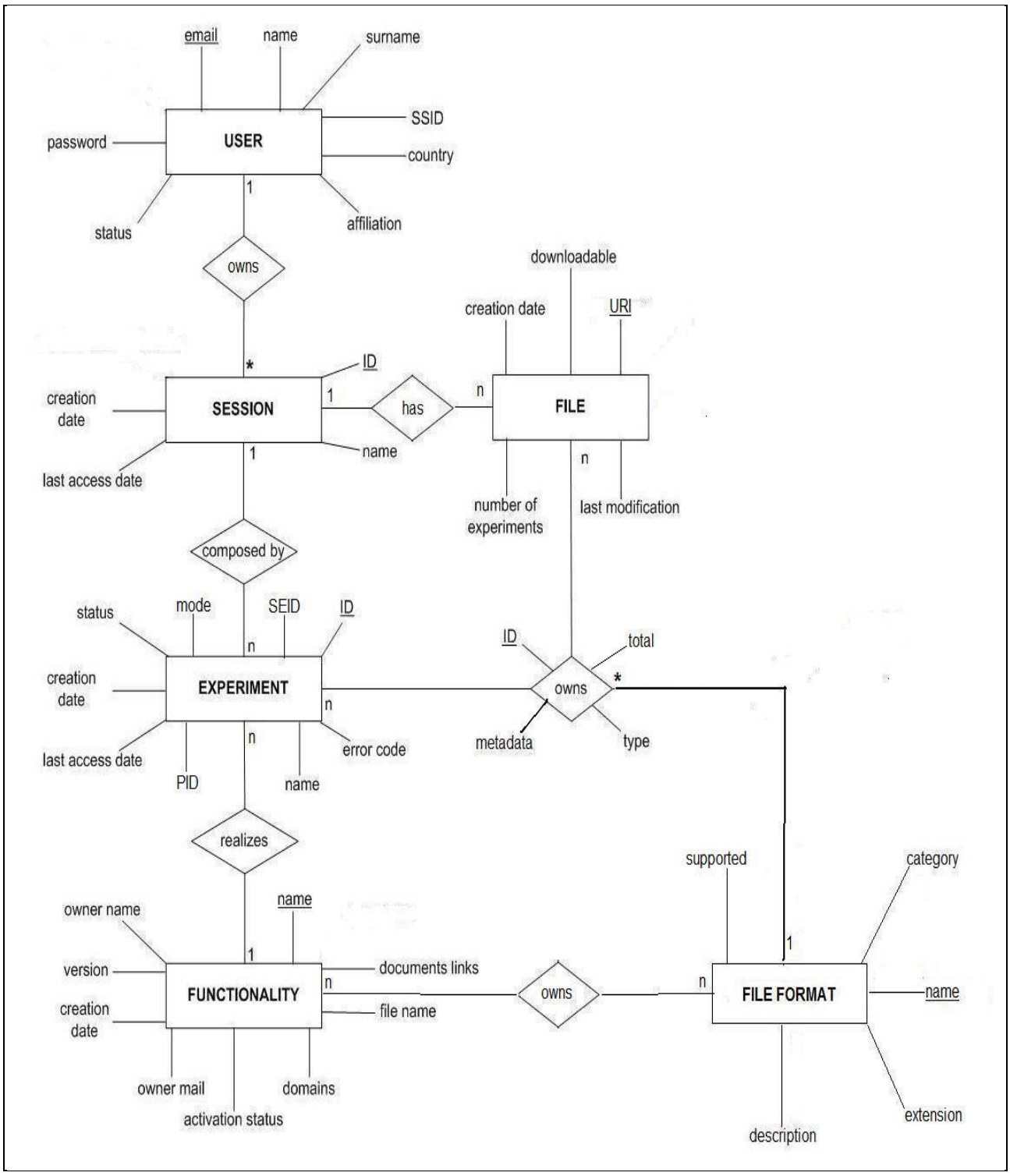

Figura 9 – Modello ER del REDB.

1.2.5 Data Mining Model

Infine, il componente che racchiude i modelli di data mining è il **Data Mining Model** (**DMM)**, in particolare oggetto di modifica e integrazione per il presente lavoro (paragrafo 2.2).

Tali modelli sono algoritmi basati su tecniche di *Artificial Intelligence* (**AI**), come le reti neurali, implementati sottoforma di API gestite tramite interfaccia JAVA.

Ogni modello implementato nel DMM offre all'utente la possibilità di utilizzare una serie di casi d'uso[3]:

- **Funzione "train":** permette l'addestramento della rete neurale, tramite un set di dati (*training set*), rendendola utilizzabile nella risoluzione di un problema concreto; il training set contiene i risultati osservati per ogni combinazione dei parametri di input;
- **Funzione "run":** permette l'utilizzo vero e proprio della rete neurale scelta per eseguire l'esperimento; si forniscono alla rete delle combinazioni dei parametri di input di cui è ignoto il risultato; l'output, nel caso di reti neurali supervisionate, sarà calcolato in base ai dati forniti nella fase di addestramento della rete;
- **Funzione "test":** permette di valutare il risultato ottenuto nella fase di addestramento di una rete neurale supervisionata; alla rete neurale è fornito in input un insieme di dati, diverso dal training set, ma di cui si conosce a priori il risultato per valutare l'accuratezza del training della rete;
- **Funzione "full"**: permette di utilizzare in sequenza le tre fasi in precedenza descritte nell'ordine train, test e run.

[3] In questo contesto, un "**caso d'uso**" rappresenta una delle modalità utilizzate per eseguire i vari modelli della suite

1.3 Funzionalità della suite

La suite DAME offre varie funzionalità utilizzabili dall'utente per eseguire un esperimento ed esse sono rappresentate tramite i **Data Mining Plugin (DMPlugin)**. I DMPlugin sono blocchi di codice implementati come librerie condivise caricate dinamicamente a run-time dal programma principale, che durante il loro ciclo di vita comunicano con uno dei vari modelli del componente DMM.

Considerando che, nella prima release alpha della suite l'unica categoria di algoritmi di "**apprendimento**" offerta è quella di tipo "**supervisionata**" o "*supervised*"; tale categoria permette l'utilizzo di due tipi di funzionalità:

- **Classificazione;**
- **Regressione;**

Esse sono specializzate dall'utente finale tramite la specifica dei parametri di input. Se l'utente non imposta alcun valore specifico per i parametri, l'applicazione assegna automaticamente dei valori di default, diversi a seconda del modello scelto per eseguire l'esperimento.

1.3.1 Classificazione

La funzionalità **classificazione** viene utilizzata per associare agli elementi appartenenti ad uno spazio di parametri X gli elementi, detti "**etichette**" di un insieme Y. Formalmente, il problema può essere esposto come segue:

"*Dato un insieme di dati iniziale* $\{ (x_1,y_1),\ldots,(x_n,y_n) \}$ *(dove* x_i *sono dei vettori), la funzione di classificazione, o classificatore,* ***h: X->Y*** *associa a ogni elemento* $x \in X$ *un'etichetta* $y \in Y$*."*

I problemi di classificazione possono essere suddivisi in due categorie:

- **Crispy Classification:** dato un vettore di input x, detto anche "*pattern*", il classificatore ritorna come valore di output un'etichetta y (scalare);

- **Probabilistic Classification:** dato un pattern x, il classificatore ritorna in output un vettore y, in cui ogni elemento rappresenta la probabilità che y_i sia l'etichetta "*giusta"* associabile al vettore x.

L'operazione di classificazione avviene in tre passi:

- **Training** o addestramento: il classificatore è *"addestrato"* inserendo coppie input-output (patterns-etichette) note a priori;
- **Testing**: in input si inserisce un insieme di dati per il test ottenendo in output una statistica relativa al grado di appartenenza alle diverse classi;
- **Evaluation:** in input si inserisce un insieme di dati non ancora etichettato, cioè coppie input-output non utilizzate nel training, e per cui in output si ha l'associazione (o *"etichettatura*") in base alle classi predefinite.

1.3.2 Regressione

La funzionalità **regressione** è utilizzata per la ricerca di un'associazione tra elementi appartenenti a un dominio $\mathbf{R}^{n}$ ed elementi appartenenti a un dominio $\mathbf{R}^{m}$, con n>m. Si possono distinguere due tipi diversi di regressione:

- **Data table statical correlation (Function Approximation):** cerca l'associazione senza alcuna assunzione a priori sul tipo di distribuzione dei dati; l'algoritmo prende in input solo coppie di vettori (x,y) cercando la funzione che approssima nel miglior modo la distribuzione dei dati derivata dalle coppie di vettori;
- **Function Fitting (Curve Fitting):** tenta di validare l'ipotesi che la distribuzione dei dati segua una determinata funzione; l'algoritmo prende in input coppie di vettori (x,y) ed la funzione che si vuole associare, e tenta di trovare in ogni parametro il valore che si adatta maggiormente alla funzione data in input.

L'operazione di regressione avviene in tre passi, come per la classificazione:

- **Training:** l'addestramento avviene inserendo coppie input-output note a priori;
- **Testing:** in input si inserisce un insieme di dati per il test ottenendo in output una statistica riguardante gli errori e i valori di output ottenuti;
- **Evaluation:** in input si inseriscono coppie input-output non utilizzate nel training, e per cui in output si hanno i valori ottenuti.

1.4 Modelli della suite DAME

Per ciascuna delle precedenti funzionalità, classificazione e regressione, l'utente finale può scegliere di utilizzare uno dei modelli (algoritmi) presenti nella suite. Allo stato attuale, gli algoritmi della categoria Supervised presenti nella suite sono:

- **Support Vector Machine (SVM):** rappresenta un insieme di metodi di apprendimento supervisionato ed è nato per risolvere problemi di classificazione, successivamente esteso da Vapnik per problemi di regressione. Le SVM possono essere utilizzate per separare classi linearmente separabili tramite un classificatore lineare, ma anche per classi non linearmente separabili in uno spazio fortemente multidimensionale, detto "*feature space*". Lo spazio di input viene rimappato nel feature space in cui viene identificato un classificatore lineare e riportato nello spazio iniziale (Figura 10).

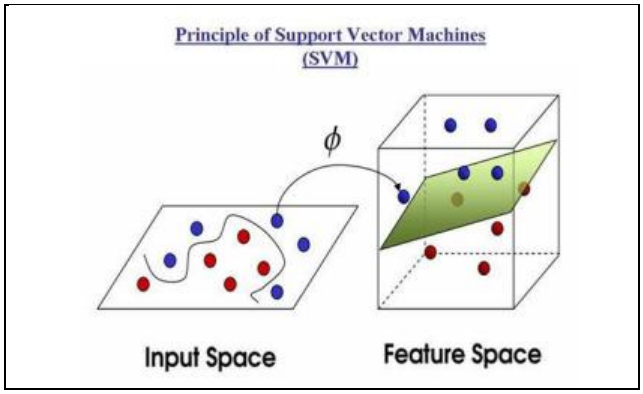

Figura 10 – Prinicipio del modello SVM

Il modello SVM implementato nella suite DAME si basa sulla libreria "***libsvm***"[ref. (1)] che offre quattro tipi di funzioni dette "*kernel*" (**funzione Φ**):

- **Lineare;**
- **Polinomiale;**
- **Radial Based Function (RBF);**
- **Sigmoid;**

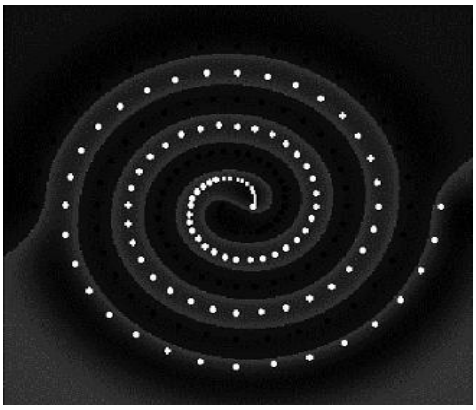

Figura 11 – Esempio di immagine per l'applicazione del modello SVM.

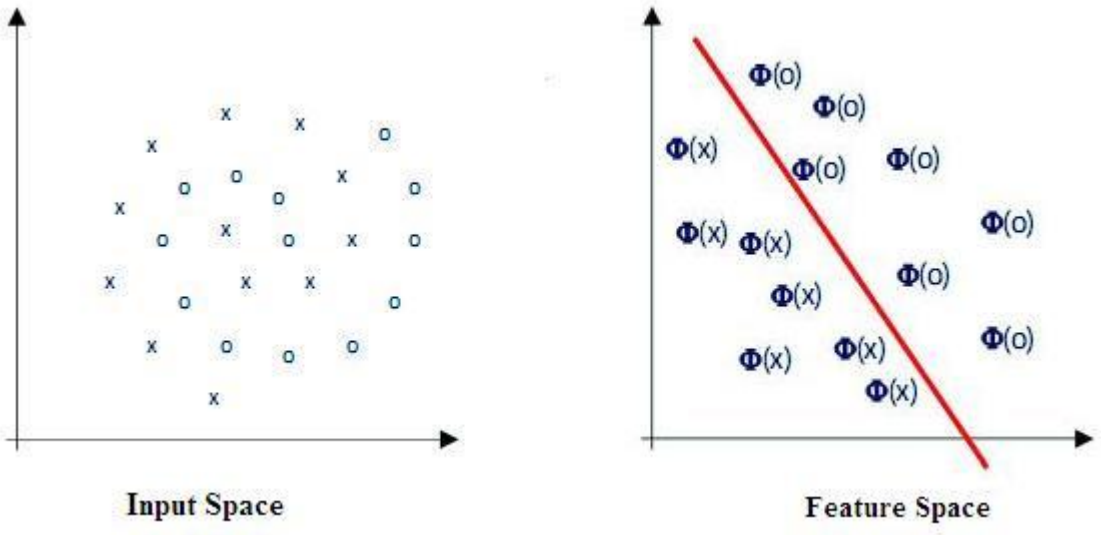

Figura 12 – Risultati dell'applicazione del modello SVM

In figura si può vedere un esempio dell'applicazione di una delle funzioni kernel rese disponibili dalla libSVM. Esso mostra l'importanza di avere a disposizione un insieme di funzioni in modo

tale da poter scegliere la configurazione più efficace e adatta a risolvere il problema dato e ad evitare il cosiddetto "*over-fitting*", cioè un eccessivo adattamento della funzione alla distribuzione dei dati in modo tale da rendere ambiguo l'utilizzo della funzione stessa.

- **Multi-Layer Perceptron (MLP):** è una rete neurale artificiale feed-forward di tipo supervisionato che necessita di una fase di addestramento. Tale modello nasce dalla necessità di dover classificare datasets non linearmente separabili e può essere applicato in vari campi: *pattern recognition*, *process modelling*, *prediction*, ecc…;

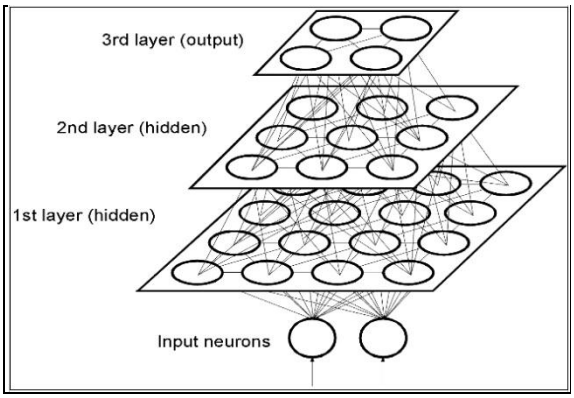

Figura 13 – Struttura di una rete neurale MLP

Come si può osservare nella Figura 13, una rete neurale MLP è formata da vari strati: uno strato di input, uno o più strati interni (*hidden layers*) e uno strato di output e i neuroni appartenenti a strati adiacenti sono fortemente connessi tra di loro.

