Università degli studi di Napoli

Federico II

Facoltà di Scienze MM. FF. NN.

Corso di Laurea in Informatica

Tesi di Laurea Sperimentale

Il componente Data Mining Model del progetto DAME

Tutor Accademico: Dott.ssa Anna Corazza

Tutor Aziendale: Dott. Massimo Brescia

Candidato: Alessandro Di Guido

Matricola: 566/2288

Anno Accademico 2008-2009

Sommario

Indice delle figure

1. Introduzione

In questo capitolo verrà introdotto il problema propostomi durante la mia attività di tirocinio, svolta presso l'Osservatorio Astronomico di Capodimonte. Tale problema riguarda la progettazione e l'implementazione di una componente relativa al Progetto DAME (Data Mining & Exploaration).

Il Progetto DAME nasce con lo scopo di fornire una suite di data mining orientata al web, che permetterà ai suoi utilizzatori di effettuare esperimenti di data mining astronomico con un alto grado di complessità.

Per tale motivo, la suite DAME utilizzerà una potente infrastruttura computazionale, realizzata dal progetto SCoPE (Sistema Cooperativo per Esperimenti scientifici ad alte prestazioni), basata su paradigma Grid e sulle più moderne tecnologie per il calcolo distribuito, messa a disposizione dall' Università di Napoli Federico II per la Ricerca di base. Il Progetto SCoPE infatti prevede lo sviluppo di codici innovativi per applicazioni in diversi settori della ricerca scientifica di base tra i quali:

- Astrofisica (dove appunto si colloca il Progetto DAME)
- Fisica Subnucleare
- Matematica Numerica
- Scienze della vita
- Informatica
- Elettromagnetismo e telecomunicazioni

L'infrastruttura di SCoPE consiste nella creazione di una Grid Metropolitana capace di unire dipartimenti e struttura di ricerca afferenti o in collaborazione con la Federico II, situate in varie zone della città di Napoli[1].

Partnership:

- Dipartimento di Fisica (sezione Astrofisica) – Università degli studi di Napoli Federico II
- INAF – Osservatorio Astronomico di Capodimonte
- California Institute of Technology, Pasadena – USA

Collaborazioni:

- VOTECH (Virtual Observatory Technological Infrastructures)
- S.Co.P.E.
- INAF - Osservatorio Astronomico di Trieste

[1] Vedere anche www.scope.unina.it

- MIUR (Ministero dell'Istruzione, dell'Università e della Ricerca)
- EURO-VO (laboratorio di osservazione virtuale europeo)

1.1. Problema proposto

Il Problema a me proposto dal mio Tutor Aziendale (Dr. Massimo Brescia) è stato quello di progettare e implementare in linguaggio Java una componente della suite DAME. Tale componente denominata DMM (Data Mining Models) ha il compito principale di effettuare data mining sui dati che l'utente dell'applicazione web darà in input, utilizzando opportunamente le librerie a disposizione, e di dare in output i risultati da esse ottenuti; queste librerie implementano 3 dei principali modelli di apprendimento auto adattivo:

- SVM (Support Vector Machine)
- MLP(Multi-Layer Perceptron)
- MLPGA (Multi-Layer Perceptron with Genetic Algorithm)

Nei prossimi capitoli verrà descritto dettagliatamente il Progetto DAME, ed il lavoro da me svolto per la progettazione e l'implementazione della componente DMM.

2. Progetto DAME (DAta Mining & Exploration)

In questo capitolo si descriverà il Progetto DAME, discutendo sui motivi della sua nascita e la struttura della suite integrata al progetto. Verranno descritti anche alcuni aspetti teorici strettamente correlati al progetto, fondamentali per il mio lavoro all'interno del progetto.

DAME è un progetto volto a sviluppare strumenti per il data mining scientifico[2], basati sulla tecnologia dell'informazione e della comunicazione. I dati scientifici, di natura astrofisica e caratterizzati da enormi datasets, sono raccolti attraverso l'utilizzo di diverse tecniche, e sono memorizzati in diversificati e spesso incompatibili archivi di dati.

Il Progetto DAME mira a creare una singola "e-infrastuctrure" distribuita per il data mining, nonché per l'esplorazione e la visualizzazione di grosse moli di dati o MDS (Massive Data Sets). Essa deve fornire un accesso integrato ai dati raccolti da strumenti, esperimenti, e comunità scientifiche molto diversi fra loro, al fine di essere in grado di correlare e migliorare la loro utilità scientifica e l'interoperabilità.

[2] Vedi anche www.voneural.na.infn.it

Il Progetto consiste nella progettazione e nello sviluppo di una "*Data Mining Suite*" che fornirà alle comunità astronomiche, e non solo, con potenti strumenti software, la possibilità di lavorare su enormi dataset in un ambiente di calcolo distribuito rispettando gli standard internazionali e i requisiti IVOA (International Virtual Observatory Alliance, www.ivoa.net).

2.1. Data Mining

Il progetto DAME, nasce dall'esigenza nell'Astrofisica di fare Data Mining.
In generale per Data Mining (letteralmente: *estrazione di dati*) si intende estrarre conoscenza, non nota a priori, a partire da grandi quantità di dati intrinsecamente dotati di notevoli quantità di noise.
Oggi il data mining ha una duplice valenza:

- **Estrazione**, cioè estrarre informazione implicita o nascosta da dati già strutturati, con tecniche analitiche all'avanguardia, per renderla disponibile e direttamente utilizzabile
- **Esplorazione ed analisi**, eseguita in modo automatico o semiautomatico, su grandi quantità di dati allo scopo di scoprire pattern significativi.

Nella ricerca scientifica molti fattori hanno contribuito allo sviluppo del data mining come le grandi accumulazioni di dati in formato elettronico, il data storage poco costoso e l'introduzione di nuovi metodi e tecniche di analisi come apprendimento automatico e riconoscimento di pattern.
Le tecniche di data mining sono fondate su specifici algoritmi e hanno lo scopo principale di trovare nuovi pattern, che a loro volta, potrebbero essere il punto di partenza per la scoperta di nuove relazioni di tipo causale tra fenomeni, e quindi, effettuare delle previsioni su nuovi insiemi di dati.
Una tecnica molto diffusa per il data mining è **l'apprendimento mediante classificazione**, che consiste nel classificare esempi non noti, partendo da un insieme di esempi di classificazione noti o base di conoscenza (Capitolo 2, Paragrafo 2.1.2.1). Tale approccio viene anche detto supervisionato (supervised), nel senso che lo schema di apprendimento è "guidato" dai dati forniti in input di cui si conosce l'output, ovverossia controllata dagli esempi di classificazione forniti per i casi noti.
Altra tecnica supervisionata è **la regressione** (Capitolo 2, Paragrafo 2.1.2.2), che consiste nell'identificare una funzione che meglio approssima la distribuzione correlata di dati fornita in input, senza conoscerne a priori la forma analitica.
Viceversa i metodi non supervisionati sono metodi per la scoperta di pattern che utilizzano i dati stessi (vedi paragrafo 2.4.2).

La figura sottostante mostra i vari livelli di conoscenza che un astrofisico deve conoscere per fare data mining.

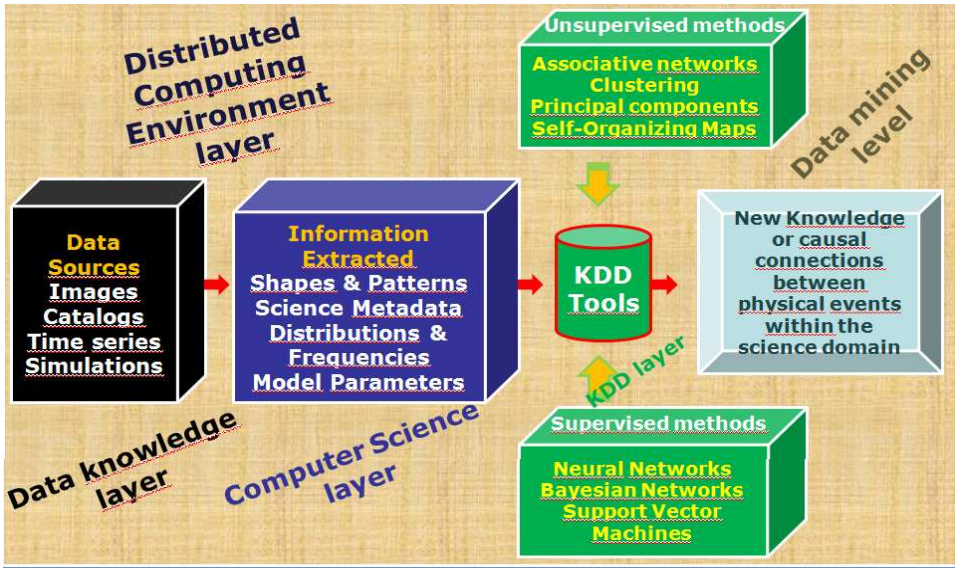


Fig. 1 I Livelli di conoscenza nell'Astroinformatica

Egli infatti deve avere contemporaneamente conoscenze informatiche (anche abbastanza avanzate) e fisiche per arrivare a poter fare data mining. In altre parole, l'astrofisico moderno deve approcciare il problema del data mining in senso multidisciplinare, assumendo il ruolo ibrido che va sotto il nome di "astroinformatico".

2.2. Descrizione Progetto DAME

In Astrofisica, come in molti campi in campi dell'attività umana, negli ultimi dieci anni la ricerca basata sul data mining è stata guidata dal progresso tecnologico dei database è anche dalla necessità di dover operare su dataset di enormi dimensioni.

Nonostante questi progressi, rimane di assoluta rilevanza il problema di trovare metodi adeguati per l'esplorazione e l'analisi di grandi insiemi di dati astronomici.