- **Multi-Layer Perceptron with Genetic Algorithm (MLPGA):** è un modello "*ibrido*" tra il Multi-Layer Perceptron e gli Algoritmi Genetici, ottenuto tramite la tecnica del "*soft-computing*" utilizzata per la realizzazione di modelli ibridi che ereditano le caratteristiche insite nei vari modelli autoadattivi. Come per il modello MLP, l'MLPGA necessita di una tecnica di addestramento che consiste nel "*popolare*" in modi differenti le reti neurali così da poter scegliere la "*popolazione*" che si adatta maggiormente alle coppie input-output utilizzate.

Capitolo 2.Tecnologie di base

In questo capitolo è esposta una breve trattazione delle librerie utilizzate nella realizzazione della soluzione al problema proposto. Il primo paragrafo riguarda la libreria NExTII, che mette a disposizione operazioni utili per la segmentazione automatica e semiautomatica delle immagini astronomiche; nel secondo e nel terzo paragrafo, invece, sono descritti due componenti della suite DAME: il DMM e il DMPLugin. In particolare, il DMM rappresenta il componente principale su cui sono state apportate delle modifiche, rappresentate dall'integrazione delle reti di tipo non supervisionato.

2.1 La libreria NExTII (Neural ExTractor)

La libreria NExTII (Neural ExTractor II) [Rif. (2)] fa ampio uso di tecniche di intelligenza artificiale, in particolare di reti neurali non supervisionate, per la segmentazione di immagini astronomiche di tipo FITS.
In generale, un'immagine astronomica può essere rappresentata come una matrice bi-dimensionale di elementi detti "*pixel*" (contrazione dei termini inglesi *Picture Elements*, elementi d'immagine) che, rappresentano la luminosità della stessa in determinati punti. Essa è caratterizzata da vari parametri:

- Dimensione del pixel, che varia tra i 15 e i 55 micron;
- Numero di pixel;
- Intervallo dinamico dal valore minimo e massimo del segnale registrabile su ogni singolo pixel e dipende dall'elettronica di acquisizione utilizzata.

La segmentazione consiste nella ripartizione delle immagini in regioni connesse e disgiunte, in cui ogni regione è omogenea, e tale ripartizione avviene tramite il "*clustering",* ovvero il raggruppamento dei pixel dell'immagine in due classi: segnale e fondo. Il risultato che si ottiene è una maschera binaria in cui i pixel accesi formano gli oggetti astronomici da catalogare, mentre i pixel spenti rappresentano il fondo, cioè le regioni di cielo che non contengono oggetti.
La segmentazione avviene in due fasi:

- clustering dei pixel dell'immagine data in input e costruzione della maschera binaria;
- classificazione dei pixel dell'immagine in base alla luminosità media del cluster di appartenenza.

Nel presente lavoro è considerata esclusivamente la fase che riguarda il clustering delle immagini tramite l'utilizzo di reti neurali SOM applicate singolarmente oppure inserite in una struttura multi-livello.

2.1.1 Reverse Engineering

Per "*Reverse Engineering*" si intende il processo di analisi mirato all'identificazione delle componenti, e delle relazioni tra componenti, di un sistema software già esistente ed ha come obiettivo la comprensione del sistema in analisi. Solitamente, questa tecnica è utilizzata dai progettisti per aumentare il grado di conoscenza di un sistema software quando quest'ultimo deve essere sottoposto ad operazioni di modifica, manutenzione o integrazione ed evoluzione. Infatti, non è raro che, nei casi reali, i progettisti debbano eseguire operazioni di integrazione o manutenzione del codice senza avere una conoscenza pregressa e approfondita dello stesso.

Le tecniche e le applicazioni a disposizione per il Reverse Engineering forniscono i mezzi necessari per generare, a partire dal codice, un'adeguata documentazione; in particolare, sono in grado di produrre nuove rappresentazioni, spesso grafiche, o recuperare quelle perse oppure incoerenti con ciò che in realtà è stato implementato.

Il lavoro che riguarda il Reverse Engineering consiste di diverse fasi:

- **comprensione del codice**: analisi approfondita del software o di parte di esso, poiché se si vuole apportare una modifica occorre capire come funziona e dove apportare le modifiche.
- **analisi di impatto**: capire quali funzionalità devono essere aggiornate in modo da renderle consistenti con la modifica da apportare;
- **modifica del codice**: fase di codifica dei cambiamenti o estensioni da implementare;
- **validazione**: verificare che le modifiche apportate non introducano errori.

La fase più critica è sicuramente quella della comprensione del codice, fase in cui un programmatore cerca di comprendere la struttura interna del software, o parte di essa, e il suo funzionamento prima di apportare eventuali modifiche. In questa fase le tecniche di Reverse

Engineering sono le più utili poiché permettono di recuperare informazioni, anche mai esistite, riguardanti il sistema software su cui si va ad operare.

I risultati dell'applicazione delle tecniche per l'interpretazione del codice della libreria NExTII e della suite DAME è descritta nei paragrafi successivi. In particolare, il lavoro si è concentrato esclusivamente sulle classi della libreria utili per l'integrazione del modello di rete neurale SOM e del clustering multi-livello nella suite DAME.

2.1.2 La struttura di NExTII

La libreria NExTII, implementata in linguaggio C++ e "*object-oriented*", è strutturata secondo una gerarchia di classi come mostrato nel diagramma seguente[4].

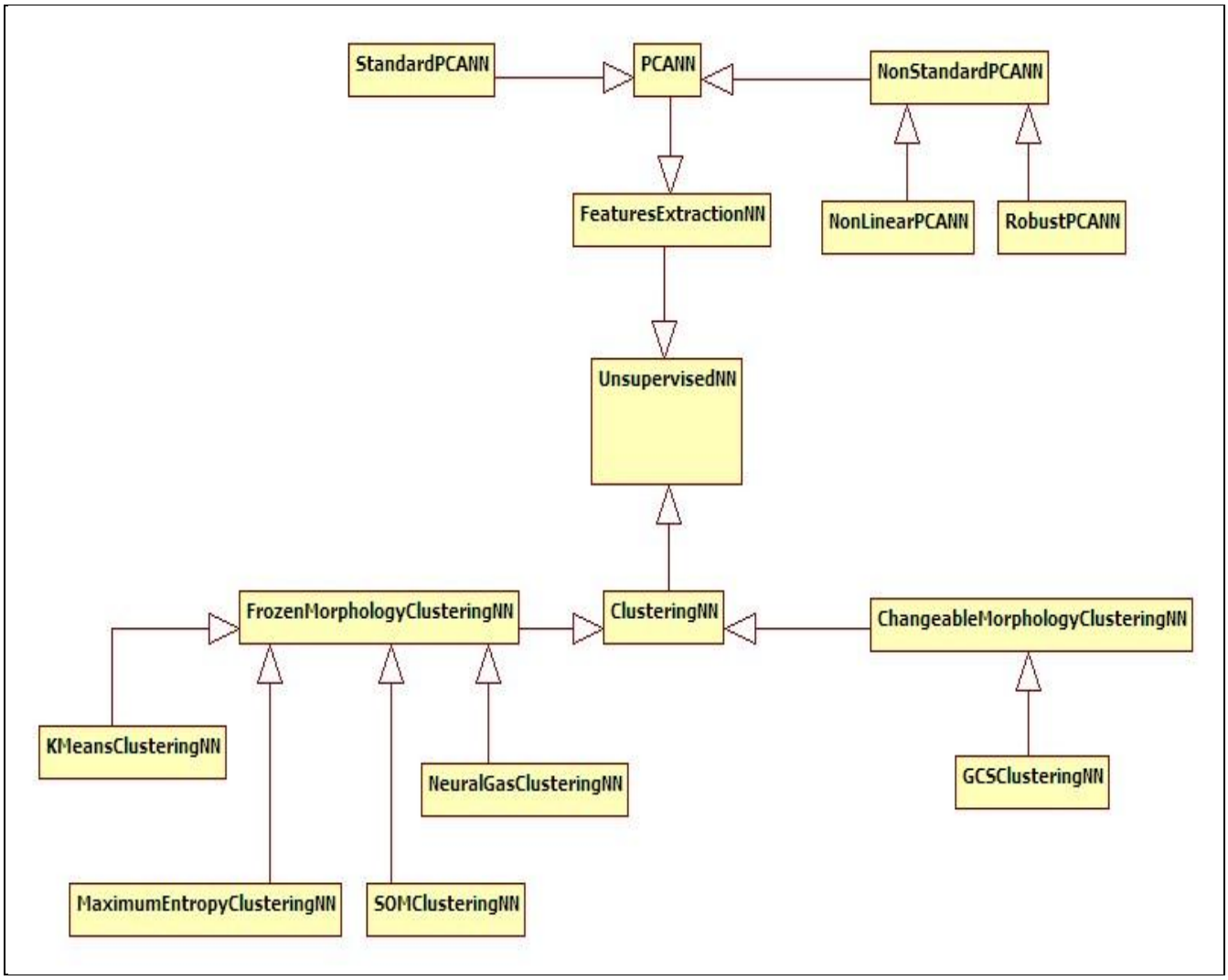

Figura 14 – La gerarchia delle UnsupervisedNN.

[4] Nel diagramma sono stati omessi metodi e attributi per motivi di spazio. Nel paragrafo 4.1.2 è presente una descrizione più dettagliata solo delle classi utilizzate nel lavoro da me svolto.

La classe UnsupervisedNN comprende alcune reti neurali di tipo non supervisionato. Essa si suddivide in:

- **ClusteringNN:** comprende alcuni modelli di rete utilizzati in NExTII per il partizionamento di pattern; esse si dividono in:
    - **ChangeableMorphologyClusteringNN:** reti neurali non supervisionate a morfologia variabile, dove per morfologia si intende lo schema di interconnessione della rete;
    - **FrozenMorphologyClusteringNN**: reti neurali non supervisionate a morfologia fissa, dove per morfologia si intende lo schema di interconnessione della rete;
- **FeaturesExtractionNN:** comprende alcuni modelli di rete utilizzati in NExTII per la riduzione della dimensionalità di un vettore di input;

La libreria presenta inoltre una classe, la **MultiLayerClusteringNN**, utilizzata per il clustering multi-livello, che per ogni livello, prevede l'utilizzo di un qualsiasi modello di rete non supervisionato presente nella libreria stessa.

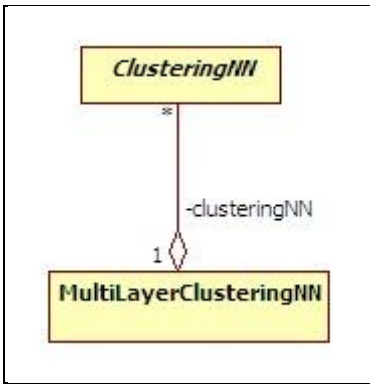

Figura 15 – Relazione tra ClusteringNN e MultiLayerClusteringNN.

Per la rappresentazione dei dati di input e per la costruzione dei dataset da utilizzare, NExTII utilizza le classi Image e Pixel.

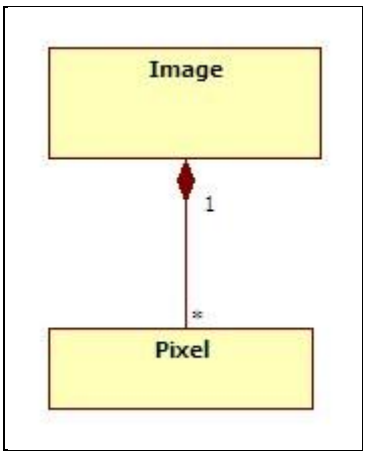

Figura 16 – Relazione tra Image e Pixel

Nei paragrafi successivi sono descritte in dettaglio le classi utilizzate per il lavoro da me svolto all'interno del progetto DAME.

2.1.2.1 Le classi Image e Pixel

La libreria NExTII si occupa, come già accennato, della segmentazione di immagini astronomiche. La prima fase di tale operazione è rappresentata dal clustering dei pixel, ovvero il raggruppamento dei pixel dell'immagine in sottoinsiemi (*cluster*) e per tale attività si utilizzano reti neurali con apprendimento non supervisionato.

Un *pixel* è l'unità elementare dell'informazione contenuta in un'immagine e corrisponde a un punto luminoso che è, a sua volta, il risultato della combinazione di una o più bande di colore; in altre parole un pixel è una "**sequenza**" di uno o più livelli di luminosità, secondo il numero di bande che lo costituiscono.

Nella fase di segmentazione si deve tener conto non solo della luminosità del singolo pixel ma anche del **contesto** in cui esso si trova. Più precisamente, per stabilire il cluster di appartenenza di ciascun pixel consideriamo anche quelli che cadono in un intorno (quadrato), centrato nel pixel considerato, di diametro *n*.

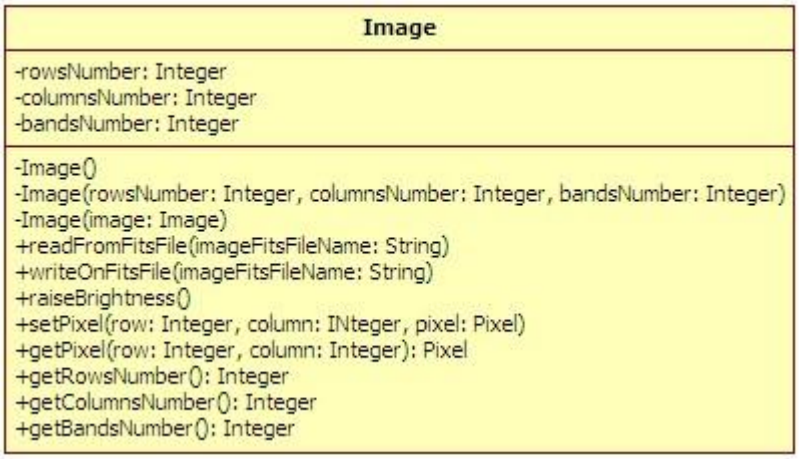

Figura 17 – La classe Image

Figura 18 – La classe Pixel

2.1.2.2 La classe Pattern

Nella libreria NExTII i pattern sono vettori utilizzati per la rappresentazione dei pixel dell'immagine. La lunghezza di tali vettori è data da (n x n) x b, dove:

- n è un numero naturale, dispari e maggiore di uno, che rappresenta il diametro dell'intorno del pixel;
- b è il numero di bande[5] di ogni pixel dell'immagine.

[5] Per "banda" si intende la lunghezza d'onda rilevata dai sensori utilizzati durante il rilevamento. Per immagine multi-banda si intende un'immagine acquisita su diverse lunghezze d'onda.

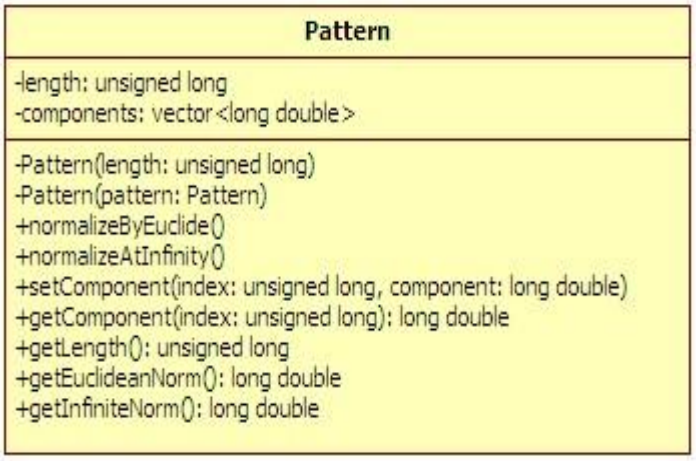

Figura 19 – La classe Pattern

Le componenti del vettore sono rappresentate dalle sequenze luminose dei pixel che cadono nell'intorno (pixel centrale incluso) L'ordine di concatenazione parte dall'angolo in alto a sinistra dell'intorno e termina nell'angolo in basso a destra dello stesso, procedendo da sinistra a destra e dall'alto verso il basso.