L'accento è posto su metodi che colmino il divario tra accurate rappresentazione ed estrazioni di dati, e anche il divario tra le potenzialità della tecnologia attuale e gli utenti.

I metodi di analisi statistica basata parzialmente sulle scelte casuali (algoritmi genetici) e sulle esperienze delle conoscenze acquisite (apprendimento adattivo supervisionato/non supervisionato) potrebbero realizzare la scoperta di leggi nascoste in fenomeni particolari, basate su leggi della natura già note, ma di cui non era chiara l'applicazione.

DAME è una Data Mining suite adatta a lavorare su dataset di grandi dimensioni. Inoltre è una suite web-oriented che rispetta gli standard di rappresentazione dei dati della comunità internazionale VO (Virtual Observatory).

Al fine di effettuare esperimenti di data mining e di esplorazione di dati, la suite DAME è stata progettata con una strategia top-down, a partire dalla tassonomia delle varie strategie di data mining e di ricerca dei dati, che sono associate a specifici algoritmi e metodi di lavorazione.
Nella prima versione, la suite offre strumenti per le seguenti funzionalità:

- Classificazione
- Regressione

Le tecnologie che caratterizzano l'intera suite possono essere riassunte come segue:

- OOP(Object Oriented Programming) & UML(Unified Modeling Language)
- Standard e protocolli interni basati su XML
- Architettura Java Web Application
- Database utilizzato per dati di I/O degli esperimenti, registrazione e autenticazione degli utenti
- Web-based user I/O (GWT)
- Interfaccia utente basata su servizi web e applicazioni web (web-tools)

Caratteristiche principali:

- Espandibile tramite integrazione o modifica di plugin di modelli(librerie di nuovi modelli) non ancora supportati dalla suite
- Indipendente dall'Hardware attraverso la piattaforma "driver" che permette alla suite di essere utilizzabile sia in modalità "stand alone" che in una infrastruttura di computazione distribuita
- Sessione di lavoro interattiva e/o asincrona durante l'intera fase del lancio dell'esperimento. Nel primo caso l'utente vuole monitorare/valutare/terminare l'esperimento in fase di lancio,in modo tale da modificare alcuni parametri e rilanciarlo. Nel secondo caso per "sessione asincrona" si intende che l'utente non vuole tenere traccia dell'esperimento in fase di lancio, chiudendo la connessione dopo aver inserito tutti i parametri e aspettando i risultati (attraverso e-mail per esempio).

2.2.1. Fuzionalità della Suite

Nella prima versione della suite, le due funzionalità previste saranno generalizzate il più possibile, la cui specializzazione è lasciata all'utente finale nella fase di definizione dei parametri. Infatti l'utente può scegliere il grado di specializzazione personalizzando i

valori dei parametri; se non si imposta nessun parametro, l'applicazione assegnerà a tutti i parametri il loro valore di default che varia a seconda del modello scelto; se si impostano i valori di tutti i parametri, la funzionalità richiesta (classificazione, regressione) sarà specializzata al caso d'uso dell'utente. Ovviamente tra questi due esempi c'è un elevato grado di libertà.
Di conseguenza il Front-End (Interfaccia Utente della Suite) deve dare la possibilità di configurare tutti i parametri del modello di data mining.
In questo scenario, tale requisito implica una corrispondenza uno a uno tra funzionalità e modelli di data-mining implementati dalla suite.
È stato previsto che nelle future evoluzioni della suite DAME saranno implementate delle funzionalità più specializzate, le quali non richiederanno una modifica strutturale di alcuni componenti (componenti Framework e Front-end), ma semplicemente richiederanno un arricchimento delle strutture e interfacce esistenti.
Nei prossimi sottoparagrafi saranno descritte le già citate funzionalità della suite DAME.

2.2.1.1. Classificazione

La funzione di Classificazione consiste nel raggruppare elementi singoli in base a informazioni su una o più caratteristiche interne e attraverso una procedura con supervisione (apprendimento con dati etichettati, cioè di cui si conosce l'etichetta);
Un classificatore compie una mappatura da uno spazio di parametri **X** ed un insieme di etichette **Y** (assegna una etichetta predefinita per ogni campione);
Formalmente:

classificatore h: $\mathbf{X} \rightarrow \mathbf{Y}$ associa $x \in \mathbf{X}$ ad una label $y \in \mathbf{Y}$

in cui x è un vettore ed y è un insieme finito di etichette.

Ci sono due formati diversi di output che si possono ottenere facendo classificazione:

- ***Crispy:*** dato un pattern input **x** (vettore), restituisce la sua label **y** (scalare);
- ***Probabilistic :*** dato un pattern **x** (vettore), restituisce un vettore **y** contenente le probabilità di appartenenza di **x** agli elementi della classe **y;**

Entrambi i casi si possono applicare ai casi di classificazione "2-class" (caso in cui il numero di elementi dell'insieme delle label è uguale a 2) e "multi-class".
la classificazione è basata su almeno tre step:

- **Training,** che consiste "nell'addestrare" il classificatore inserendo coppie di input-output note a priori;

- **Testing,** mediante un test set di dati di input, il cui output è una statistica relativa al grado di appartenenza a diverse classi (matrice di confusione, probabilità etc...);
- **Evaluation,** mediante un dataset non ancora etichettato, cioè altre coppie di input-output, note a priori e non utilizzate nella fase di **training,** il cui output è l'etichettatura rispetto alle classi predefinite;

La classificazione ha chiaramente una natura con supervisione.

2.2.1.2. Regressione

Consiste nella ricerca supervisionata di un'associazione da un dominio $\mathbf{R^n}$ ad un dominio $\mathbf{R^m}$ con n>m, di cui distinguiamo due tipi:

- **Curve Fitting**: tenta di validare un'ipotesi che la distribuzione di dati segua una certa funzione; in altre parole date le coppie di vettori (x,y) e la forma funzionale che si vuole associare, il sistema trova i migliori parametri che identificano l'associazione ipotizzata;
- **Function approximation**: date le coppie di vettori (x, y), il sistema trova il modello che meglio identifica la correlazione tra i dati;

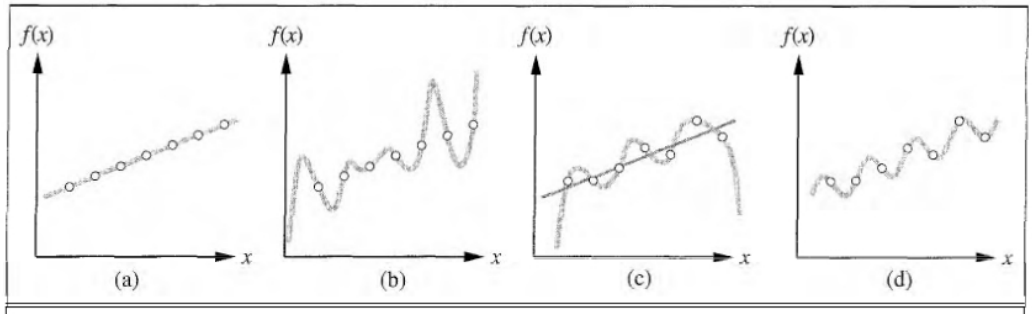


Fig. 2 Fit dei parametri di un modello partendo dai dati

Come si può vedere in Fig.2 la regressione consiste nel trovare una funzione che meglio approssima una distribuzione di dati in input. Nel caso in cui si voglia effettuare una regressione mediante il metodo **Curve Fitting** l'algoritmo prende in input delle coppie di vettori (x,y) e una funzione, cercando di trovare in ogni parametro il valore che meglio si adatta alla funzione presa in input.
Nel caso del metodo **Function approximation** l'algoritmo prende in input solo le coppie di vettori (x,y) cercando una funzione che meglio approssima la distribuzione di dati derivata dalle coppie di vettori.

2.3. Descrizione dei Componenti della Suite DAME

Il Progetto è basato su 5 componenti principali:

- Front End (FE)
- Framework(FW)
- Registry&DB(REDB)
- Driver(DR)
- Data Mining Models (DMM)

Il seguente schema mostra il diagramma della componenti dell'intera suite DAME .