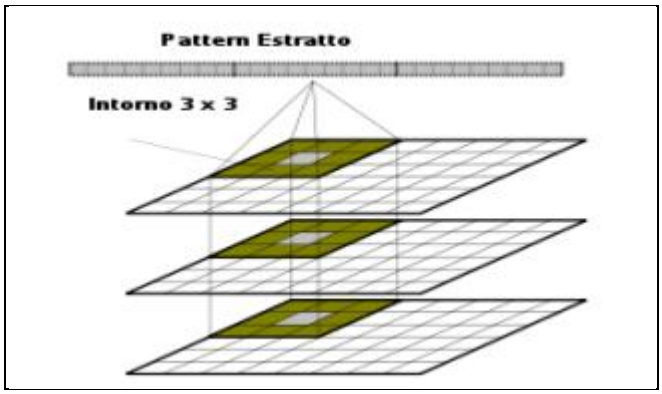

Figura 20 – Esempio di estrazione di un pattern di un'immagine multi-banda

2.1.2.3 La classe UnsupervisedNN

Come già accennato, NExTII fa uso di alcune reti neurali con apprendimento di tipo non supervisionato per la segmentazione delle immagini. Tali reti sono rappresentate dalla classe astratta UnsupervisedNN, che fornisce metodi e operazioni per l'addestramento delle reti neurali sottostanti e, inoltre, operazioni di normalizzazione dei dati di input.

La fase di addestramento è rappresentata dal metodo

train(trainingPattern:Pattern[*])

Tale fase è suddivisa in due sottofasi:

- **Preparazione all'apprendimento:** in questa fase è inizializzata la struttura della rete (inizializzazione dei pesi associati[6] ai neuroni e di altri parametri) ed è rappresentata tramite il metodo

  prepareToLearn(trainingPattern:Pattern[*]);

- **Apprendimento**: in questo caso si tratta di apprendimento non supervisionato; la rete determina e aggiorna i pesi associati ai neuroni esclusivamente basandosi sui dati di input senza avere conoscenze pregresse dell'output. Questa fase permette alla rete di apprendere una "rappresentazione", più generica possibile, dell'insieme degli input possibili (training set), in modo tale da poter rispondere, una volta addestrata, a nuovi input non presenti nel training set.

  Tale fase è rappresentata dal metodo:

  learn(trainingPattern:Pattern[*])

[6] Nella libreria NExTII il vettore dei pesi associati a ciascun neurone della rete è noto anche come "pattern di riferimento";

Figura 21 – La classe UnsupervisedNN

Entrambi i metodi sono astratti e sono specializzati in base alle esigenze della rete neurale realizzata.

2.1.2.4 La classe ClusteringNN

La classe astratta ClusteringNN rappresenta alcune reti neurali utili per eseguire il clustering delle immagini. La principale caratteristica è di avere un numero di neuroni non inferiore a due, indipendentemente dalla lunghezza specificata per i pattern di input.

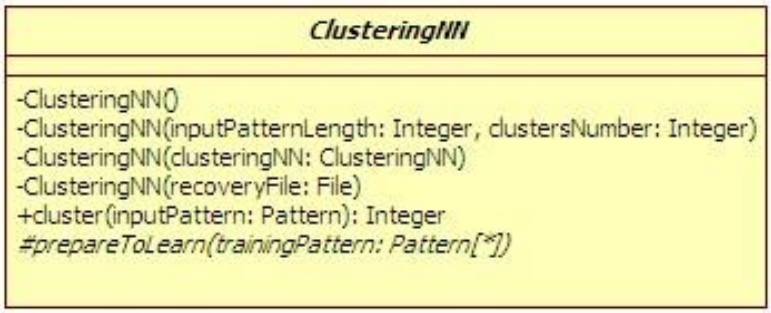

Figura 22 – La classe ClusteringNN

L'individuazione del cluster di appartenenza di un pattern di input è rappresentata dal metodo:

cluster(inputPattern:Pattern):integer

che riporta in output l'indice del neurone il cui pattern di riferimento[7] ha la minima distanza dal pattern di input.

2.1.2.5 La classe FrozenMorphologyClusteringNN

La classe astratta FrozenMorphologyClusteringNN rappresenta alcune reti neurali per il clustering che hanno uno schema di interconnessione fisso durante la fase di addestramento.

Figura 23 – La classe FrozenMorphologyClusteringNN

[7] Per pattern di riferimento si intende il vettore dei pesi associato a ciascun neurone.

Le FrozenMorphologyClusteringNN hanno le seguenti proprietà:

- un tasso di apprendimento[8] iniziale **(***initialLearningRate)***,** numero reale compreso tra 0 e 1;
- un tasso di apprendimento finale (*finalLearningRate*), numero reale compreso tra 0 e 1 e minore del tasso iniziale di apprendimento;
- il numero di epoche (*learningEpochsNumber*);
- in NExTII sono suddivise in:
  - KMeansClusteringNN;
  - MaximumEntropyClusteringNN;
  - NeuralGasClusteringNN;
  - SOMClusteringNN

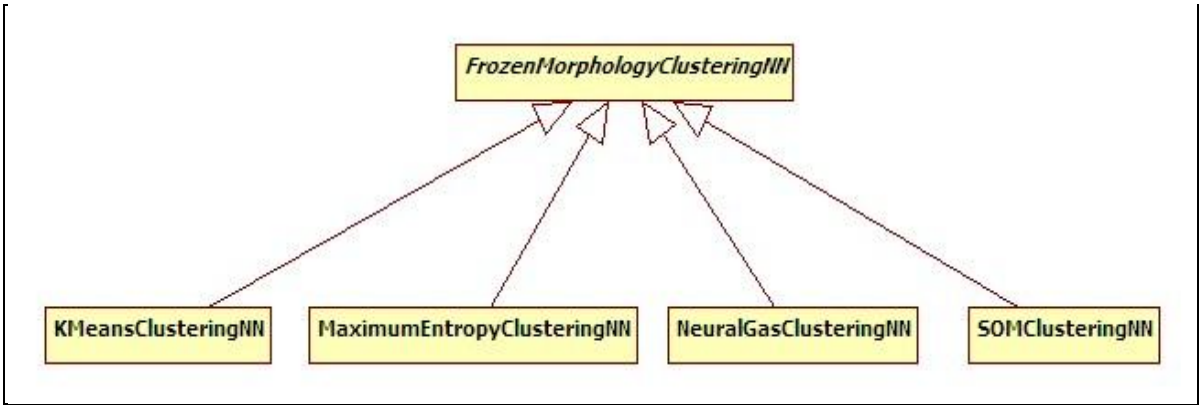

Figura 24 – Tipi di ForzenMorphologyClusteringNN

2.1.2.6 La classe SOMClusteringNN

In NExTII sono state implementate reti neurali SOM con apprendimento non supervisionato, le SOMClusteringNN, che riprendono il modello di Teuvo Kohonen ed hanno le seguenti proprietà:

- Una varianza[9] iniziale (*initialVariance)* per la fase di addestramento, con valore reale positivo;
- Una varianza finale (*finalVariance*) per la fase di addestramento, con valore reale positivo inferiore o al più pari alla varianza iniziale;

[8] Il tasso di apprendimento specifica la velocità di apprendimento della rete neurale;
[9] La varianza è utilizzata durante la fase di aggiornamento dei pattern di riferimento dei neuroni.

- I neuroni organizzati in una griglia con un dato numero di righe (*rowsNumber*) e colonne (*columnsNumber*), tali che il prodotto righe per colonne sia uguale al numero di neuroni specificati per la rete.

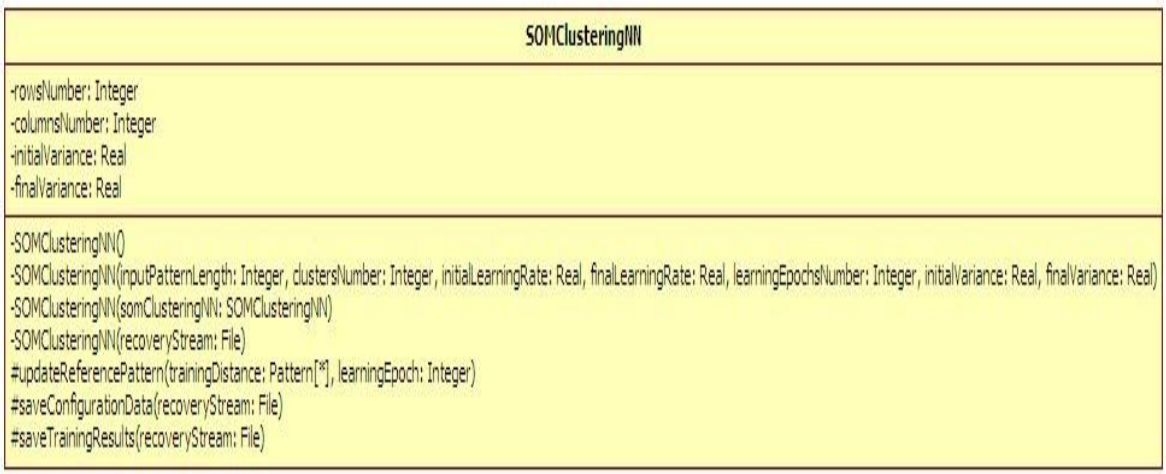

Figura 25 – La classe SOMClusteringNN

2.1.2.7 La classe MultiLayerClusteringNN

In NExTII le reti neurali multi-livello per il clustering, dette MultiLayerClusteringNN, e sono state introdotte per risolvere due tipi di problemi:

- **Sotto-dimensionamento di una rete**: il numero di neuroni della rete è pari al numero di cluster desiderati ma è un numero troppo piccolo; di conseguenza, si potrebbe ottenere un'insufficiente capacità di generalizzazione della rete, ovvero una sovrapposizione delle classi, fenomeno noto come *over-fitting*;
- **Etichettatura (o *labeling*) manuale dei neuroni**: associazione a ogni neurone di un'etichetta di classe al termine dell'addestramento.

Una MultiLayerClusteringNN ha le seguenti proprietà:

- è una struttura composta da un dato numero di livelli *(layersNumber*);
- offre le stesse funzionalità di base di una generica ClusteringNN: addestramento e clustering;
- ciascun livello può essere configurato per usare un tipo differente di ClusteringNN (KMeansClusteringNN, SOMClusteringNN, ecc..), oppure i livelli possono essere configurati tutti con lo stesso tipo di rete neurale (ad esempio, ogni livello configurato con una SOMClusteringNN);
- i vari livelli hanno un numero decrescente di neuroni (forma piramidale della rete);

Tale classe offre operazioni di configurazioni dei livelli che prevedono l'utilizzo di un qualsiasi delle reti con apprendimento non supervisionato presenti in NExTII utilizzate per il clustering.

Figura 26 – La classe MultiLayerClusteringNN

2.2 Il componente Data Mining Model (DMM)

Il componente Data Mining Model (**DMM**), rappresenta gli algoritmi utilizzati nella suite DAME per l'esecuzione di esperimenti di data mining. Esso è stato realizzato con tecnologia JAVA e in modo tale da favorire il riutilizzo del codice e il polimorfismo, tramite l'uso di un particolare pattern strutturale, il Bridge Pattern, che permette di evitare la duplicazione di codice.

2.2.1 Il Bridge Pattern

La progettazione del DMM, realizzata dal collega Alessandro Di Guido con la collaborazione del Dott. Stefano Cavuoti si è basata prevalentemente sulla separazione tra classi di funzionalità (classificazione e regressione) e modelli di reti neurali. Tale scelta strutturale è dovuta al fatto che i modelli implementati possono essere utilizzati sia per la classificazione sia per la regressione; in particolare, le classi sono state progettate seguendo la struttura del Bridge Pattern, evitando la duplicazione del codice, fonte di difficile individuazione degli errori e di spreco di tempo e di risorse.

Come accennato, l'esigenza di separare il livello di astrazione dal livello di implementazione ha portato all'utilizzo del Bridge Pattern.

Il Bridge Pattern è un design pattern che ha lo scopo principale di separare un'astrazione dalla sua implementazione, in modo tale che esse possano variare indipendentemente.

Nella figura in basso è rappresentata la struttura del Bridge Pattern.

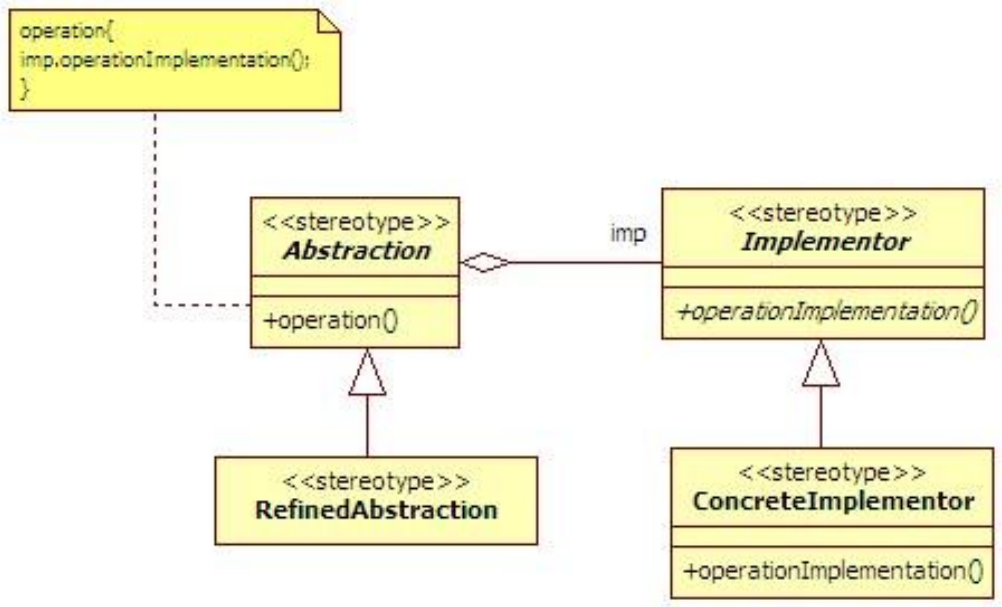

Figura 27 – Struttura del Bridge Pattern.

Esso è composto da :

- classe astratta **Abstraction**: specifica l'interfaccia dell'astrazione e gestisce un riferimento ad un oggetto Implementor; essa specifica i metodi di base per agire sulla classe Implementor;
- classe **RefinedAbstraction**: implementa l'interfaccia definita dall'Abstraction;
- classe astratta **Implementor**: specifica l'interfaccia definita per le classi di implementazione;
- classe **ConcreteImplementor**: implementano l'interfaccia Implementor

2.2.2 La struttura del DMM

Nel paragrafo precedente è stata data una descrizione del pattern su cui si basa la struttura del componente DMM. In particolare, nel diagramma si può osservare la struttura generale del DMM ottenuta tramite l'applicazione del Bridge Pattern:

Figura 28 – Diagramma UML del DMM.