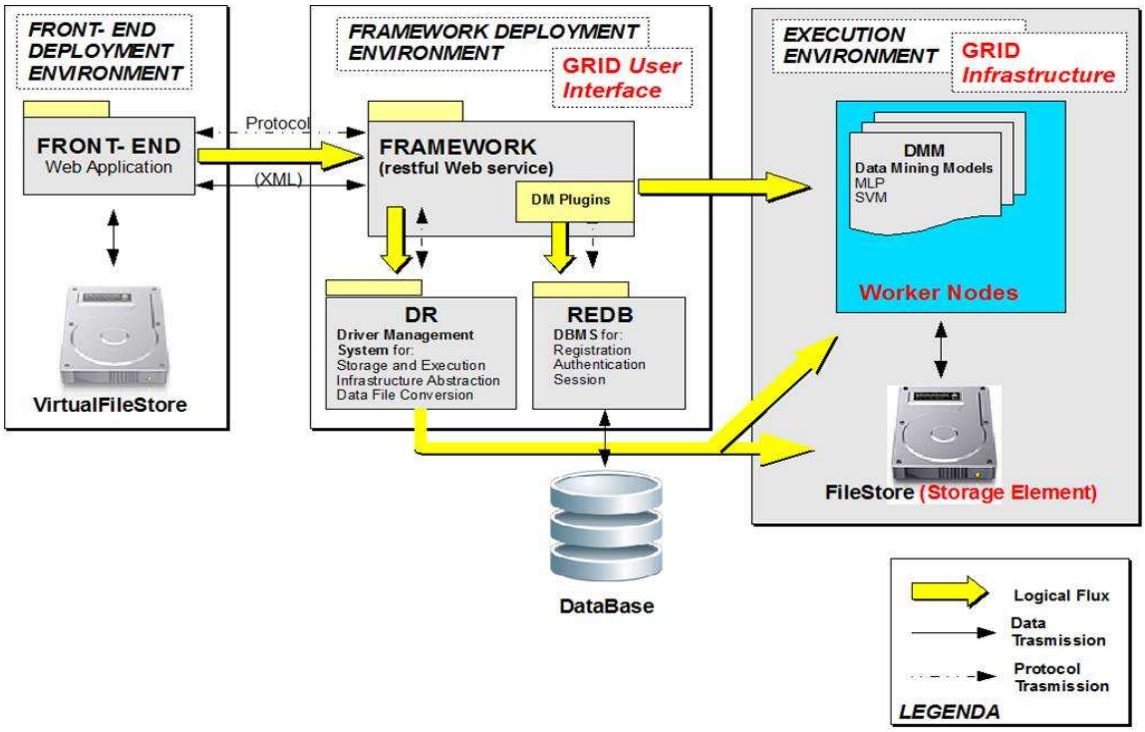


Fig. 3 Diagramma Suite DAME

2.3.1. Front-End

Il componente Front-End, realizzato dal collega Francesco Manna, è la GUI (Graphical User Interface) principale della Suite. Esso contiene le pagine WEB dinamiche che verranno utilizzate dagli utenti finali per interagire con l'applicazione, utilizzare i modelli, e facilitare esperimenti scientifici.

Ogni utente, prima di poter utilizzare le funzionalità della suite, deve effettuare una fase di registrazione, creandosi un account personale. L'interfaccia prevede una procedura di autenticazione (login) che reindirizza l'utente in un ambiente di lavoro personale (cartella di lavoro) dove verranno memorizzati sia i dati di input per il lancio di un esperimento

caricati dall'utente stesso, sia i dati di output ottenuti dall'esperimento in esecuzione. Tali dati vengono memorizzati in VFS(Virtual File Store) che rappresenta un'astrazione del File Store fisico.
Nel proprio ambiente di lavoro ogni utente ha la possibilità di creare varie sessioni (cartelle), che ad esempio potrebbero rappresentare una catalogazione delle varie tipologie di esperimento.
Inoltre in tale ambiente è possibile controllare lo stato di alcuni esperimenti ancora in fase di esecuzione, oltre che trovare procedure guidate per configurare il lancio di un nuovo esperimento, rendendo disponibili gli strumenti e gli algoritmi di data mining implementati nella Suite.
Il Front-End comunica solo con il Framework (vedi paragrafo 2.3.2) tramite documenti di tipo XML (eXtensible Markup Language)

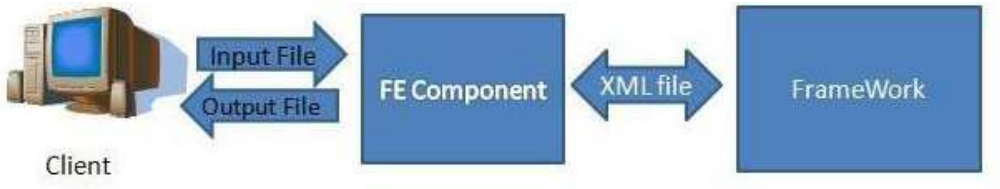


Fig. 4 Comunicazione tra Client FE e FW

2.3.2. FrameWork

Realizzato dai colleghi Michelangelo Fiore e Omar Laurino, il Framework rappresenta il nucleo della suite DAME ed è responsabile del coordinamento di tutti i componenti della suite. Esso ha 2 compiti principali:

- Rispondere alle richieste del Front-End, prendendo le informazioni dal DBMS(DataBase Management System);
- Fornire un servizio agli amministratori della suite, presentando un'interfaccia di amministrazione, che permette di installare nuove funzionalità ed effettuare operazioni statistiche sul sistema;

Il Framework è diviso in 3 parti:

- **Web Service**: parte responsabile di gestire le iterazioni con il Front-End;
- **DMPlugin (Data Mining Plugin)**: GUI che permette a sviluppatori interni ed esterni di aggiungere nuove funzionalità riguardanti le tecniche di data mining

(cioè aggiungere nuovi modelli o farne una variante dei modelli già presenti nella suite) utilizzate dalla suite DAME;

- Questa parte del framework è descritta meglio nel paragrafo 2.3.5;
- **Interfaccia di amministrazione**: permette ad amministratori di installare nuove funzionalità (che non riguardano il data mining), oltre che applicare operazioni statistiche sulla suite;

2.3.3. Registry & DataBase

Il componente Registry & DataBase (REDB), progettato e implementato dal collega Alfonso Nocella, ha il compito principale di memorizzare tutto ciò che avviene all'interno della suite. Infatti, come si può dedurre dal nome esso è:

- **Registry**, nel senso che contiene tutte le informazioni relative agli utenti registrati e gli account, e le loro relative sessioni e esperimenti;
- **DataBase,** perché contiene tutte le informazioni circa i dati di input/output di ogni esperimento, oltre che i dati temporanei e finali provenienti da processi (esperimenti) lanciati dall'utente ancora in fase di esecuzione;

Il Database è modellato con un classico sistema entità-relazione, nel quale il DBMS permette una efficiente navigazione all'interno delle informazioni memorizzate tramite query standard o specifiche. Sia gli utenti che i componenti possono accedere alla base di dati in modo sicuro e protetto.
Il REDB è basato sulle seguenti tecnologie:

- MySQL DBMS Server
- Il connettore JDBC
- API di accesso ai dati

Atte ad implementare le seguenti funzionalità:

- Gestione degli informazioni degli utenti (registrazione, autenticazione, sessioni di lavoro, esperimenti e file) e eventuali relazioni tra queste informazioni;
- Memorizzazione gestione con tre tipi differenti di file:
  - Supported: per i dataset
  - Exotic: per i file di configurazione dei modelli

- Custom: per dati intermedi

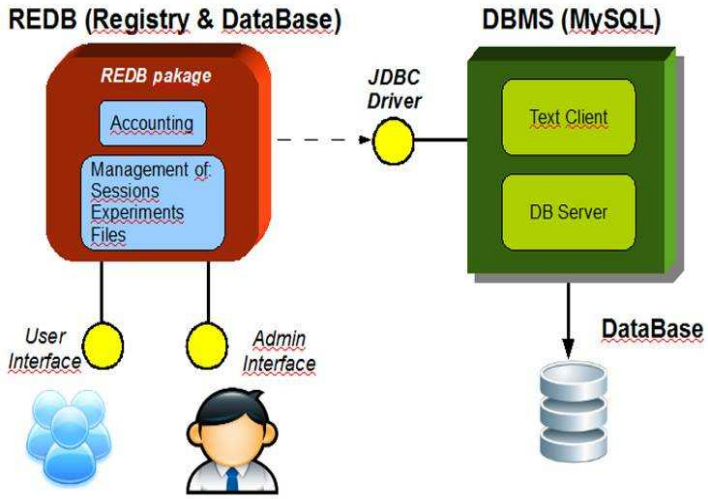


Fig. 5 Comunicazione tra REDB e DBMS

Il Framework si interfaccia sempre con il componente Registry & DataBase (vedi paragrafo 2.3.4) per garantire l'integrità dei dati e la consistenza delle informazioni memorizzate dagli utenti durante la configurazione di un esperimento e durante operazioni di carico o modifica di file.

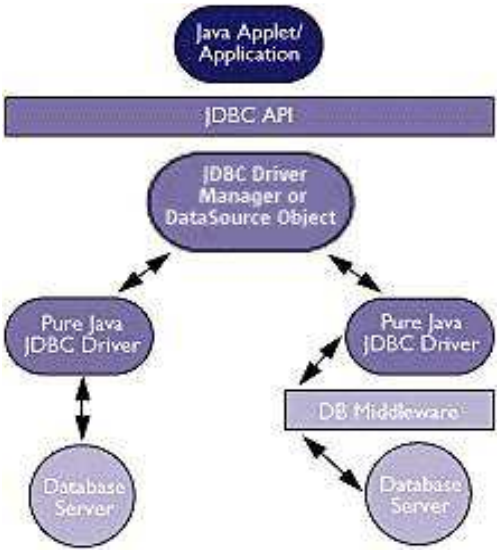


Fig. 6 Architettura del componente REDB con tecnologia Java

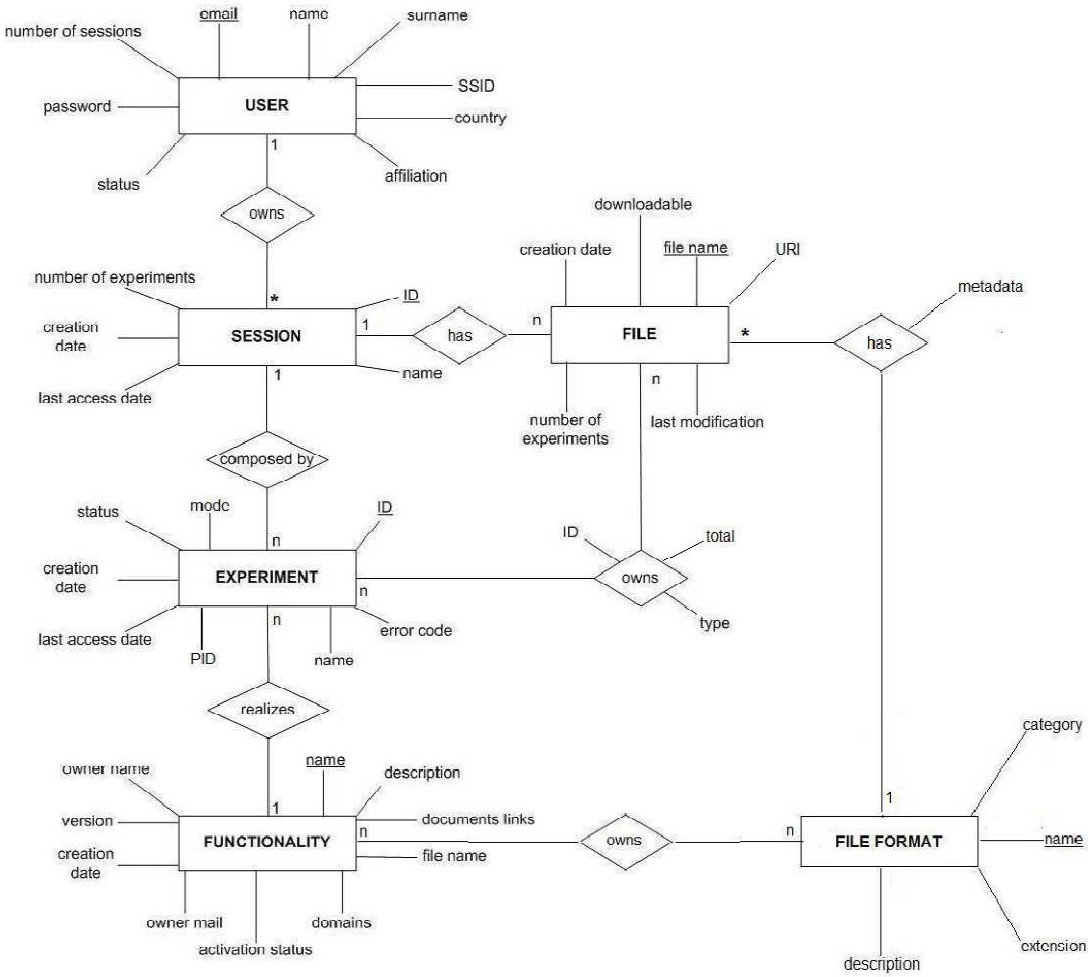


Fig. 7 Diagramma Entità-Relazione del REDB

2.3.4. Driver

Il componente Driver, realizzato dai colleghi Giancarlo D'Angelo e Mauro Garofalo, è stato progettato per separare i requisiti funzionali del Framework dalla sua implementazione.

In altre parole, il Driver ha l'utilità di astrarre il Framework dalla piattaforma su cui esso adopera. Infatti, come già accennato, la Suite DAME è stata progettata per poter funzionare su due diverse piattaforme:

- **GRID**, infrastruttura di calcolo distribuita
- **Stand-Alone**, cioè su singola macchina

Inoltre, il Driver viene utilizzato per permettere, in tutti i casi d'uso, l'accesso adeguato alle risorse della piattaforma utilizzata.

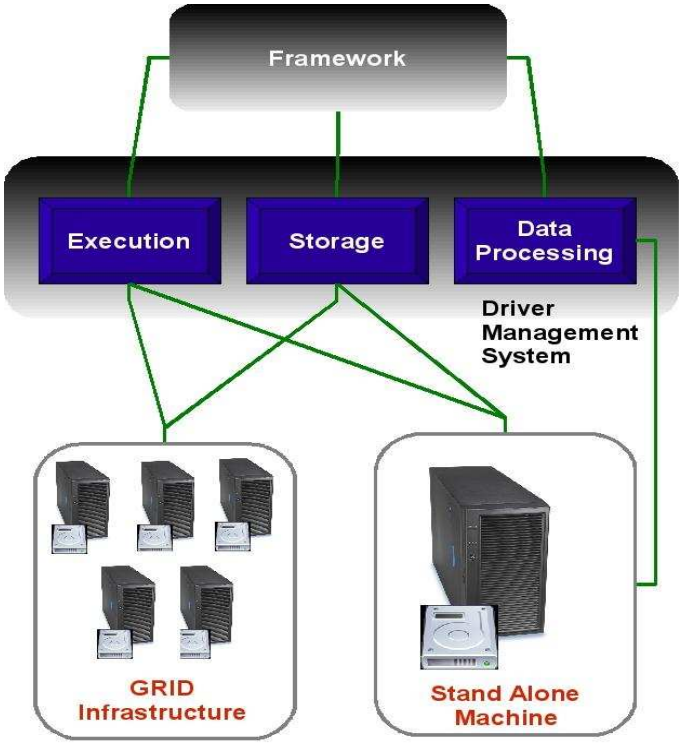


Fig. 8 Funzioni Driver

La fase di I/O di un esperimento consiste essenzialmente nei dati di input forniti dall'utente e nei dati di output forniti dalla suite. Al fine di evitare la ridondanza di questi dati, memorizzandoli in ogni componente, è stato deciso di fornire alla suite un FILESTORE (FS) che si occupa della memorizzazione di file fisici; esso è gestito direttamente dal Framework tramite il Driver.

Inoltre il Driver prevede anche una libreria di codice per la conversione di dati, dando la possibilità all'utente di caricare e utilizzare diversi formati di dati. Infatti i modelli di data mining richiedono diversi formati di dati, perciò l'utente può utilizzare la funzionalità della suite indipendentemente dal formato del file che ha a disposizione, essendo compito del Driver la conversione nel formato giusto, cioè nel formato richiesto dalla funzionalità scelta. Per adesso la suite supporta i seguenti formati:

- FITS (Flexible Image Transport System): è un formato usato per memorizzare, trasmettere, e manipolare immagini scientifiche ad alta risoluzione. FITS è il formato più usato in astronomia, progettato proprio per dati scientifici, contenente quindi un header con molte informazioni relative alla calibrazione fotometrica e spaziale, insieme a vari metadati che riportano dettagli astro metrici.

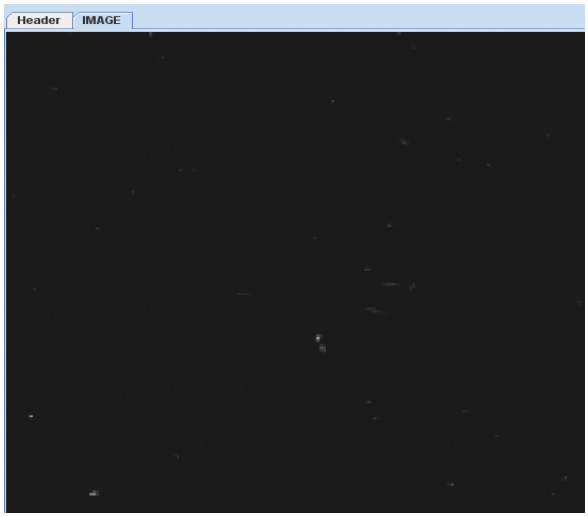


Fig. 9 Immagine FITS

- VOTable : è il formato standard XML stabilito dalla comunità VO per l'interscambio dei dati.

```
<DATA>
<TABLEDATA>
<TR>
  <TD>010.68</TD><TD>+41.27</TD>
  <TD>N 224</TD><TD>-297</TD>
  </TR>
  <TR>
  <TD>287.43</TD><TD>-63.85</TD>
  <TD>6</TD><TD>10.4</TD>
  </TR>
 </TABLEDATA>
</DATA>
```

Fig. 10 File VOTable

- ASCII : è l'acronimo di **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange (ovvero *Codice Standard Americano per lo Scambio di Informazioni*). È un sistema di codifica dei caratteri a 7 bit comunemente utilizzato nei calcolatori.

- CSV (Comma-separated values): è un formato di file basato su file di testo utilizzato per l'importazione ed esportazione (ad esempio da fogli elettronici o database) di una tabella di dati.

OPERA	AUTORE	CASA EDITRICE
I Robot e l'Impero	Isaac Asimov	Mondadori
Il lungo meriggio della Terra	Brian W. Aldiss	Minotauro
Absolute OpenBSD "2d Edition"	Michael W. Lucas	No Starch Press
I mercanti dello spazio	Frederik Pohl, C. M. Kornbluth	Mondadori

Fig. 11 Immagine CSV

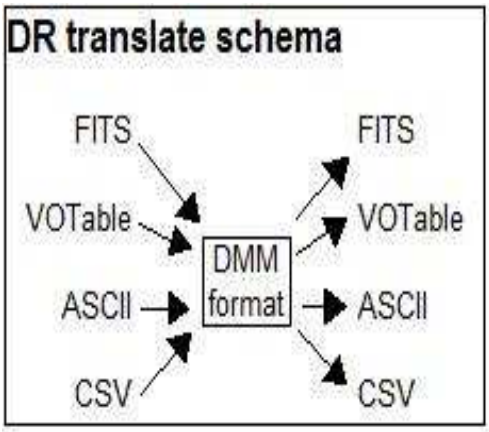


Fig. 12 Schema di traduzione del Driver

In futuro il Driver avrà la capacità di scegliere, in maniera intelligente, quale piattaforma sia più opportuno utilizzare, in base ai parametri dell'esperimento da lanciare.

2.3.5. Il DMPlugin

il DMPlugin(Data Mining Plugin) è una GUI che permette a sviluppatori interni ed esterni di inserire nuovi modelli all'interno della suite, oppure inserire varianti di modelli già inseriti in DAME.

Come già accennato, il DMPlugin, fisicamente è una sottocomponente del Framework, ma può essere tranquillamente considerata come un componente vero e proprio della Suite.

Ho utilizzato questa componente per inserire i Plugin dei modelli di data mining da me trattati (Support Vector Machine, Multi-Layer Perceptron, Multi-Layer Perceptron with Genetic Algorithm) che verranno descritti meglio nel prossimo capitolo.