Il componente DMM è suddiviso in due livelli:

- **abstraction layer** : la classe **Supervised** , rappresentante la categoria degli algoritmi di tipo supervisionato, e le classi **Classification** e **Regression**, che rappresentano le funzionalità ;

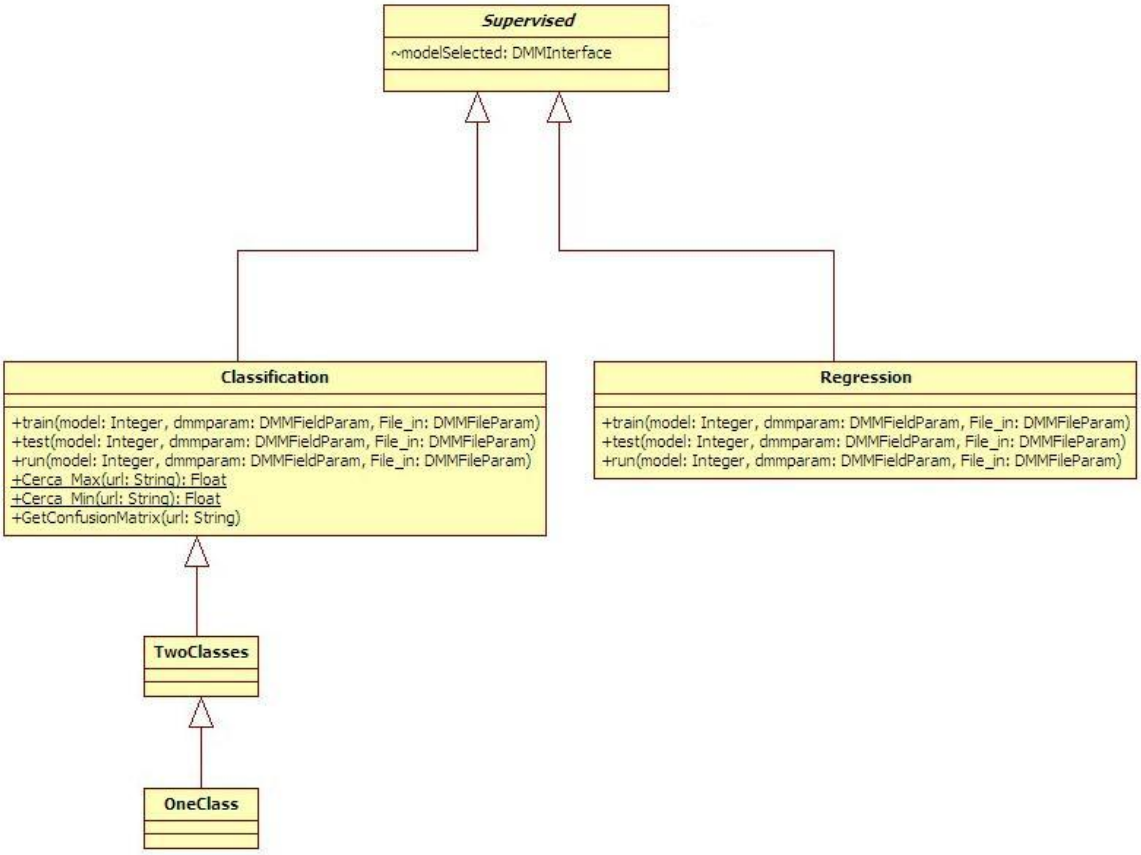

Figura 29 – Abstraction Layer del DMM.

- **implementation layer:** un'interfaccia **DMMInterface** che rappresenta un generico modello di data mining e le classi **MLP**, **SVM** e **MLPGA**, modelli già implementati nella suite (Capitolo 1);

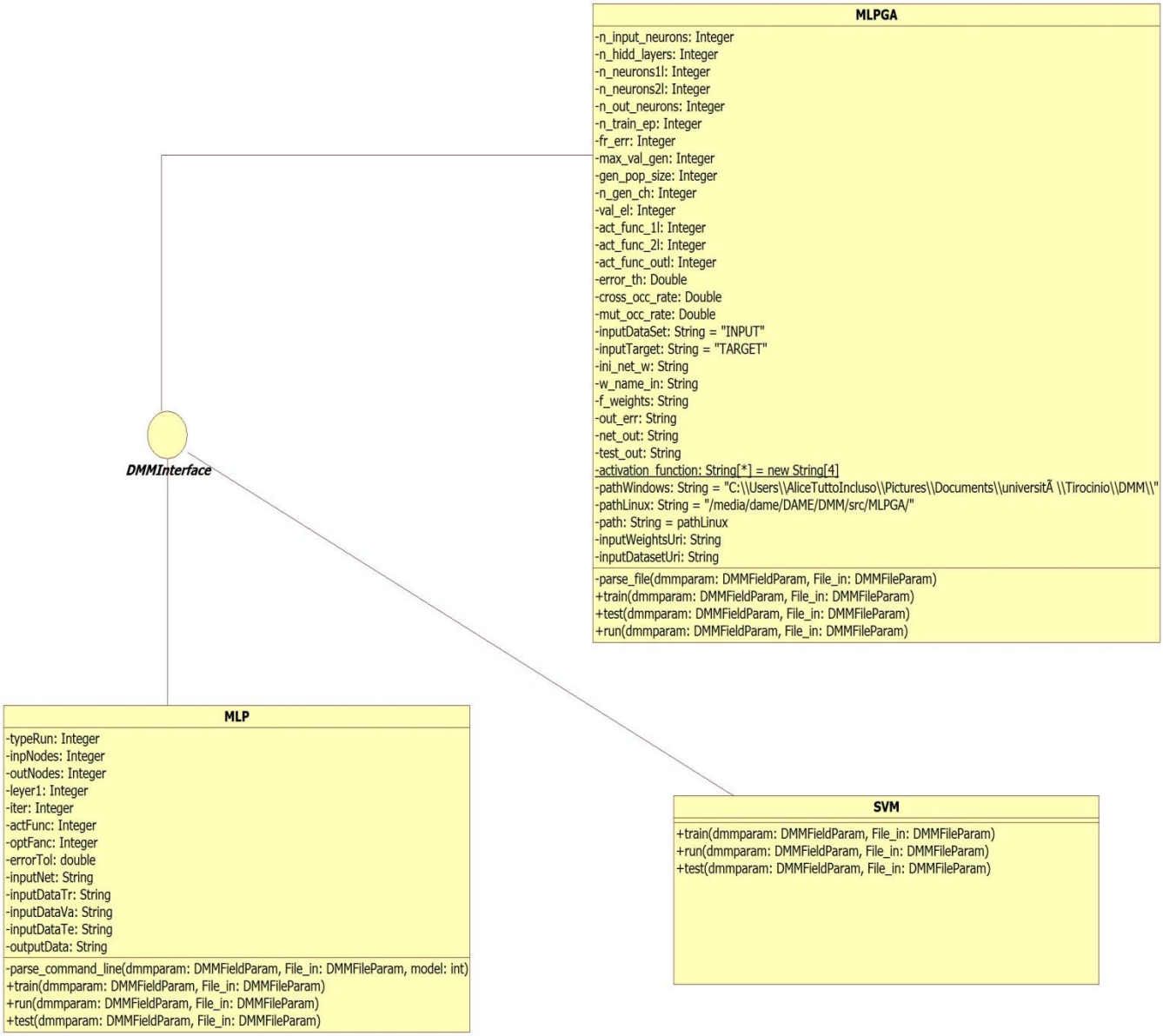

Figura 30 – Implementation Layer del DMM.

Inoltre , nel DMM, sono presenti anche:

- la classe **Statistic**: utile per eseguire statistiche sul dataset di input;
- la classe **Visualization**: permette la produzione di grafici e immagini a partire da file di input ASCII;

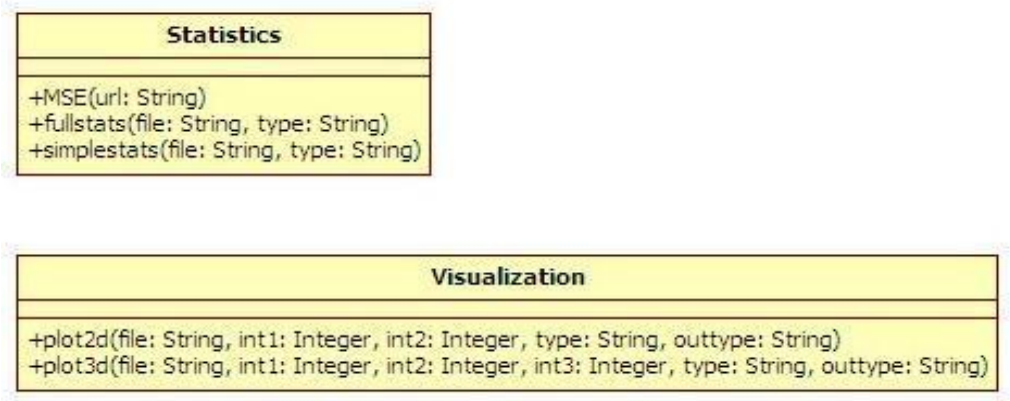

Figura 31 – Le classi Statistics e Visualization.

2.2.3 Il componente DmmParams

Il pacchetto **DmmParams** è stato creato per ottenere una generalizzazione degli elementi necessari per la configurazione dei modelli implementati: parametri, file di input e di output. Il DmmParams è costituito dalle seguenti classi:

- **DmmFieldParam**: contiene attributi utili per la rappresentazione dei parametri dei modelli e metodi per inserire e ottenere informazioni riguardanti il parametro preso in considerazione in un dato istante;
- **DmmFileParam**: contiene attributi e metodi utili per la rappresentazione dei file di input o input-output;
- **DmmFileOutputParam**: contiene attributi e metodi utili per la rappresentazione dei file di output ottenuti dall'esecuzione di un esperimento di data mining tramite l'utilizzo di uno dei modelli presenti nella suite;
- **Tag**: contiene metodi utili per associare un nome, o *tag* alle colonne del file di input;
- **Constraint**: definisce i vincoli sul valore dei campi dati in input:
    - **NO_CONSTRAINT**: il campo in input non ha vincoli;
    - **RANGE**: il valore è compreso in un dato intervallo di cui sono specificati gli estremi;
    - **VALUES**: i campi assumono i valori enumerati;

La struttura generale del pacchetto DmmParams è rappresentata nella figura in basso.

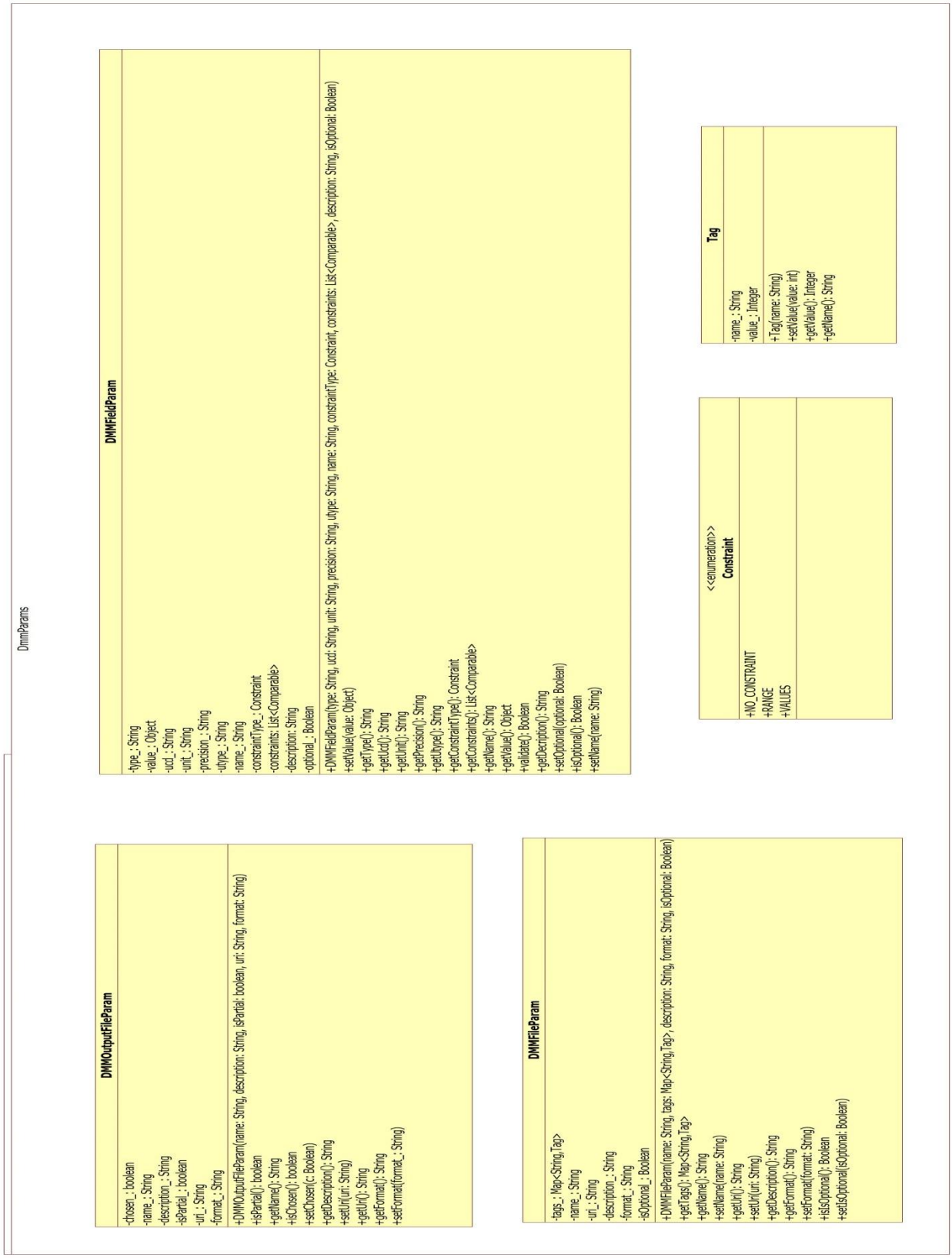

Figura 32 – Il pacchetto DmmParams

2.3 Il componente Data Mining Plugin (DMPlugin)

La suite DAME offre varie funzionalità che possono essere scelte dall'utente per l'esecuzione di un esperimento di data mining. Tali funzionalità sono rappresentate tramite i DMPlugin, blocchi di codice implementati come librerie condivise caricate dinamicamente a run-time dal programma principale. La struttura del DMPlugin permette a uno sviluppatore (anche esterno al team di lavoro) di inserire nuove funzionalità all'interno della suite.

Capitolo 3. Sviluppo della soluzione adottata

In questo capitolo si descrive la soluzione al problema affrontato, partendo dall'estensione della libreria NExTII, del componente DMM, fino allo sviluppo dei DMPlugin che permettono l'utilizzo del modello SOM all'interno della suite DAME.

3.1 Estensione di NExTII

Nel Capitolo 2 è stata descritta la libreria NExTII che, come visto, fornisce una serie di classi e operazioni utilizzabili e molto utili nel lavoro da me svolto.

Durante l'analisi e il reverse engeeniring della suddetta libreria è emerso che:

- La classe Image può essere utilizzata esclusivamente per la rappresentazione di immagini di tipo FITS standard, cioè comprensivi di un HDU e di una estensione di tipo IMAGE_HDU; ciò ha portato all'introduzione di una nuova classe, ImageTable, che permette di rappresentare dataset con diversi tipi di estensione e immagini di tipo Fits con estensione di tipo ASCII_TABLE;
- La classe MultiLayerClusteringNN fornisce operazioni utili per la configurazione di ciascun livello di rete e può essere sfruttata per eseguire operazioni di clustering multi-livello;
- Occorre la creazione di una classe di gestione della libreria che permetta la configurazione e l'utilizzo di una rete multi-livello, l'estrazione dei dati dai file, ecc.;
- Occorre creare un "mezzo" per l'utilizzo della libreria NExTII nella suite DAME.