Nella figura seguente è mostrata l'interfaccia utente di questo componente.

Fig. 13 Interfaccia Utente DMPlugin

Le istruzioni su come inserire un nuovo plugin di un nuovo modello utilizzando l'interfaccia del DMPlugin verranno descritte nel prossimo capitolo.

2.4. Data Mining Models (DMM)

Come già detto, lo scopo principale della suite DAME è quello di fare data mining. Questo paragrafo descrive in maniera sommaria gli strumenti utilizzati per fare data mining, o meglio, descrive dei veri e propri modelli che vengono utilizzati per fare data mining ognuno con i propri vantaggi e svantaggi.

Questi modelli verranno utilizzati per dare la possibilità alla suite di fornire all'esterno le funzionalità che al momento dovrà disporre, ovverossia classificazione e regressione (vedi paragrafi 2.2.1.1 e 2.2.1.2). Tali modelli si basano principalmente su concetti come Artificial Intelligence (AI) e Reti Neurali di cui discuteremo nel prossimo paragrafo.

2.4.1. Le Reti Neurali

Le tecniche ormai consolidate di Artificial Intelligence (AI) si prestano ad un massiccio impiego nell'indagine astrofisica.
Alcune tecniche di AI sono:

- Reti Neurali
- Algoritmi Genetici
- Sistemi esperti

L'idea originaria delle reti neurali artificiali si ispira alla struttura del cervello umano.
Nell'uso moderno si intende solitamente una rete di neuroni artificiali interconnessi a strati, che cerca di simulare il funzionamento dei neuroni biologici. Può essere composta sia da programmi che da hardware dedicato. In quasi tutti gli organismi viventi sono presenti complesse organizzazioni di cellule nervose, con compiti di riconoscimento delle configurazioni assunte dall'ambiente esterno, memorizzazione e reazione agli stimoli provenienti dallo stesso. Il cervello umano rappresenta probabilmente il più mirabile frutto dell'evoluzione per le sue capacità di elaborare informazioni. Al fine di compiere tali operazioni, le reti biologiche si servono di un numero imponente di semplici elementi computazionali (neuroni) fittamente interconnessi in modo da variare la loro configurazione in risposta agli stimoli esterni: in questo senso può parlarsi di apprendimento ed i modelli artificiali cercano di emulare questo tratto distintivo della biologia.
Una rete neurale artificiale riceve segnali esterni su uno strato di nodi (unità di elaborazione) d'ingresso, ciascuno dei quali è collegato con numerosi nodi interni, organizzati in più livelli. Ogni nodo elabora i segnali ricevuti e trasmette il risultato a nodi successivi.
Le reti neurali, per come sono costruite, lavorano in parallelo e sono quindi in grado di trattare molti dati. Si tratta in sostanza di un sofisticato sistema di tipo semi-statistico dotato di una buona immunità al rumore; se alcune unità del sistema dovessero funzionare male, la rete nel suo complesso avrebbe delle riduzioni di prestazioni, ma difficilmente andrebbe incontro ad un blocco del sistema. I software di ultima generazione dedicati alle reti neurali richiedono comunque buone conoscenze statistiche; il grado di apparente utilizzabilità immediata non deve trarre in inganno, pur permettendo all'utente di effettuare da subito previsioni o classificazioni, seppure con i limiti del caso.
Nelle industrie, qualora si dispongano dei dati cosiddetti storici, ovverossia delle passate sperimentazioni, le reti neurali risultano interessanti in quanto permettono di estrarre dati

e modelli senza effettuare ulteriori prove. I modelli prodotti dalle reti neurali, anche se molto efficienti, non sono spiegabili in linguaggio simbolico umano: i risultati vanno accettati "così come sono", da cui anche la definizione inglese delle reti neurali come "black box". Come per qualsiasi algoritmo di modellazione, anche le reti neurali sono efficienti solo se le variabili predittive (parametri su cui si basa la rete per l'apprendimento) sono scelte con cura. Non sono in grado di trattare in modo efficace variabili di tipo categorico (per esempio, il nome della città) con molti valori diversi. Necessitano di una fase di addestramento del sistema che fissi i pesi dei singoli neuroni e questa fase può richiedere molto tempo, se il numero dei record e delle variabili analizzate è molto grande.

Non esistono teoremi o modelli che permettano di definire la rete ottima, quindi la riuscita di una rete dipende molto dall'esperienza del creatore.

La figura sottostante mostra una tassonomia delle Reti Neurali.

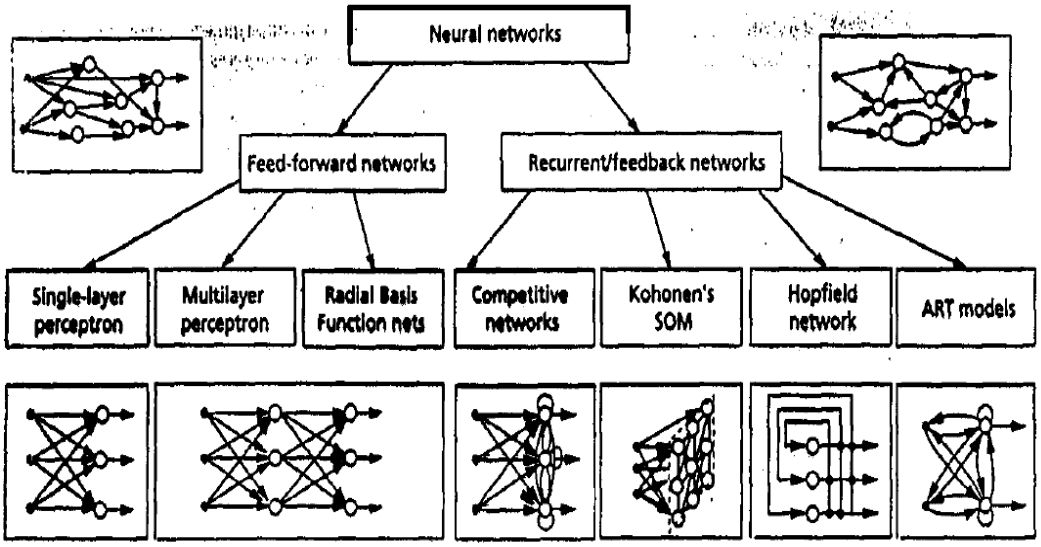


Fig. 14 Tassonomia Reti Neurali

Esse si dividono principalmente in:

- **Reti feed-forward**: flusso di dati uni-direzionale attraverso vari strati. Un neurone è collegato a neuroni di altri strati.
- **Reti recurrent**: un neurone può essere collegato anche a se stesso.

2.4.2. Modalità di Apprendimento

Dal punto di vista della modalità d'apprendimento, è necessario operare una distinzione tra almeno due diverse tipologie: **con e senza supervisione.**

2.4.2.1. Apprendimento con supervisione

Un apprendimento si dice "con supervisione" (supervised learning), qualora si disponga di un insieme di dati per l'addestramento (o training set) comprendente esempi tipici d'ingressi con le relative uscite loro corrispondenti: in tal modo la rete può imparare ad inferire la relazione che li lega.
Successivamente, la rete è addestrata mediante un opportuno algoritmo (tipicamente, la Back Propagation, algoritmo d'apprendimento supervisionato), il quale usa tali dati allo scopo di modificare i pesi ed altri parametri della rete stessa in modo tale da minimizzare l'errore di previsione relativo all'insieme d'addestramento. Se l'addestramento ha successo, la rete impara a "generalizzare", ossia a riconoscere la correlazione incognita che lega le variabili d'ingresso a quelle d'uscita, in grado quindi di fare previsioni anche laddove l'uscita non sia nota a priori; in altri termini, l'obiettivo finale dell'apprendimento supervisionato è la previsione del valore dell'uscita per ogni valore valido dell'ingresso, basandosi soltanto su un numero limitato di esempi di corrispondenza (vale a dire, coppie di valori input-output). Ciò consente di risolvere problemi di regressione o classificazione con estrema accuratezza e robustezza.
Per esempio immaginiamo di voler costruire un programma di riconoscimento vocale: addestriamo la nostra rete registrando un numero sufficiente di frasi di cui conosciamo la classe (in questo caso spesso esistono frasi e parole specifiche che ottimizzano questa fase) e la rete sarà in grado di riconoscere anche parole non presenti nel campione con cui è stata addestrata.

2.4.2.2. Apprendimento senza Supervisione

Un apprendimento si dice "senza supervisione" (unsupervised learning), se è basato su algoritmi d'addestramento che modificano i pesi della rete facendo esclusivamente riferimento ad un insieme di dati che include le sole variabili d'ingresso. Tali algoritmi tentano di raggruppare i dati d'ingresso e di individuare pertanto degli opportuni cluster rappresentativi dei dati stessi, facendo uso tipicamente di metodi topologici o statistici. L'apprendimento non supervisionato è anche impiegato per sviluppare tecniche di compressione dei dati. Vediamo ora anche un esempio di addestramento non supervisionato.

Immaginiamo di avere un certo numero di pazienti, soggetti a quella che per noi è una patologia, di ognuno dei quali conosciamo un certo numero di parametri clinici e eventuali sintomi, ma non con estrema certezza; tramite una rete neurale non supervisionata possiamo raggruppare questi pazienti in un certo numero di sottoclassi aventi caratteristiche accomunanti e quindi individuare eventualmente errori di diagnostica o casi parzialmente correlati con buona affidabilità.

2.4.3. Modelli di apprendimento automatico utilizzati in DAME

Nei paragrafi successivi verranno descritti, in maniera sintetica, i modelli di apprendimento automatico di cui la suite DAME, al momento, dispone.
I modelli sono stati implementati sia in Java, come il Support Vector Machine (SVM), che in C++ come per il MultiLayer Perception (MLP) e il MultiLayer Perception with Genetic Algorithms (MLPGA). Essi in pratica sono delle vere e proprie librerie, ognuna delle quali fornisce metodi che implementano tutte le funzionalità descritte nel paragrafo 2.4.2.

2.4.3.1. Il Modello Support Vector Machine (SVM)

Le Support Vector Machine (SVM) sono un insieme di metodi di apprendimento supervisionato che possono essere utilizzati sia per fare classificazione sia per fare regressione. In un breve lasso temporale dalla loro prima implementazione hanno trovato numerose applicazioni in un nutrito varie branche scientifiche come fisica, biologia, chimica.

- **Preparazione di farmaci** (discriminazione tra leganti e non leganti, inibitori e non inibitori, etc.);
- **Ricerca di relazioni quantitative sulle attività di strutture** (dove le SVM, utilizzate come regressori sono usate per trovare varie proprietà fisiche, biologiche e chimiche);
- **Chemiometria** (ottimizzazione della separazione cromatografica o per la misura della concentrazione di un composto basandosi sui dati spettrali ad esempio);
- **Sensori** (per ottenere informazioni su parametri non realmente misurati dal sensore ma di interesse);
- **Ingegneria chimica** (ricerca degli errori e modellazione dei processi industriali),
- etc. (ad esempio riconoscimento di volti in una foto o in un filmato, utilizzato da alcuni aeroporti americani per individuare ricercati);

I modelli SVM furono originariamente definiti per la classificazione di classi di oggetti linearmente separabili. Per ogni gruppo di oggetti divisi in due classi una SVM identifica l'iperpiano avente il massimo margine di separazione (nella fig. 15 di può vedere come la linea verde non separa le due classi, la linea blu le separa ma con un piccolo margine mentre la linea rossa massimizza la distanza delle due classi).

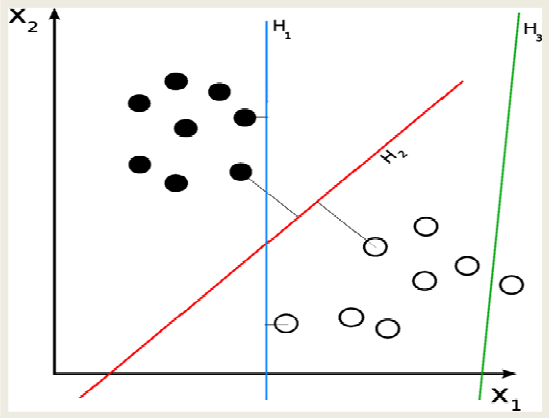


Fig. 15 Separazione di funzioni tramite iperpiani

Nella fig. 16 l'iperpiano H1 definisce il bordo della classe i cui oggetti sono rappresentati dai "+1" mentre l'iperpiano H2 quello degli oggetti rappresentati dai "-1". Potete notare che due oggetti della classe "+1" servono a definire H1 (sono quelli cerchiati) e ne servono tre della classe "-1" per definire H2; questi oggetti vengono chiamati "support vectors", quindi il nostro problema di identificare la miglior separazione tra le due classi è risolto individuando i vettori di supporto che determinano il massimo margine tra i due iperpiani.

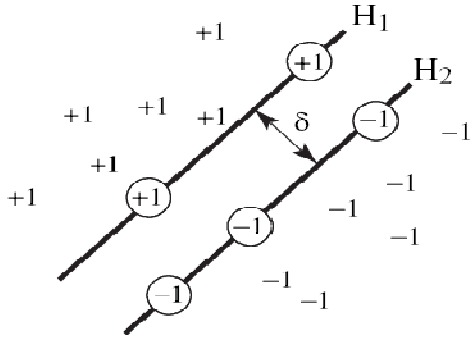


Fig. 16 Individuazione dei Support Vectors

In un piano le combinazioni di tre punti possono essere separate da una linea però già quattro punti non è detto che lo siano

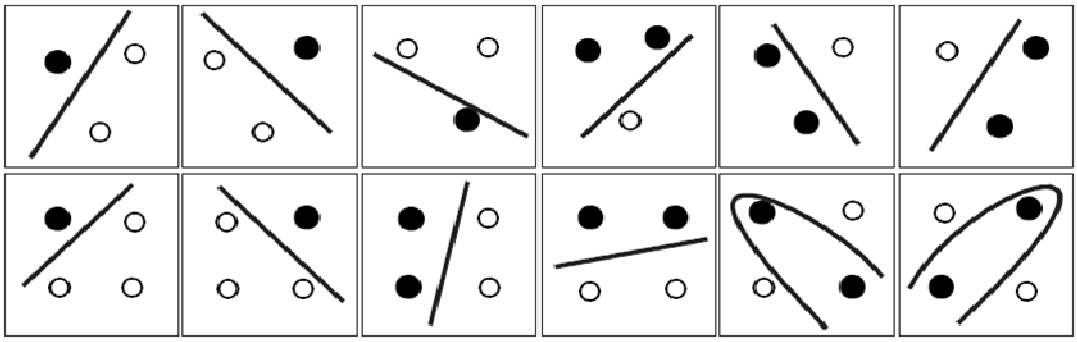


Fig. 17 Separazioni di punti tramite linea

Ovviamente le SVM possono essere usate per separare classi che non potrebbero essere separate con un classificatore lineare, altrimenti la loro applicazione a casi di reale interesse non sarebbe possibile. In questi casi le coordinate degli oggetti sono mappate in uno spazio detto "*feature space*" utilizzando funzioni non lineare, chiamate "*feature function*" ϕ. Il feature space è uno spazio fortemente multidimensionale in cui le due classi possono essere separate con un classificatore lineare.

Quindi lo spazio iniziale viene rimappato nel nuovo spazio, a questo punto viene identificato il classificatore che poi viene riportato nello spazio iniziale, come illustrato in figura.

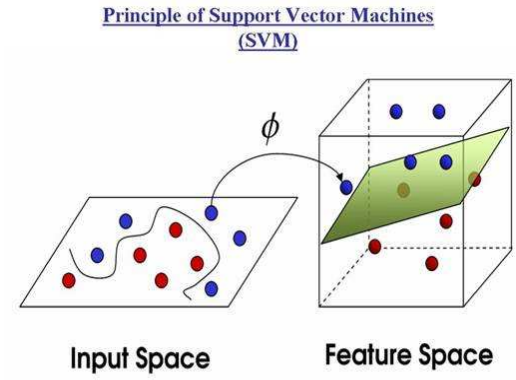


Fig. 18 Principio delle SVM

A causa del fatto che lo spazio iniziale ha molte dimensioni non sarebbe pratico utilizzare una funzione generica per trovare l'iperpiano di separazione, quindi vengono usate delle funzioni dette "kernel" e si identifica la funzione ϕ tramite una combinazione di funzioni di kernel.

Tra le implementazioni più famose delle SVM c'è la *libSVM* , che è stata creata da Chih-Wei Hsu, Chih-Chung Chang e Chih-Jen Lin e implementata nei più disparati linguaggi (Java, c++, phyton, matlab ed R per citare le sole libSVM) e mette a disposizione quattro possibili kernel:

- Lineare
- Polinomiale
- Radial Basis Function (RBF)
- Sigmoid

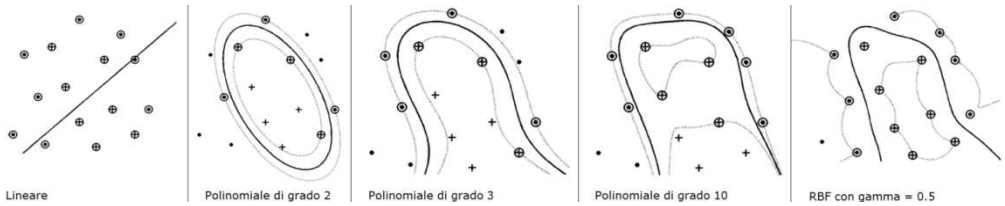


Fig. 19 Applicazione dei kernel della SVM

Come si può vedere in Fig. 19 il kernel lineare non è assolutamente adatto a questo esempio, mentre gli altri 4 riescono a discriminare le due classi perfettamente, sebbene le soluzioni siano molto differenti l'una dall'altra. E' importante quindi avere un set di prova che permetta di scegliere la migliore configurazione in modo da evitare quello che si chiama usualmente "over-fitting", ossia il caso in cui l'algoritmo si adatti molto ai dati con cui è addestrato, non riuscendo poi a generalizzare il problema.

Si può notare inoltre che, eccezion fatta per il kernel lineare, parliamo non di funzioni semplici ma famiglie di funzioni, che dipendono da un certo numero di parametri (detti usualmente "hyper-parameters"). Questo se da un lato ci dà maggiori speranze di individuare la soluzione ottimale, dall'altro complica il nostro lavoro di ricerca dal momento che dobbiamo cercare il kernel con i migliori parametri.

Le SVM, che come detto nascono per risolvere problemi di classificazione, furono estese da Vapnik al problema della regressione.