Il lavoro riguardante NExTII è stato svolto utilizzando il linguaggio C++ in ambiente Linux/Unix.

3.1.1 La classe ImageTable

Tale classe è stata introdotta per permettere l'utilizzo da parte degli utenti di vari tipi di file. Più precisamente, come già accennato, la classe Image, della libreria NExTII, rappresenta solo immagini di tipo Fits standard, mentre la classe ImageTable è stata da me creata per rappresentare anche altri tipi di immagini o dati e per permettere l'utilizzo del tipo general-purpose di rete SOM, utilizzata per effettuare esperimenti su insiemi di dati rappresentati in forma tabellare.

Più nel dettaglio, la classe è fornita di metodi per la lettura/scrittura di file con estensione "**.dat**" o "**.csv**" contenenti dati e tabelle, file comunemente utilizzati all'interno della suite DAME.

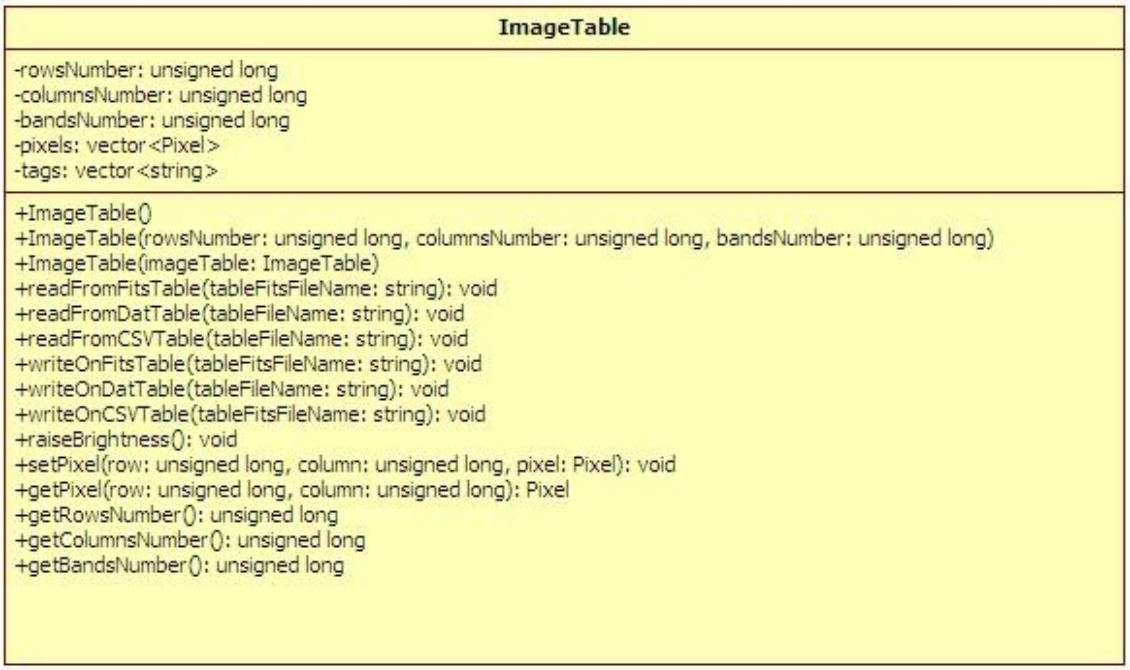

Figura 33 – Classe ImageTable

L'aggiunta di tale classe permette all'utente un maggior grado di libertà sulla scelta di file su cui eseguire le operazioni di clustering.

3.1.2 La classe NextControl

La classe NextControl è stata introdotta per la gestione della libreria NExTII. Essa contiene metodi che permettono di eseguire l'addestramento, l'estrazione dei pattern e il clustering .

Allo stato dell'arte, la libreria NExTII permette la segmentazione delle immagini astronomiche esclusivamente utilizzando architetture multi-livello con un numero fisso di tre livelli di rete, configurabili tramite file di input che contengono tutti i parametri per ciascun livello di rete.

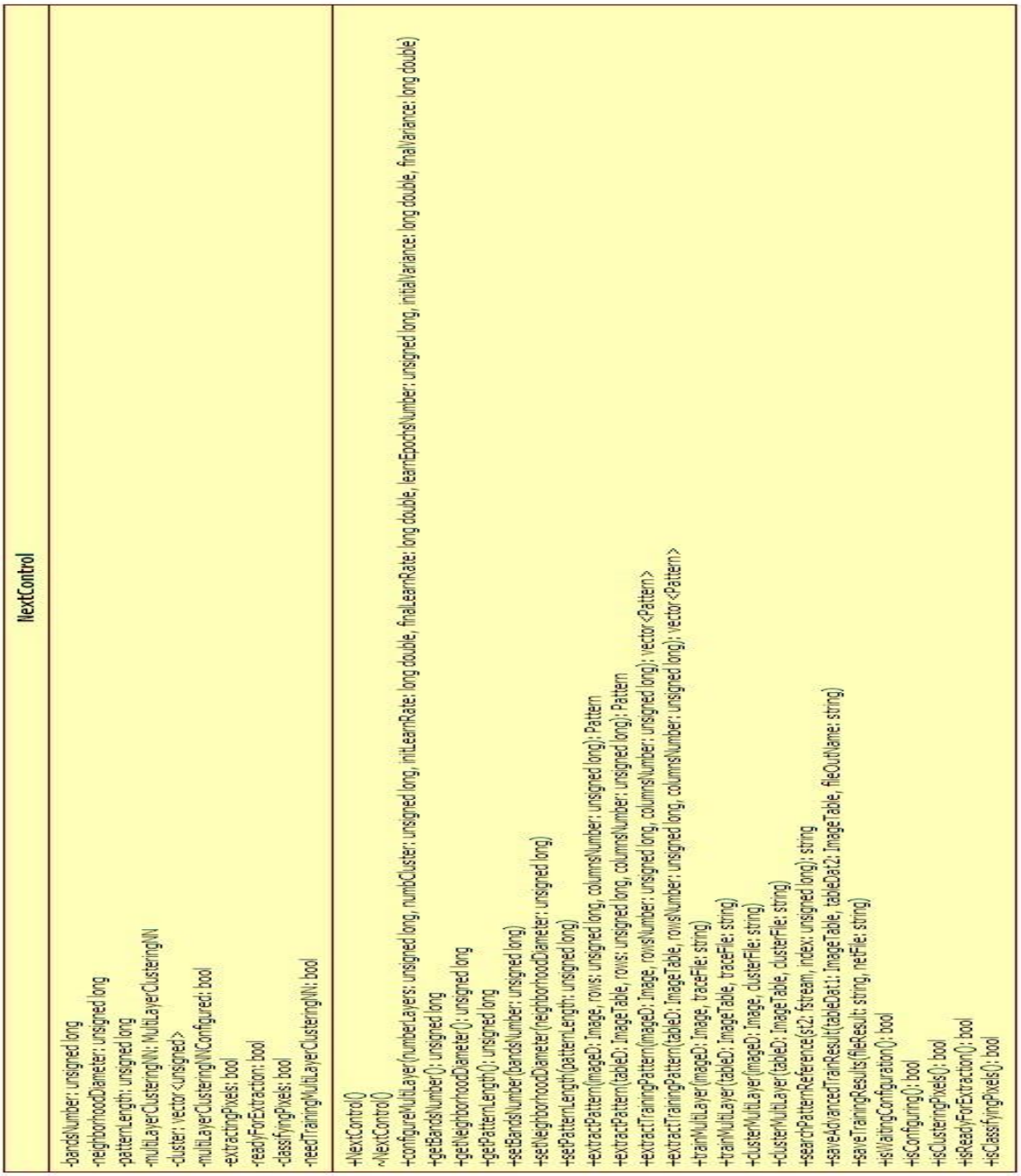

Figura 34 – Classe NextControl

La classe NextControl supera questo limite utilizzando il metodo

configureMultiLayer(...),

che permette la configurazione di una rete multi-livello e, non implica l'utilizzo "*forzato*" di tre livelli, ma tale numero varia in un intervallo tra 1 e 3, ed è specificato dall'utente durante l'impostazione dei parametri. Questo tipo di soluzione permette all'utente sia l'uso di un singolo modello di rete sia l'uso di reti multi-livello.

3.1.3 Il controllo di NExTII: NextMain

NextMain si occupa del controllo di tutti i parametri dati in input dall'utente e della gestione degli errori. L'esecuzione di tale file produce in output, oltre i file richiesti da ogni specifico caso d'uso utilizzato, un file di output, denominato "*StatusExperiment.log*" in cui sono elencati tutti i parametri di configurazione della rete multi-livello utilizzata oppure l'errore rilevato durante l'esecuzione dell'esperimento.

Di seguito è illustrato qualche tabelle, con la descrizione dei parametri da passare come argomento alla libreria nei casi più interessanti:

- Caso d'uso "train" (typeRun=1) o "run" (typeRun = 2) e un solo livello:

Argomento	Variabile Argomento	Tipo Argomento	Descrizione
Argv[1]	typeRun	Unsigned Long	Tipo di caso d'uso:{ 1 per il train; 2 per il run; 3 per il test}
Argv[2]	numberLayers	Unsigned Long	Numero totale di livelli della rete multi-livello
Argv[3]	typeNet	Unsigned Long	Tipo di rete neurale da utilizzare
Argv[4]	diameter	Unsigned Long	Intorno scelto per l'estrazione dei pattern
Argv[5]	epochs	Unsigned Long	Numero di epoche
Argv[6]	numNeurons	Unsigned Long	Numero neuroni primo livello
Argv[7]	initLearnRate	Long Double	Initial learning Rate primo livello
Argv[8]	finLearnRate	Long Double	Final learning Rate
Argv[9]	initVar	Long Double	Initial variance primo livello
Argv[10]	finVar	Long Double	Final variance primo livello
Argv[11]	Dataset1	String	Input dataset

Tab. 1 – Parametri casi d'uso: train o run, un livello di rete

- Caso d'uso "test"(typeRun = 3) e un solo livello:

Argomento	Variabile Argomento	Tipo Argomento	Descrizione
Argv[1]	typeRun	Unsigned Long	Tipo di caso d'uso:{ 1 per il train; 2 per il run; 3 per il test}
Argv[2]	numberLayers	Unsigned Long	Numero di livelli della rete multi-livello
Argv[3]	typeNet	Unsigned Long	Tipo di rete neurale da utilizzare
Argv[4]	diameter	Unsigned Long	Intorno scelto per l'estrazione dei pattern
Argv[5]	diameter	Unsigned Long	Intorno scelto per l'estrazione dei pattern
Argv[6]	epochs	Unsigned Long	Numero di epoche
Argv[7]	numNeurons	Unsigned Long	Numero di neuroni
Argv[8]	initLearnRate	Long Double	Tasso di apprendimento iniziale
Argv[9]	finLearnRate	Long Double	Tasso di apprendimento finale
Argv[10]	initVar	Long Double	Varianza iniziale
Argv[11]	finVar	Long Double	Varianza finale
Argv[12]	Dataset1	String	Primo dataset di input
Argv[13]	Dataset1	String	Secondo dataset di input

Tab. 2 – Paramatri caso d'uso "test", un solo livello di rete

- Caso d'uso "test" e due livelli:

Argomento	Tipo Argomento	Tipo Argomento	Descrizione
Argv[1]	typeRun	Unsigned Long	Tipo di caso d'uso:{ 1 per il train; 2 per il run; 3 per il test}
Argv[2]	numberLayers	Unsigned Long	Numero livelli della rete multi-livello
Argv[3]	typeNet	Unsigned Long	Tipo di rete neurale da utilizzare
Argv[4]	diameter	Unsigned Long	Intorno scelto per l'estrazione dei pattern
Argv[5]	epochs	Unsigned Long	Numero di epoche
Argv[6]	numNeurons	Unsigned Long	Numero di neuroni per il primo livello

Argv[7]	initLearnRate	Long Double	Tasso di apprendimento iniziale per il primo livello
Argv[8]	finLearnRate	Long Double	Tasso di apprendimento finale per il primo livello
Argv[9]	initVar	Long Double	Varianza iniziale per il primo livello
Argv[10]	finVar	Long Double	Varianza finale per il primo livello
Argv[11]	numNeurons2	Unsigned Long	Numero neuroni per il secondo livello
Argv[12]	initLearnRate2	Long Double	Tasso di apprendimento iniziale per il secondo livello
Argv[13]	finLearnRate2	Long Double	Tasso di apprendimento finale per il secondo livello
Argv[14]	initVar2	Long Double	Varianza iniziale per il secondo livello
Argv[15]	finVar2	Long Double	Varianza finale per il secondo livello
Argv[16]	Dataset1	String	Primo dataset di input
Argv[17]	Dataset2	String	Secondo dataset di input

Tab. 3 - Parametri casi d'uso "test" ,due livello di rete

- Caso d'uso "train" o "run" e tre livelli:

Argomento	**Variabile Argomento**	**Tipo Argomento**	**Descrizione**
Argv[1]	typeRun	Unsigned Long	Tipo di caso d'uso:{ 1 per il train; 2 per il run; 3 per il test}
Argv[2]	numberLayers	Unsigned Long	Numero di livelli della rete multi-livello
Argv[3]	typeNet	Unsigned Long	Tipo di rete neurale da utilizzare
Argv[4]	diameter	Unsigned Long	Intorno scelto per l'estrazione dei pattern
Argv[5]	epochs	Unsigned Long	Numero di epoche
Argv[6]	numNeurons	Unsigned Long	Numero di neuroni per il primo livello
Argv[7]	initLearnRate	Long Double	Tasso di apprendimento iniziale per il primo livello
Argv[8]	finLearnRate	Long Double	Tasso di apprendimento finale per il primo livello
Argv[9]	initVar	Long Double	Varianza iniziale per il primo livello
Argv[10]	finVar	Long Double	Varianza finale per il primo livello

Argv[11]	numNeurons2	Unsigned Long	Numero neuroni per il secondo livello
Argv[12]	initlearnRate2	Long Double	Tasso di apprendimento iniziale per il secondo livello
Argv[13]	finLearnRate2	Long Double	Tasso di apprendimento finale per il secondo livello
Argv[14]	initVar2	Long Double	Varianza iniziale per il secondo livello
Argv[15]	finVar2	Long Double	Varianza finale per il secondo livello
Argv[16]	numNeurons3	Unsigned Long	Numero neuroni per il terzo livello
Argv[17]	initlearnRate3	Long Double	Tasso di apprendimento iniziale per il terzo livello
Argv[18]	finLearnRate3	Long Double	Tasso di apprendimento finale per il terzo livello
Argv[19]	initVar3	Long Double	Varianza iniziale per il terzo livello
Argv[20]	finVar3	Long Double	Varianza finale per il terzo livello
Argv[21]	Dataset1	String	Input dataset

Tab. 4 - Parametri casi d'uso:train o run, tre livelli di rete

- Caso d'uso "test" e tre livelli:

Argomento	**Variabile Argomento**	**Tipo Argomento**	**Descrizione**
Argv[1]	typeRun	Unsigned Long	Tipo di caso d'uso:{ 1 per il train; 2 per il run; 3 per il test}
Argv[2]	numberLayers	Unsigned Long	Numero di livelli della rete multi-livello
Argv[3]	typeNet	Unsigned Long	Tipo di rete neurale da utilizzare
Argv[4]	diameter	Unsigned Long	Intorno scelto per l'estrazione dei pattern
Argv[5]	epochs	Unsigned Long	Numero di epoche
Argv[6]	numNeurons	Unsigned Long	Numero di neuroni per il primo livello
Argv[7]	initLearnRate	Long Double	Tasso di apprendimento iniziale per il primo livello
Argv[8]	finLearnRate	Long Double	Tasso di apprendimento finale per il primo livello
Argv[9]	initVar	Long Double	Varianza iniziale per il primo livello
Argv[10]	finVar	Long Double	Varianza finale per il primo livello

Argv[11]	numNeurons2	Unsigned Long	Numero neuroni per il secondo livello
Argv[12]	initlearnRate2	Long Double	Tasso di apprendimento iniziale per il secondo livello
Argv[13]	finLearnRate2	Long Double	Tasso di apprendimento finale per il secondo livello
Argv[14]	initVar2	Long Double	Varianza iniziale per il secondo livello
Argv[15]	finVar2	Long Double	Varianza finale per il secondo livello
Argv[16]	numNeurons3	Unsigned Long	Numero neuroni per il terzo livello
Argv[17]	initlearnRate3	Long Double	Tasso di apprendimento iniziale per il terzo livello
Argv[18]	finLearnRate3	Long Double	Tasso di apprendimento finale per il terzo livello
Argv[19]	initVar3	Long Double	Varianza iniziale per il terzo livello
Argv[20]	finVar3	Long Double	Varianza finale per il terzo livello
Argv[21]	Dataset1	String	Primo dataset di input
Argv[22]	Dataset2	String	Secondo dataset di input

Tab. 5 - Parametri casi d'uso "test", tre livelli di rete

NextMain si occupa di verificare e controllare che i parametri sopra specificati siano corretti per ogni caso d'uso e numero di livelli scelti. Inoltre in output si avranno i seguenti file, in base al caso d'uso specificato dall'utente:

- MLCSOMTrain.log : file contenente dati riguardanti la rete multi-livello dopo il l'addestramento;
- MLCSOMCluster.log: file contenente i dati riguardanti la rete multi-livello dopo il clustering;
- overlap.log : file utilizzato nel caso d'uso "test"
- stat.log: file contenente i risultati del caso d'uso "test";
- StatusExperiment.log: file contenente i parametri riguardanti l'intero esperimento e il tipo di errore se l'esperimento non va a buon fine.