Il set di parametri con cui si addestra il modello è utilizzato per ottenere un modello di regressione che può essere rappresentato come un ipertubo di raggio e, che sarà quindi un ulteriore parametro, visto che la sua grandezza determinerà il risultato ottenuto. Nel caso ideale, la regressione tramite le SVM trova una funzione che mappa tutti i dati di input con una deviazione massima pari proprio ad ε, dal valore del "target". In questo caso tutti i punti con cui si addestrano le SVM si trovano all'interno del tubo di regressione. Comunque, usualmente non è possibile "fittare" tutti gli oggetti all'interno del tubo e avere ancora un modello che abbia un qualche significato quindi nel caso generale le SVM in modalità di regressione considerano zero l'errore per gli oggetti all'interno del tubo mentre quelli all'esterno hanno un errore che dipende dalla distanza dal margine del tubo

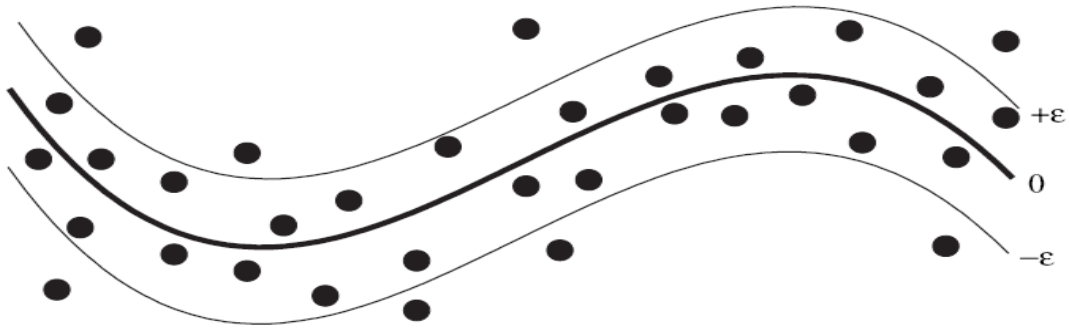


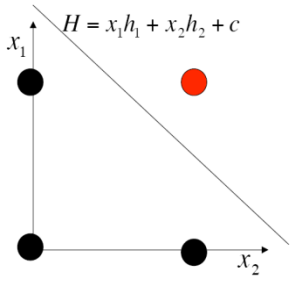
Fig. 20 Rappresentazione regressione delle SVM

2.4.3.2. Il Modello Multi-Layer Perceptron (MLP)

Il modello Multi-Layer Perceptron è una rete neurale feedforward (vedi paragrafo 2.4.1), supervisionata, che ha lo scopo principale di risolvere problemi di classificazione e regressione.

Essendo un rete supervisionata, essa necessita di una fase di addestramento (vedi paragrafo 2.4.4.1)

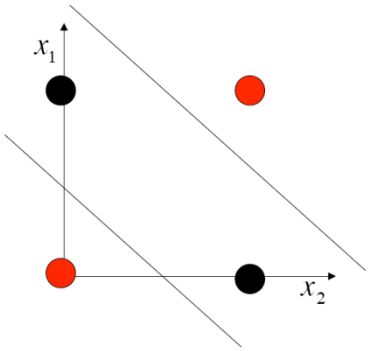
Essa può essere vista come lo sviluppo del Single Layer Perceptron, che per definizione, è il tipo di rete neurale artificiale feed-forward più semplice, capace di classificare datasets linearmente separabili.



x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

Fig. 21 Una funzione linearmente separabile: AND logico

Il Multi-Layer Perceptron nasce con l'esigenza di dover classificare datasets o più semplicemente **funzioni non linearmente separabili.**



x_1	x_2	t
0	0	1
0	1	0
1	0	0
1	1	1

Fig. 22 Funzione non linearmente separabile: lo XOR

La funzione XOR è un esempio di funzione non linearmente separabile, quindi è necessario un percettrone multistrato (MLP) come mostra la figura 17.

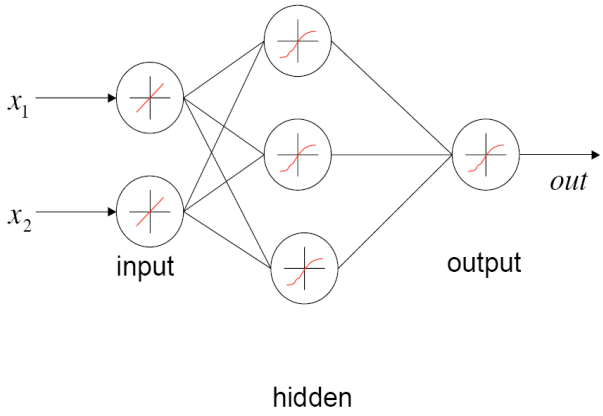


Fig. 23 MLP che simula lo XOR

2.4.3.3. Il Modello Multi-Layers Perceptron with Genetic Algorithm (MLPGA)

Questo modello può essere considerato come un modello ibrido tra l'MLP e Algortmi Genetici.

Esso è stato realizzato con la tecnica del *"soft-computing"* che consiste nel realizzare modelli ibridi che ereditano concetti insiti in vari modelli autoadattivi.

Gli algoritmi genetici si basano sull'evoluzione darwiniana delle specie viventi. Essi consistono in un processo ciclico di evoluzione di generazione, inizialmente casuale, di una popolazione di individui. Ad ogni generazione, mediante una meccanismo di riproduzione, vengono generati nuovi individui, manipolando il materiale genetico della popolazione precedente. Gli individui, in competizione tra di loro, che meglio si adattano all'ambiente hanno maggiori probabilità di sopravvivenza e di trasmettere il patrimonio

genetico alle generazioni future, facendo evolvere la popolazione di generazione in generazione, incrementando il numero degli individui migliori (soluzioni al problema in esame) in essa presenti.
A differenza dell'MLP la fase di addestramento nel modello MLPGA consiste nel proporre diverse "popolazioni" di reti neurali, scegliendo quella che meglio risponde alle coppie di input-output utilizzate nella fase di addestramento.

2.4.3.4. Differenze tra i Modelli

In questo paragrafo verranno messi a confronto i vari modelli implementati dalla suite DAME.
Statisticamente l'MLPGA converge più rapidamente dell'MLP, trovando a volte soluzioni migliori.
Il modello MLP può trovare soluzioni non ottimali, essendo soggetta al classico problema dei "minimi locali"; tale problema può essere bypassato utilizzando il modello SVM.
In conclusione SVM e MLPGA funzionano meglio, ma entrambi hanno due svantaggi di non poco conto:

- L' SVM è pesante in termini computazionali;
- L' MLPGA soffre di un grande numero di gradi di libertà che rendono difficile la configurazione dei parametri per trovare una soluzione ottimale;

2.4.4. Le Funzionalità principali dei DMM

In questo paragrafo verranno descritte le principali operazioni effettuabili sui DMM, come la fase di addestramento e quella di test.

2.4.4.1. L'addestramento di una rete neurale: la funzione "train"

Consiste nel fornire alla rete un certo set di dati (training set) al fine di realizzare un addestramento previo al loro utilizzo sul problema concreto che si vuole risolvere.
Questo set di dati di addestramento (*training set*) deve contenere i risultati osservati per ogni particolare combinazione dei parametri di input. Lo scopo della fase di addestramento è quello di minimizzare l'errore commesso tra i risultati forniti dalla rete (output) e i dati osservati. L'errore generato viene misurato però non sul training set, ma su un ulteriore gruppo di dati (*validation set*) in modo da evitare il fenomeno noto come *overfitting*, nel quale la rete diventa così aderente ai dati del training set che risulta

inutilizzabile al di fuori di questo insieme di dati, cioè perde le capacità di generalizzazione per le quali è stata progettata. Quindi, la verifica del funzionamento della rete è basato sul metodo di *cross validation* dei dati; a questo scopo la totalità dei dati a disposizione viene suddivisa in due parti corrispondenti al training set e al validation set.

2.4.4.2. Il lancio di una rete neurale: La funzione "Run"

Questa fase consiste nell'utilizzo vero e proprio della rete neurale. Cioè consiste nel fornire alla rete delle combinazione dei parametri di input di cui non si conoscono i risultati a priori; sarà la rete neurale a fornire l'output corrispondente che, nel caso delle reti neurali supervisionate (vedi paragrafo), è calcolato in base ai dati forniti nella fase di addestramento della rete stessa (vedi paragrafo 2.4.3.1).

2.4.4.3. Valutare l'accuratezza di una rete neurale:La funzione "Test"

Questa fase consiste nel testare la bontà dell'apprendimento della rete neurale supervisionata. Alla rete viene fornito un insieme di input, non utilizzato nella fase di addestramento, ma di cui si conosce l'output per valutare l'accuratezza della rete neurale, in termini di percentuale di risposte corrette.
Quindi nasce l'esigenza di non fornire tutto il dataset nella fase di addestramento, ma di destinare una parte del dataset alla fase di Test della rete neurale.

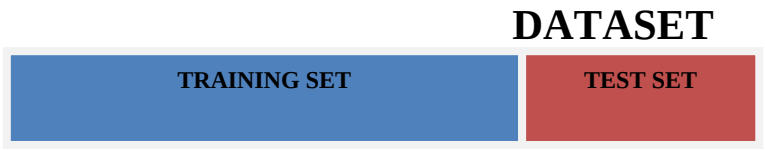


Fig. 24 Divisione Dataset

2.4.4.4. La Funzione "Full"

È semplicemente la successione delle tre fasi descritte in precedenza (in sequenza la fase di Train, quella di Test e infine quella del Run), cioè consiste nel dare in input alla rete prima un set di dati di cui si conosce l'output per addestrare la rete, poi dare alla rete un set di dati non utilizzato in precedenza, ma di cui si conosce l'output, per valutare l'accuratezza della rete, e infine vengono dati degli input di cui non si conosce l'output, che verranno calcolati dalla rete.

3. Il componente DMM nel Progetto DAME

In questo Capitolo verrà descritto il lavoro da me svolto all'interno della suite DAME, cioè la progettazione e l' implementazione del componente Data Mining Model (DMM). Tale componente si occupa di produrre, tramite l'utilizzo di modelli, i file di output che la web application ritornerà all'utente finale.