Nel caso del test i file contenenti i dati riguardanti la rete sono:

- MLCSOMDAT1: contenente i valori della rete sul primo dataset di input;.

- MLCSOMDAT2: contenente i valori della rete sul secondo dataset di input.

3.2 Estensione del DMM

Finora l'unica categoria di apprendimento implementata nella suite DAME era la categoria Supervised, come visto nel Capitolo 1.

Il mio lavoro ha avuto come scopo principale l'inserimento nella suite della nuova categoria di apprendimento Unsupervised, unita alla funzionalità Clustering e al modello generico di rete neurale, chiamato "*generic SOM*" (GSOM). Tale modello è stato poi specializzato in un' ulteriore classe, "*clustering SOM*" (CSOM) che esegue i metodi train, run, test e full solo su file di input di tipo FITS.

Per lo sviluppo del componente DMM della suite DAME è stato scelto Java 1.5 per i seguenti motivi:

- Compatibilità: grazie alla *Java Virtual Machine* (JVM), le applicazioni Java sono compatibili a quasi tutte le piattaforme esistenti; tale caratteristica è fondamentale poiché la suite è stata progettata per lavorare su diverse piattaforme computazionali (GRID/CLOUD e in generale *High Performance Computing*, HPC), sia in ambiente distribuito, sia in modalità Stand Alone;
- Alta astrazione dalla macchina fisica: ciò implica semplicità di sviluppo, uniformità e portabilità, e di conseguenza impedisce l'uso di caratteristiche specifiche di una determinata macchina o sistema operativo;
- Velocità di sviluppo: data la sua semplicità di utilizzo;
- Grande disponibilità di librerie: che possono essere utilizzate in qualsiasi tipo di applicazione e riducendo i tempi di realizzazione;
- Alta integrazione con il web: caratteristica fondamentale per la suite;
- Sicurezza: il codice è più robusto e si elimina il rischio di operazioni dannose per il sistema come l'uso improprio di puntatori.

3.2.1 Adattamento della soluzione al Bridge Pattern

Il primo passo per lo sviluppo della soluzione è stato l'adattamento della stessa alla struttura già esistente, ovvero l'inserimento delle nuove classi **Unsupervised**, **Clustering**, **GSOM** e **CSOM** utilizzando il pattern strutturale Bridge Pattern.

3.2.2 Livello di astrazione: le classi Unsupervised e Clustering

L'implementazione del livello funzionale riguarda la dichiarazione della classe astratta "Unsupervised" che generalizza le funzionalità senza supervisione e la creazione della classe di funzionalità "Clustering" che permette l'utilizzo del modello SOM.

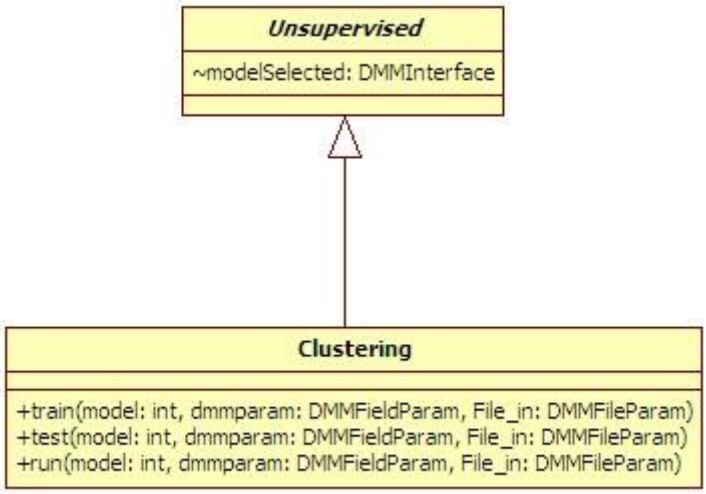

Figure 35 – Le classi Unsupervised e Clustering

Per chiarire il compito svolto dalla classe Clustering mostriamo il codice:

```
public class Clustering extends Unsupervised {
    public void train(int model, DMMFieldParam[] dmmparam, DMMFileParam[] File_in) throws
  IOException {
      // the param model is a flag to select a model
      // 0 SOM Clustering + NEXTII
      // 1 SOM Generic
      if (model == 0) {
         modelSelected = new CSOM();
         modelSelected.train(dmmparam, File_in);
      }
      if (model == 1) {
         modelSelected = new GSOM();
         modelSelected.train(dmmparam, File_in);
      }
   }
```

```
  public void test(int model, DMMFieldParam[] dmmparam, DMMFileParam[] File_in) throws
IOException {
    // the param model is a flag to select a model
    // 0 SOM Clustering + NEXTII
    // 1 SOM Generic
    if (model == 0) {
      modelSelected = new CSOM();
      modelSelected.test(dmmparam, File_in);
    }
    if (model == 1) {
      modelSelected = new GSOM();
      modelSelected.test(dmmparam, File_in);
    }
  }
  public void run(int model, DMMFieldParam[] dmmparam, DMMFileParam[] File_in) throws
IOException {
    // the param model is a flag to select a model
    // 0 SOM Clustering + NEXTII
    // 1 SOM Generic

    if (model == 0) {
      modelSelected = new CSOM();
      modelSelected.run(dmmparam, File_in);
    }
    if (model == 1) {
      modelSelected = new GSOM();
      modelSelected.run(dmmparam, File_in);
    }
  }
}
```

Come si può notare, per ogni caso d'uso si sceglie il modello da utilizzare per eseguire l'esperimento tramite l'utilizzo di un numero intero progressivo:

- 0 per il modello per il clustering su immagini fits CSOM;
- 1 per il modello general purpose GSOM.

Questo permetterà in futuro di aggiungere altri modelli di reti neurali non supervisionati, utilizzabili per eseguire il clustering in modo molto semplice e dà la possibilità al componente DMPlugin di indicare al DMM quale modello, tra quelli a disposizione, è stato selezionato dall'utente, e quindi di richiamare il metodo analogo al modello scelto. I due vettori di input sono oggetti della componente "DmmParams" (vedi paragrafo 2.2.3): il primo contiene tutti i parametri di configurazione del caso d'uso, l'altro contiene oggetti di tipo File di I/O, come ad esempio un file FITS; essi sono praticamente i parametri da dare in input alla libreria che implementa i modelli.

3.2.3 Livello di implementazione: il modello SOM

L'implementazione vera e propria dei modelli di apprendimento automatico consiste in un'interfaccia chiamata "*DMMInterface*" che generalizza i modelli con e senza supervisione a disposizione della suite DAME. Tale interfaccia è implementata da tutte le classi che sono responsabili dei modelli: SVM, MLP e MLPGA e i nuovi modelli CSOM e GSOM; essa prevede l'implementazione dei tre casi d'uso: train, test e run. In questo caso però, i tre metodi, dovranno gestire tutti i parametri di input, configurandoli nella maniera più opportuna per poter essere utilizzati dalle librerie richiamate.

Nei paragrafi successivi è descritta dettagliatamente la configurazione dei file e dei parametri di I/O delle librerie utilizzate. Al termine dell'inserimento delle classi CSOM e GSOM, il diagramma UML del livello di implementazione del DMM si presenta come nella figura che segue.

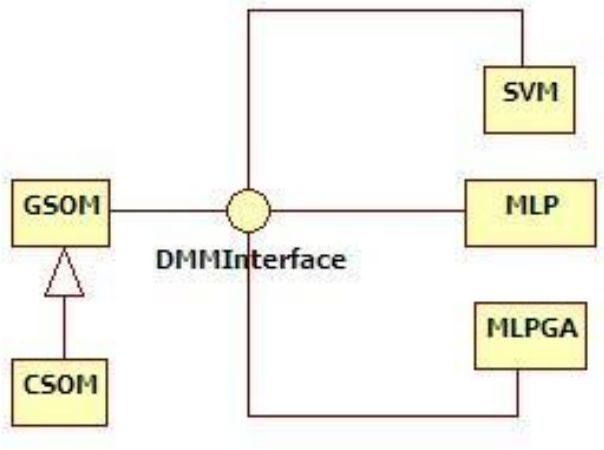

Figura 36 – Diagramma UML del livello di implementazione del DMM integrato con i modelli CSOM e GSOM

Le due classi, CSOM e GSOM, sono state implementate in modo tale da gestire il clustering multi-livello: il parametro "*numberLayers*" indica il numero di livelli di rete da utilizzare ed è specificato dall'utente.

3.2.3.1 Il modello GSOM

Tale modello di rete implementa i tre metodi train, test e run, dichiarati nell'interfaccia *DMMInterface*:

- ***train(dmmparam DMMFieldParam, File_In: DMMFileParam)***: rappresenta la fase di addestramento; questo metodo provvede alla configurazione dei parametri richiamando la funzione "***parse_ line()***" che scorre il vettore di oggetti "*DMMFieldParam*" chiamato *dmmparam*; questo vettore contiene i valori dei parametri del modello da impostare, fondamentali per un addestramento corretto. Alcuni parametri di input sono opzionali, quindi l'utente può decidere di non impostarli; di conseguenza, i parametri non impostati non saranno inseriti nel vettore dal FrameWork ma assumeranno un valore di default; ad ogni parametro è associato un nome e un valore per cui, all'interno del vettore, i parametri non assumono un ordine ben preciso; da ciò consegue la necessità del metodo "***parse_line()***"; il nome dei parametri è opportunamente scelto durante la fase di creazione del plugin di ogni modello, che sarà descritta nei prossimi paragrafi.

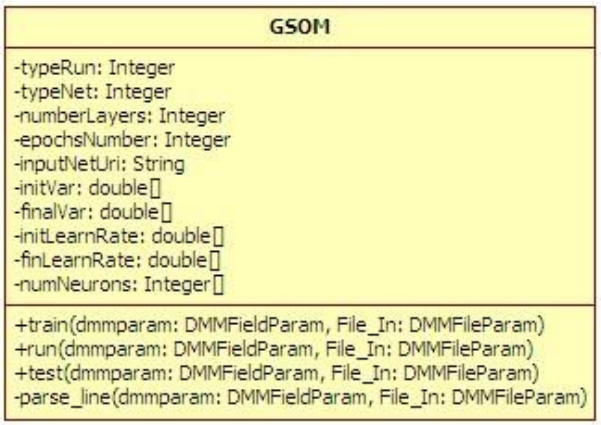

Figura 37 – La classe GSOM

Per utilizzare la libreria NExTII, scritta in linguaggio C++, all'interno di una suite implementata utilizzando tecnologia Java, è stato necessario l'utilizzo di una classe che permette l'esecuzione di un qualsiasi file: la classe **Runtime**, di cui mostriamo un esempio:

```
Runtime r=Runtime.getRuntime();
```

```
try {
        parseLine(dmmparam, File_in);
.....
        typeNet= 2;
        String path = "/data/DAME/suite/utilities/NEXTII/MLCSOM";
                        ...
        String cmd = path + " " + typeRun + " " + numberLayers + " "+ typeNet + " " + diameter + " "
        + epochsNumber;

        String  cmd2 = "sh genericSOM.sh";
        FileOutputStream somF = new FileOutputStream("genericSOM.sh");
        PrintStream ps=new PrintStream(somF);
        ps.println("#!/bin/bash\n"+cmd);

        Process p = r.exec(cmd2);
 .....
}
```

Come si evince dal codice, la prima cosa da fare è la creazione dell'oggetto di tipo "**Runtime**"; dopodiché si impostano i parametri attraverso il metodo "***parse_Line***"; si imposta il percorso (*path*) in cui è situata la libreria, si crea un file script con estensione "***.sh***" in cui è trascritto il comando da eseguire (la stringa "**cmd**"), infine, si esegue il metodo "**exec**"dell'oggetto Runtime; tale metodo esegue il comando dato come argomento nello stesso modo in cui si farebbe tramite linea di comando. Le tabelle contenenti tutti i parametri da inserire nella stringa "**cmd**" sono presenti nel paragrafo 3.1.3 del presente elaborato. Per maggior chiarezza, di seguito è indicato un esempio della stringa "**cmd**", nel caso si scelga un numero di livelli pari a 1 (vedi par. 3.1.3, Tab. 1 per l'associazione dei parametri):

"/data/DAME/suite/utilities/NEXTII/MLCSOM 1 1 1 5 50 50 0.7 0.05 4.0 0.1 M_101.fits"

- ***run(dmmparam DMMFieldParam, File_In: DMMFileParam)***: rappresenta la fase di utilizzo della rete neurale; questo metodo provvede, come nel metodo "train(...)" alla configurazione dei parametri richiamando la funzione "***parse_ line***"; anche in questo caso viene utilizzato un file script in cui è trascritto il comando da eseguire.
- ***test(dmmparam DMMFieldParam, Fiel_In: DMMFileParam):*** nel caso di tecniche di apprendimento supervisionato, rappresenta una valutazione di accuratezza dell'apprendimento della rete neurale utilizzata; più precisamente, si fornisce come input un insieme di dati, non utilizzato durante la fase di addestramento, ma di cui si conosce l'output e si calcolano le risposte corrette in termini percentuali. Nel caso di

tecniche di apprendimento non supervisionato in input è fornito un insieme di dati di cui non si conosce a priori l'output dato dalla fase di addestramento, di conseguenza, alla fase di test è associato un altro tipo di valutazione; in altre parole, anziché fornire una valutazione delle risposte corrette in termini percentuali, esso offre una valutazione statistica riguardante la "**stabilità**" del clustering effettuato. Per maggiore chiarezza, di seguito è riportato un esempio riguardante il procedimento eseguito durante questa fase:

Consideriamo un dataset di input S. I passi eseguiti sono i seguenti:

- si divide il dataset X di input in due sottoinsiemi, rispettivamente S1 e S2 secondo una **percentuale** ("*splitting percentage*"), indicata dall'utente, oppure di default impostata al 65%; questa percentuale rappresenta la prima parte di X (a partire dalla prima riga) che viene inserita in S1;
- S2 è costruito nel seguente modo:

  (X - S1) = D1; (D1 contiene il 35% restante di X)

  S2 = D1 + Random

  dove Random indica un sottoinsieme di dati preso in modo casuale nel dataset S1, la cui dimensione è data da:

  (Splitting Percentage – la dimensione di D1);

Esempio:

Sia S = 100 patterns il dataset originario, SP = 70% la percentuale (Figura 38)

# u-g	g-r	r-i	i-z
1.732	0.954	0.367	0.304
1.478	0.813	0.408	0.331
1.703	0.901	0.370	0.297
2.057	0.983	0.387	0.287
1.746	0.821	0.327	0.273
1.873	0.944	0.364	0.327
1.379	0.881	0.376	0.277
1.057	1.780	0.522	0.387
1.506	0.880	0.394	0.267
1.784	0.897	0.350	0.323

Figura 38 – Esempio:Dataset Originario

Con il metodo descritto otteniamo

S1 = 70 patterns, corrispondenti al 70% di S (Figura 39);

# u-g	g-r	r-i	i-z
1.732	0.954	0.367	0.304
1.478	0.813	0.408	0.331
1.703	0.901	0.370	0.297
2.057	0.983	0.387	0.287
1.746	0.821	0.327	0.273
1.873	0.944	0.364	0.327
1.379	0.881	0.376	0.277

Figura 39 –Esempio: Dataset estratto S1

S2 = 70 patterns, composti da 30 patterns (30% di S non presente in S1) + 40 ottenuti da (SP - 30%) che sono i patterns di overlap, estratti in modo casuale da S1 (Figura 40);

# u-g	g-r	r-i	i-z
1.057	1.780	0.522	0.387
1.506	0.880	0.394	0.267
1.784	0.897	0.350	0.323
1.478	0.813	0.408	0.331
2.057	0.983	0.387	0.287
1.746	0.821	0.327	0.273
1.379	0.881	0.376	0.277

Figura 40 – Esempio:Secondo dataset estratto S2

- i pattern generati casualmente sono salvati in un file ASCII, "*overlap*";

# u-g	g-r	r-i	i-z
1.478	0.813	0.408	0.331
2.057	0.983	0.387	0.287
1.746	0.821	0.327	0.273
1.379	0.881	0.376	0.277

Figura 41 – Esempio: Dataset di overlap inserito nel file "overlap"

- si esegue l'addestramento su S1 e si aggiunge il valore di uscita per ogni corrispondenza nella colonna con etichetta "**netout1**" del file "**overlap**" e il cluster di appartenenza nella colonna con etichetta "**o1c-[#etichetta]**";

# u-g	g-r	r-i	i-z	netout1	cluster1
1.478	0.813	0.408	0.331	0.05	o1c-1
2.057	0.983	0.387	0.287	0.08	o1c-2
1.746	0.821	0.327	0.273	0.03	o1c-3
1.379	0.881	0.376	0.277	0.05	o1c-1

Figura 42 – Esempio: File "overlap" dopo il train su S1

- si esegue l'addestramento su S2 e si aggiunge il valore di uscita per ogni corrispondenza nella colonna con etichetta "**netout1**" del file "**overlap**" e il cluster di appartenenza nella colonna con etichetta "**o1c-[#etichetta]**";

# u-g	g-r	r-i	i-z	netout1	cluster1	netout2	cluster2
1.478	0.813	0.408	0.331	0.05	o1c-1	0.47	o2c-1
2.057	0.983	0.387	0.287	0.08	o1c-2	0.38	o2c-2
1.746	0.821	0.327	0.273	0.03	o1c-3	0.16	o2c-3
1.379	0.881	0.376	0.277	0.05	o1c-1	0.27	o2c-4

Figura 43 – Esempio: File "overlap" dopo il train su S2

- si creano due tabelle temporanee con le seguenti colonne:
  - **LabelCluster**: etichetta progressiva, ad esempio C-1, C-2,...;
  - **NetOutput[#indice del dataset:{1 se S1, 2 se S2}]:** valore di output;
  - **OccurrencesOut[#indice del dataset{1 se S1, 2 se S2}]:** occorrenze di output;

LabelCluster	Net Output 1	OccurrencesOut1
C-1	0.05	2
C-2	0.08	1
C-3	0.03	1

Figura 44 – Esempio:tabella temporanea riguardante il dataset S1

LabelCluster	NetOutput 2	OccurrencesOut2
C-1	0.47	1
C-2	0.38	1
C-3	0.05	1
C-4	0.27	1

Figura 45 – Esempio: tabella temporanea riguardante il dataset S2

La creazione della tabella riguardante il primo dataset (S1) avviene come segue:

- legge il valore della colonna "**netOut1**" del file "**overlap**";
- se il valore letto già esiste nella colonna **NetOutput** del file temporaneo incrementa il valore corrispondente di **OccurrenceOut1**;
- altrimenti, se il valore non esiste inserire nel file temporaneo una nuova riga con i seguenti valori:
  - nuova etichetta: **C-[#n]**, dove *n* è un numero che cresce progressivamente;
  - inserire il nuovo valore nella colonna **NetOutput1**;
  - aggiornare il contatore **OccurrenceOut1** in corrispondenza con il valore 1.

La stessa procedura è effettuata per il secondo file temporaneo considerando il dataset S2 , la colonna "**netOut2**" del file "**overlap**";

- calcolare la somma per ogni corrispondenza come percentuale sul numero dei pattern del file "**overlap**";
- creare un nuovo file "**stat.log**" e inserire nella prima riga la percentuale;
- inserire in "**stat.log**" i dati presenti in "**overlap**";
- creare una tabella con cinque colonne all'interno del file "**stat.log**";
  - prima colonna, "**labelCluster**"
  - seconda colonna "**OccOut1**" dal primo file temporaneo;
  - terza colonna "**OccOut2**" dal secondo file temporaneo;

- quarta colonna " **Percentage%"** valore calcolato paragonando **OccOut1** e **OccOut2** come segue**:**

{[min (OccOut1,OccOut2)] \ [max(OccOut1,OccOut2)] } * 100

Total percentage: 50%

```
# u-g          g-r          r-i          i-z          netout1  cluster1     netout2  cluster2
  1.478        0.813        0.408        0.331        0.05     o1c-1        0.47     o2c-1
  2.057        0.983        0.387        0.287        0.08     o1c-2        0.38     o2c-2
  1.746        0.821        0.327        0.273        0.03     o1c-3        0.16     o2c-3
  1.379        0.881        0.376        0.277        0.05     o1c-1        0.27     o2c-4
```

LabelCluster	OccurrencesOut1	OccurrencesOut2	Percentage(%)
C1	2	1	50
C2	1	1	100
C3	1	1	100
C4	0	1	50
...	..	...	...
...	...	...	...
...	...	...	...

Figura 46 – Esempio: il file "stat.log" al termine del caso d'uso test train

3.2.3.2 Il modello CSOM

Il modello CSOM rappresenta una specializzazione del modello GSOM e si occupa dell'implementazione dei casi d'uso visti nel modello GSOM utilizzando, come dataset, solo immagini di tipo Fits date in input dall'utente.

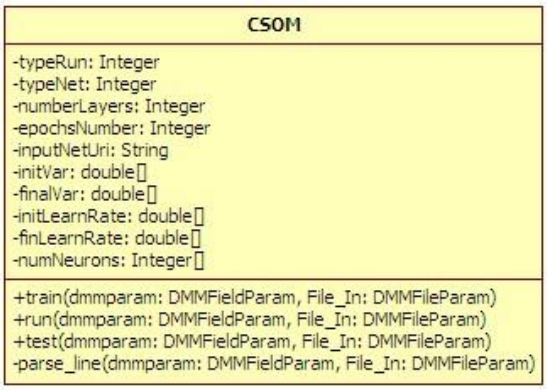

Figura 47 – La classe CSOM

Questa classe implementa , in modo differente dalla classe GSOM, il metodo "***parse_line***" per la definizione dei parametri da passare ai vari casi d'uso e, come la classe GSOM, utilizza la classe "Runtime" per eseguire i metodi della libreria NExTII.

3.3 Multi Layer Clustering Plugin (MLC Plugin)

Il DMPlugin è stato creato per dare la possibilità all'utente di utilizzare i modelli di rete implementati all'interno della suite (vedi Capitolo 2). Per questo motivo, una volta inserite tutte le modifiche e le nuove classi, è stato necessario implementare i relativi plugin. In questo caso, il lavoro da me svolto ha incluso l'implementazione di due plugin: CSOM_Clustering per il modello di clustering specifico CSOM e il plugin GSOM_Clustering per il modello più generico.

3.3.1 Introduzione

Per l'implementazione dei plugin, la suite mette a disposizione degli sviluppatori un'applicazione, il DMPluginWizard, sviluppata all'interno del Progetto DAME dal collega Fiore Michelangelo. Esso è stato creato per permettere a sviluppatori, anche esterni al gruppo di lavoro del progetto DAME, di implementare plugin per i modelli di reti neurali presenti senza avere una conoscenza pregressa del codice dell'intera suite.

Il DMPluginWizard permette però la sola implementazione automatica del metodo costruttore del plugin che, come vedremo, grazie all'interfaccia grafica di cui è dotata, risulta molto semplice e veloce.

3.3.2 Lo sviluppo del costruttore

Per semplicità, sarà introdotto il procedimento di creazione del costruttore di uno solo dei due plugin poiché esso risulta essere identico per entrambi.

Consideriamo l'interfaccia grafica nella figura che segue:

Figura 48 – Interfaccia grafica del DMPluginWizard

Per sviluppare il costruttore è necessario riempire tutti i campi. Prima di tutto bisogna inserire tutte le informazioni di base per la configurazione:

- **Name:** è il nome del plugin da aggiungere alla suite; esso deve avere lo stesso nome della classe che implementa il plugin;
- **Documentation**: Url della documentazione del plugin;
- **Version**: versione del plugin;
- **Domains**: indica quale funzionalità fornire al plugin;
- **Owner Name:** nome del proprietario;
- **Owner Mail:** e-mail del proprietario.

Il passo successivo è il riempimento dei campi nel riquadro "***Running Modes Information***" dove è indicato quale caso d'uso (train, test, run, full) implementerà il plugin.

Al termine di queste operazioni si avrà la seguente situazione:

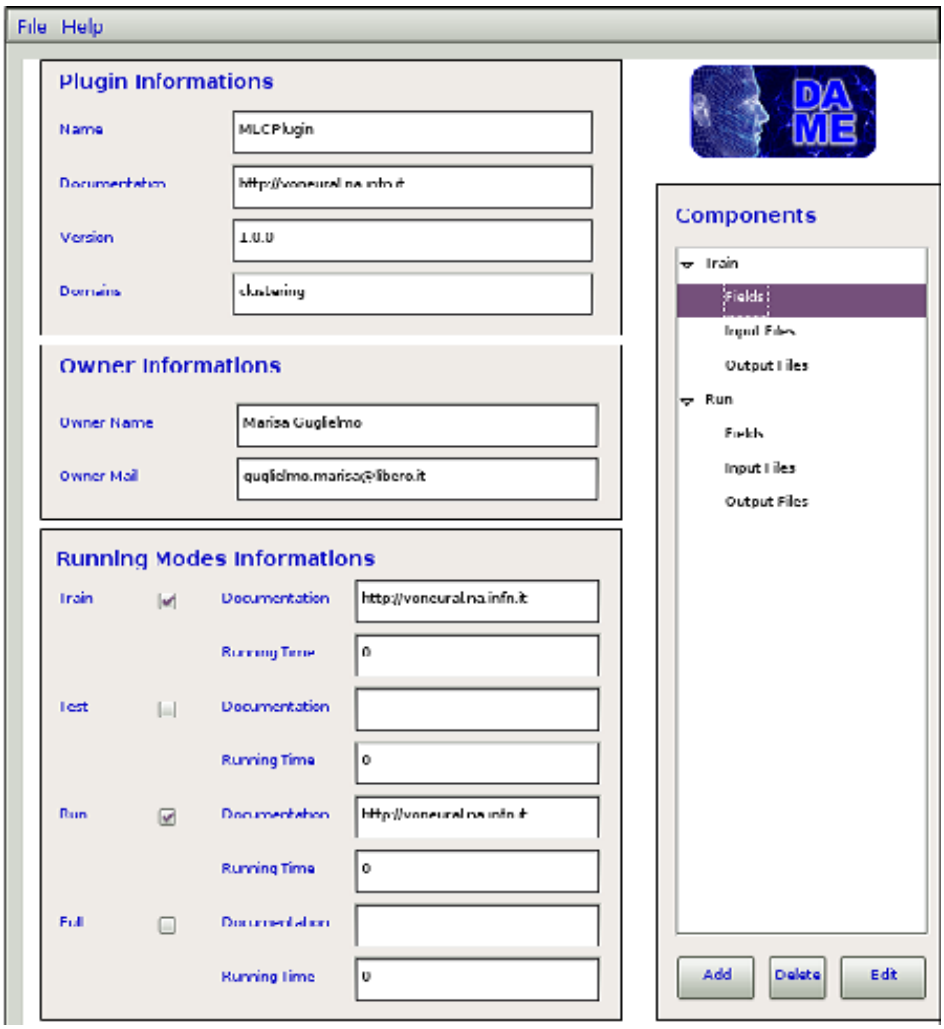

Figura 49 – Esempio di riempimento dei campi della GUI del DMPlugin

Nel riquadro ”***Components***”, a destra dell’interfaccia, sono comparse le icone relative ad ogni caso d’uso e riguardano la definizione di tutti i parametri e i file di I/O, necessari per l’esecuzione del caso d’uso stesso. Selezionando la voce “**Fields**” o la voce “**Input Files**” e selezionando “**Add**” è possibile definire tutti parametri del modello scelto.

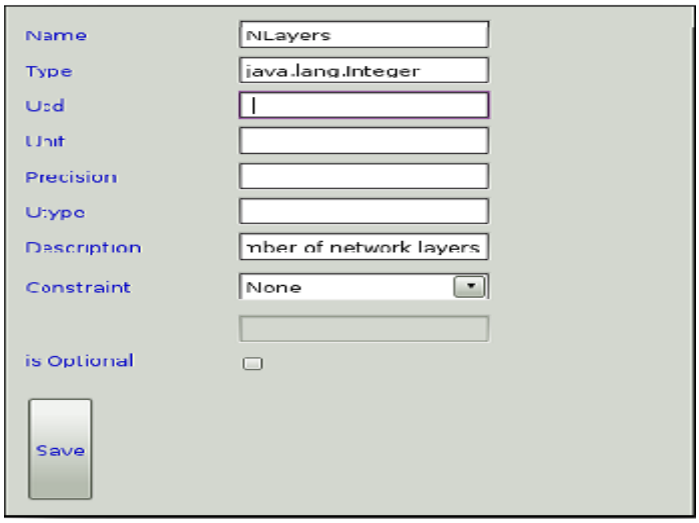

Figura 50 – Finestra della voce “Fields”

Nella figura in alto è rappresentata la finestra relativa all’aggiunta di un parametro; essa si ottiene selezionando la voce “**Fields**” e presenta alcuni parametri importanti:

- **Name**: è l’etichetta del parametro, attraverso la quale il parametro si rende riconoscibile alla classe CSOM: più precisamente si analizza la prima lettera dell’etichetta resa univoca per ogni campo;
- **Type:** tipo del parametro;
- **Ucd, Unit, Precision, Utype, Description:** campi relativi al parametro;

Lo stesso avviene selezionando la voce “**Input Files**” che apre la finestra rappresentata nella figura che segue:

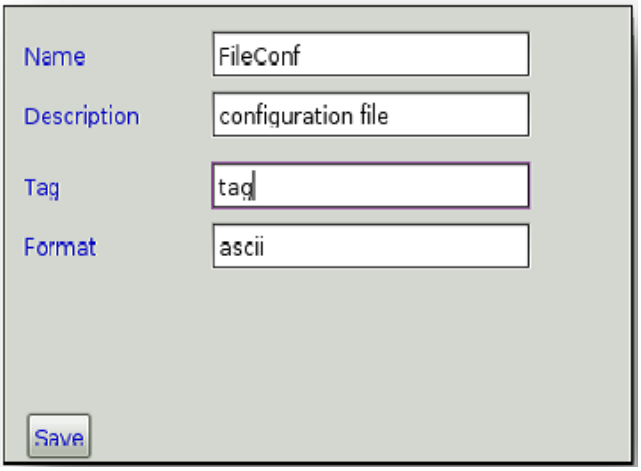

Figura 51 – Finestra della voce “Input Files”

Dopo aver riempito tutti i campi necessari, si seleziona nel menu in alto, sotto la voce “**File**”, il comando “***Generate Code***”, che genera il codice in linguaggio Java del costruttore del plugin.

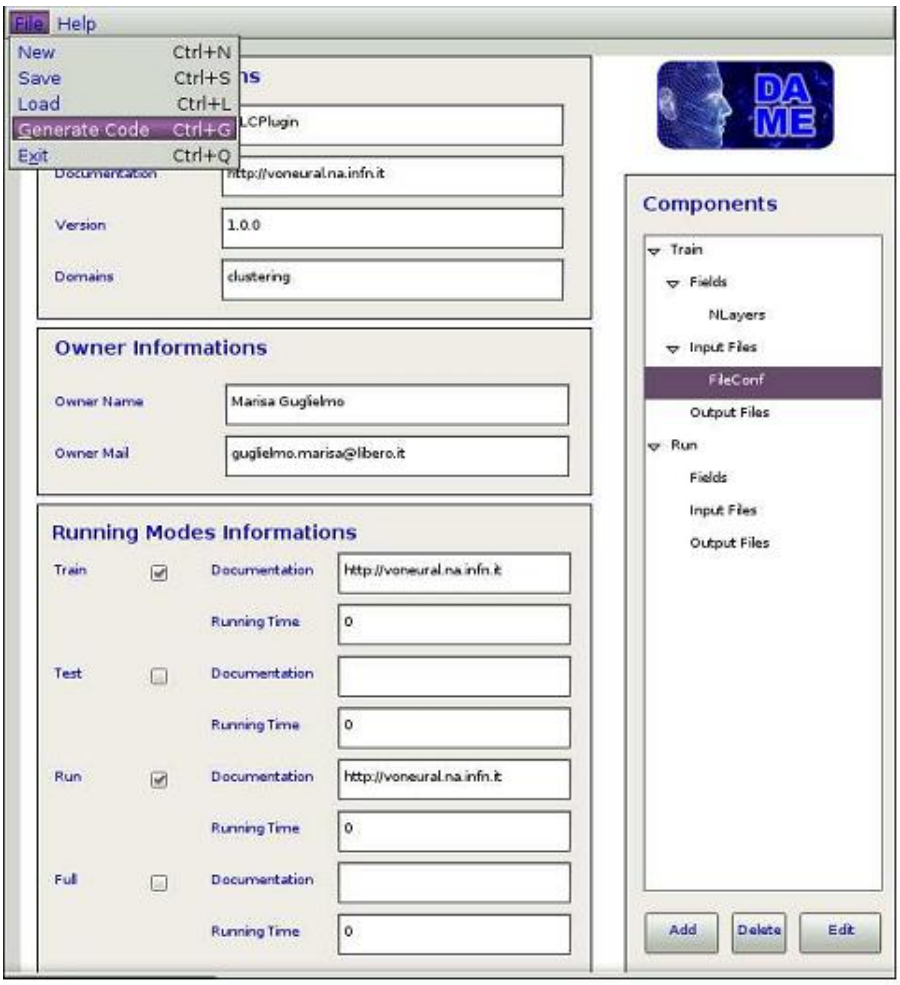

Figura 52 – Esempio di generazione codice del plugin

Una volta eseguito il comando "***Generate Code***", l'applicazione crea una classe Java con lo stesso nome del plugin, all'interno del componente DMPlugin, in cui è stato automaticamente inserito il codice relativo al costruttore della classe. Inoltre, si creano alcuni metodi da sovrascrivere per l'utilizzo dello stesso. Il numero di questi metodi varia a seconda del modello e della funzionalità scelta; complessivamente sono quattro: *trainRun*, *testRun*, *runRun* e *fullRun*. In questo caso, essi creano oggetti della classe Clustering. Durante la creazione del codice, inoltre, sono generati automaticamente anche i due vettori, DMMFieldParam e DMMFileParam, da dare in input ai metodi implementati dalla classe Clustering.

Capitolo 4. Il Testing

In questo capitolo è descritta la fase di Testing di tutte le classi e le funzioni sviluppate, sia in linguaggio JAVA per l'estensione del DMM, sia in linguaggio C++ per l'estensione della libreria NExTII.

4.1 Testing dei componenti

Il Testing del componente DMM, dopo l'integrazione delle classi Unsupervised, Clustering, CSOM e GSOM, è stato concentrato sulla verifica della corretta disponibilità per l'utente della funzionalità di Clustering.

Come già visto nel capitolo precedente, la classe Clustering contiene i tre metodi: *train*, *test* e *run*.

Gli input di questi metodi sono:

- Parametro per indicare il modello (0=CSOM, 1=GSOM);
- Valori dei parametri del modello contenuti nel vettore *dmmparam*;
- File di Input/Output contenuti nel vettore ***File_in*** di tipo DMMFileParam.

Il Testing sull'estensione e sul controllo di NExTII si è concentrato sulla verifica dei parametri inseriti. E' stata inoltre verificata la scelta corretta dei metodi da utilizzare in base ai file di input inseriti, verificando diverse combinazioni di parametri.

4.2 Caratteristiche del testing

Le caratteristiche principali da verificare sono le seguenti:

- Correttezza dei risultati e dei file di output;
- Gestione degli errori, fondamentale per comunicare al FrameWork il tipo di errore (percorso o file inesistente, parametri non corretti, ecc...).

4.3 Classi da verificare durante il testing

Le classi e i relativi metodi da verificare sono:

- Clustering e relativi metodi;
- CSOM e relativi metodi;
- GSOM e relativi metodi;
- NextControl e relativi metodi;

Inoltre è necessario verificare la gestione della libreria NExTII, ovvero "NextMain", e verificare tutti i casi possibili di combinazioni **[tipo modello, caso d'uso, tipo di file input]** nonché il numero e la correttezza dei parametri ricevuti in input dal DMM.

4.3.1 Test Cases

Di seguito sono riportati alcuni casi di test più significativi. Per ogni test case eseguito, come ulteriore file di ouput, si ha il file "StatusExperiment.log"; esso riporta tutti i parametri inseriti per l'esecuzione dell'esperimento; inoltre, in caso di esperimento non riuscito esso riporta il tipo di errore riscontrato.

TEST ID	1	
TEST NAME	testTrainDat1	
TEST DESCRIPTION	Esecuzione del metodo train su un file di tipo "dat" con un solo livello di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCSOMTrain.log	MLCSOMTrain.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	2	
TEST NAME	testRunDat1	
TEST DESCRIPTION	Esecuzione del metodo run su un file di tipo “dat” con un solo livello di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCCluster.log	MLCCluster.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	3	
TEST NAME	testDat1	
TEST DESCRIPTION	Esecuzione del metodo advanced train su un file di tipo “dat” con un solo livello di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCSOMDAT1.log MLCSOMDAT1.log overlap.log	MLCSOMDAT1.log MLCSOMDAT1.log overlap.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	4	
TEST NAME	testTrainFits1	
TEST DESCRIPTION	Esecuzione del metodo train su un file di tipo “fits” con un solo livello di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCSOMTrain.log	MLCSOMTrain.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	5	
TEST NAME	testRunFits1	
TEST DESCRIPTION	Esecuzione del metodo run su un file di tipo “fits” con un solo livello di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCCluster.log	MLCCluster.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	6	
TEST NAME	testFits1	
TEST DESCRIPTION	Esecuzione del metodo test su un file di tipo “fits” con un solo livello di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCSOMDAT1.log MLCSOMDAT1.log overlap.log	MLCSOMDAT1.log MLCSOMDAT1.log overlap.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	7	
TEST NAME	testTrainDat2	
TEST DESCRIPTION	Esecuzione del metodo train su un file di tipo “dat” con due livelli di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCSOMTrain.log	MLCSOMTrain.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	8	
TEST NAME	testRunDat2	
TEST DESCRIPTION	Esecuzione del metodo train su un file di tipo “dat” con due livelli di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCCluster.log	MLCCluster.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	9	
TEST NAME	testDat2	
TEST DESCRIPTION	Esecuzione del metodo test su un file di tipo “dat” con due livelli di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCSOMDAT1.log MLCSOMDAT1.log overlap.log, stat.log	MLCSOMDAT1.log MLCSOMDAT1.log overlap.log, stat.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	10	
TEST NAME	testTrainFits2	
TEST DESCRIPTION	Esecuzione del metodo train su un file di tipo “fits” con due livelli di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCSOMTrain.log	MLCSOMTrain.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	11	
TEST NAME	testRunFits2	
TEST DESCRIPTION	Esecuzione del metodo run su un file di tipo “fits” con due livelli di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCCluster.log	MLCCluster.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	12	
TEST NAME	testFits2	
TEST DESCRIPTION	Esecuzione del metodo test su un file di tipo “fits” con due livelli di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCSOMDAT1.log MLCSOMDAT1.log overlap.log , stat.log	MLCSOMDAT1.log MLCSOMDAT1.log overlap.log , stat.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	13	
TEST NAME	testTrainDat3	
TEST DESCRIPTION	Esecuzione del metodo train su un file di tipo “dat” con tre livelli di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCSOMTrain.log	MLCSOMTrain.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	14	
TEST NAME	testRunDat3	
TEST DESCRIPTION	Esecuzione del metodo train su un file di tipo “dat” con tre livelli di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCCluster.log	MLCCluster.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	15	
TEST NAME	testDat3	
TEST DESCRIPTION	Esecuzione del metodo test su un file di tipo “dat” con tre livelli di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCSOMDAT1.log MLCSOMDAT1.log overlap.log , stat.log	MLCSOMDAT1.log MLCSOMDAT1.log overlap.log , stat.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	16	
TEST NAME	testTrainFits3	
TEST DESCRIPTION	Esecuzione del metodo train su un file di tipo “fits” con tre livelli di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCSOMTrain.log	MLCSOMTrain.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	17	
TEST NAME	testRunFits3	
TEST DESCRIPTION	Esecuzione del metodo run su un file di tipo “fits” con tre livelli di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCCluster.log	MLCCluster.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

TEST ID	18	
TEST NAME	testFits3	
TEST DESCRIPTION	Esecuzione del metodo test su un file di tipo “fits” con tre livelli di rete SOM	
INPUT	**OUTPUT**	**EXPECTED RESULT**
Tutti i parametri	MLCSOMDAT1.log MLCSOMDAT1.log overlap.log , stat.log	MLCSOMDAT1.log MLCSOMDAT1.log overlap.log , stat.log
Parametri Scorretti	Error	Error
Nessun parametro	Error	Error

Conclusioni

Lo scenario scientifico del presente lavoro si colloca all'interno dell'attuale problematica legata all'esigenza di conciliare la produzione massiva di dati, relativi a varie osservazioni di fenomeni naturali, con la loro memorizzazione ed elaborazione. L'enorme mole di informazioni da un lato apre nuove prospettive di indagine scientifica, ma dall'altro mette a dura prova le capacità degli scienziati ed analisti di estrarre informazioni utili dagli enormi database disponibili. Il presente lavoro riguarda un'estensione di una suite di data mining (progetto DAME), preposta esattamente a facilitare il problema su citato, focalizzando l'attenzione al caso astrofisico ed astro-particellare.

Per data mining in questo caso ci si riferisce all'utilizzo di modelli e algoritmi derivati dal paradigma del machine learning, basati cioè sul meccanismo dell'apprendimento automatico, per l'esplorazione ed analisi di dati, quasi sempre caratterizzati da un basso rapporto segnale/rumore. In tale contesto, il presente lavoro riguarda la progettazione ed implementazione del modello *"Self-Organizing Map*" (SOM), afferente alla categoria dell'apprendimento non supervisionato e orientato a trattare problematiche legate alla segmentazione e clustering di immagini astronomiche multi-banda (sia sintetiche che osservative). Inoltre, il lavoro comprende anche lo sviluppo di un'architettura gerarchica a più livelli, destinata al raffinamento delle tecniche di clustering, definita come clustering multi-strato, all'interno della quale sia possibile utilizzare per ogni strato, sia il modello SOM, sia un generico modello di clustering non supervisionato. Un aspetto non secondario del presente lavoro è l'integrazione di tali sistemi nella Suite DAME, infrastruttura già sviluppata e quindi già strutturata secondo determinati standard di rappresentazione dell'informazione e di flusso di lavoro. Ciò unitamente al delicato lavoro di reverse engineering della libreria software NExTII, parzialmente riutilizzata nel nostro lavoro, relativa ad una pipeline di estrazione di cataloghi da immagini astronomiche sviluppata dal team del progetto DAME.

Dal punto di vista prettamente tecnologico, gli obiettivi descritti sono stati raggiunti tramite un'attenta analisi sia delle tecnologie a disposizione sia della struttura della suite, permettendo di adottare una soluzione che non ne modificasse la struttura interna.

Il lavoro svolto rappresenta il primo passo verso l'integrazione all'interno della suite di tutti modelli di rete non supervisionati presenti in NExTII che permetterà agli utenti della stessa di eseguire esperimenti di segmentazione automatica e semi-automatica delle immagini astronomiche.

Riferimenti Bibliografici

1. **Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin.** A Practical Guide to Support Vector Classification. 2007.

2. **Vellucci, Civita.** NEXt (Neural Extractor) II: un segmentatore di immagini astronomiche. *Tesi di Laurea Sperimentale in Informatica,Università Degli Studi di Napoli "Federico II",*. s.l. : Relatori: F. Isgrò, G. Longo, 2008/2009.

3. **Guglielmo, Marisa, Brescia, Massimo e Vellucci, Civita.** Multi Layer Clustering & Self Organizing Map Neural Network. *MLCSOM_DAME-SPE-NA-0010-REL.1.0.* 2010.

4. **Guglielmo, Marisa.** SOMEDAME (Self Organizing Map Experiment in DAME). *SOM&MLC_DAME-NA-PRE-0030_Rel.1.0.* 2010.

5. **Progetto S.Co.P.E.** http://www.scope.unina.it.

6. **Progetto Vo-Neural\DAME.** http://www.voenural.na.infn.it.

7. **IVOA.** http://www.ivoa.net.

8. **European Southern Observatory (ESO).** http://www.eso.org.

9. **Guglielmo, Marisa.** Multi Layer Clustering & SOM Models, Data Mining Model Software Requirements Specification. *MLCSOM_VONEURAL-SRS-NA-0008-Rel.1.0.* 2010.

10. **Cavuoti, Stefano.** Ricerca di Nuclei Galattici Attivi in survey fotometriche multibanda. *Tesi di Laurea in Fisica, Università Degli Studi di Napoli "Federico II".* s.l. : Relatori: G.Longo, M.Paolillo, 2006/2007.