Il mio lavoro è stato svolto con la collaborazione dal Dott. Stefano Cavuoti, laureato in Fisica, fondamentale sia per chiarimenti su molti concetti basati sul data mining mai incontrati prima, sia per i requisiti che la mia componente ha dovuto rispettare.

In sintesi, il mio lavoro consisteva nel fornire alla suite, attraverso l'utilizzo di alcuni modelli di apprendimento automatico (SVM, MLP, MLPGA), scritti sia in Java che in C++, le funzionalità di classificazione e di regressione.

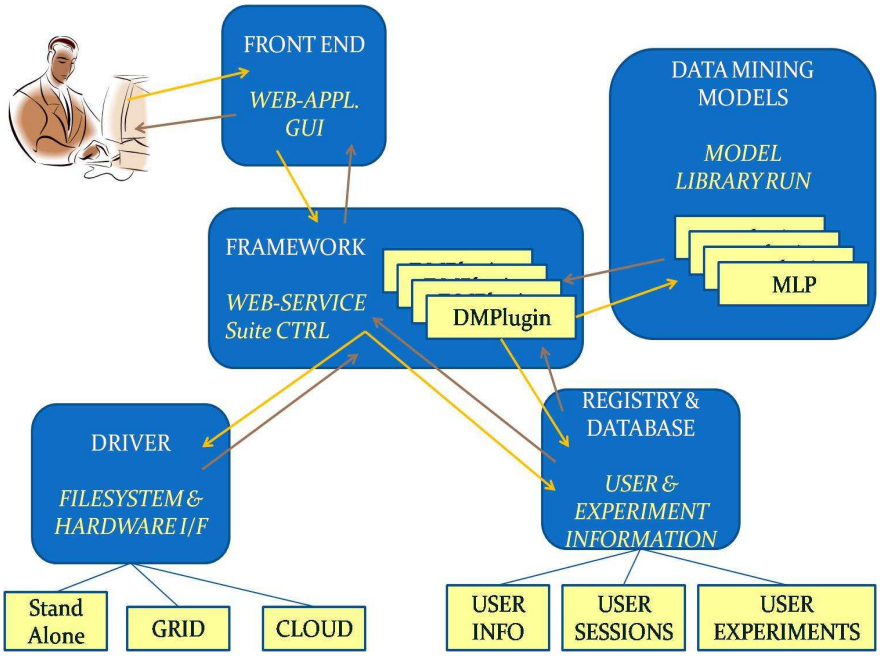


Fig. 25 Gerarchia funzionale dei componenti della suite DAME

3.1. Progettazione del componente DMM

In questo paragrafo verrà descritta la fase di progettazione del componente DMM, a partire dalla descrizione del pattern utilizzato ("Bridge Pattern"), fino ad arrivare al diagramma UML completo di tutta la componente.

La progettazione iniziale ha prodotto l' individuazione di 4 punti:

- **Implementazione finalizzata alle funzionalità: classi di funzionalità (Classificazione, Regressione)**. Abbiamo individuato due funzioni principali: la classificazione e la regressione che possono essere attuate da più di un modello. Questo ci ha portato alla costruzione di due classi ("Regression" e "Classification") che rappresentano il livello astratto di questa componente, mentre il livello di implementazione è rappresentato dai modelli (SVM, MLP, MLPGA per il momento).
- **Dare la possibilità di utilizzare queste due funzionalità con più di un modello senza duplicazione di codice:** Utilizzo del "Bridge Pattern" (vedi paragrafo 3.1.2)
- **Separazione dei modelli con supervisione e senza supervisione:** Abbiamo creato una classe astratta comune ("Supervised") per tutti i modelli con supervisione, individuando alcuni metodi comuni a tutti i modelli di questo tipo. Analogamente una classe astratta comune ("Unsupervised") è stata destinata a tutti i metodi senza supervisione.
- **Realizzazione di un'interfaccia comune a tutti parametri dei modelli:** per tale motivo è stato progettato e implementato un Pacchetto di classi Java chiamato "DmmParams" (vedi par 3.1.1) le cui classi contenute verranno utilizzate come tipi generici dei parametri di tutti i modelli. Per ogni parametro sarà possibile individuare alcune caratteristiche (nome, tipo, valore, vincoli).

3.1.1. Il pacchetto "DmmParams"

Questo pacchetto è stato creato principalmente per ottenere una generalizzazione di tutti gli oggetti strettamente legati alla configurazione di un modello auto-adattivo che sono:

- I parametri dei modelli
- File di Input
- File di Ouput

Perciò sono state create delle classi specifiche per ogni tipologia di oggetti relativi ai modelli di apprendimento:

Classe	Descrizione
DmmFieldParam	Classe per tutti i parametri dei modelli, essa contiene metodi utilissimi per la ottenere e settare informazione sui parametri come il nome o l'uri.
DmmFileParam	Classe per tutti i file di Input o Input-Output relativi ai modelli, anch'essa contiene molti metodi utili come il metodo setter "setUri" per cambiare l'Uri del file.
DmmOutputfileParam	Riguarda esclusivamente tutti i file di output di ogni modello.

Di seguito è riportato il diagrama UML del Pacchetto DmmParams:

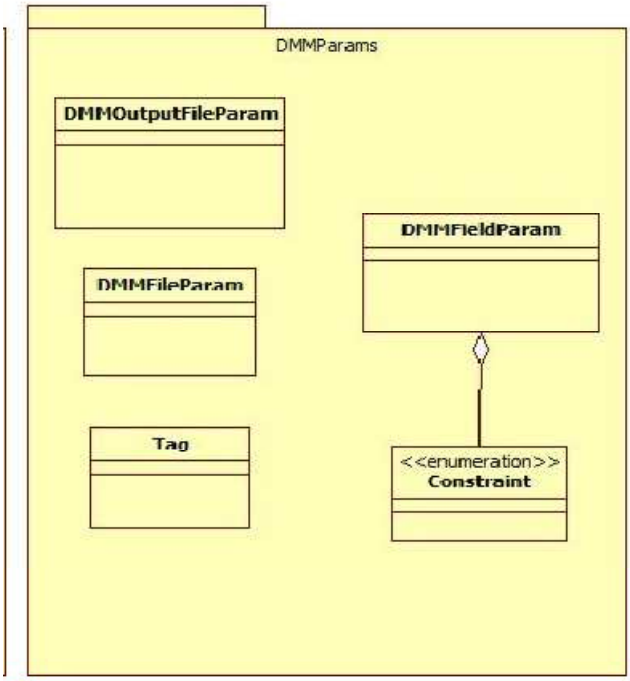


Fig. 26 UML del Pacchetto DmmParams

3.1.2. Il Pattern "Bridge Pattern"

L'esigenza di avere due livelli, uno di astrazione riguardante le funzionalità, l'altro riguardante l'implementazione dei modelli, nasce dal fatto che tutti i modelli finora implementati possono essere utilizzati sia per fare classificazione che per fare regressione. Il problema principale di questa struttura è che senza un'accurata organizzazione delle classi si avrebbe una fastidiosa duplicazione di codice, che potrebbe

portare ad una difficile individuazione degli errori, nonché ad uno spreco di tempo e di risorse. Per tale motivo è stato scelto di ispirare la struttura del DMM al Bridge Pattern.
Infatti, il Bridge Pattern, è un design pattern utilizzato in Ingegneria del Software che ha lo scopo principale di separare un'astrazione dalla sua implementazione senza duplicazione di codice, in modo che possano variare indipendentemente. Esso utilizza la tecnica dell'incapsulamento, l'aggregazione, e può utilizzare l'ereditarietà per responsabilità distinte, ognuna identificata in una classe.
Il Bridge Pattern è descritto dal seguente diagramma UML:

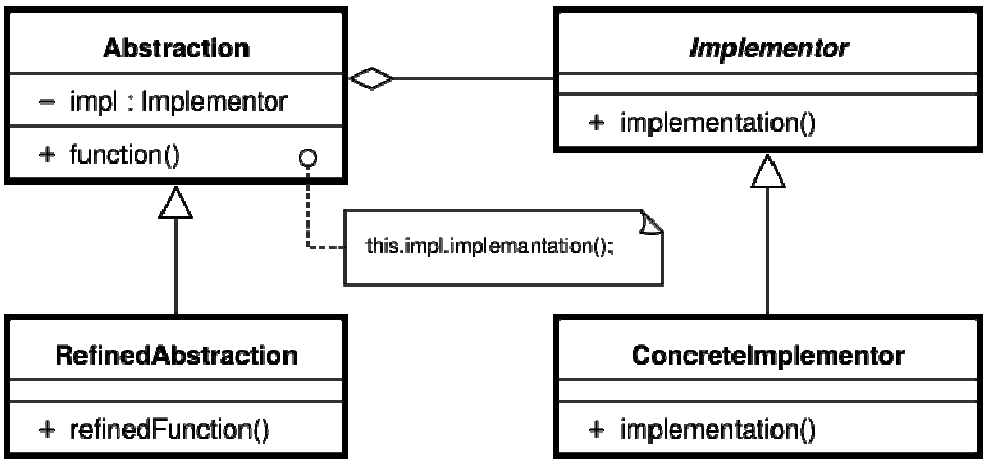


Fig. 27 UML del Bridge Pattern

La classe astratta "Abstraction" rappresenterà il livello funzionale del componente, che sarà estesa dalle classi che rappresenteranno le funzionalità dell'intera suite ("Classification","Regression") mentre la classe astratta "Implementor" rappresenta invece il livello di implementazione dei modelli, quindi la classe "ConcreteImplementor" rappresenta l'implementazione di un modello.
Di seguito viene proposto un esempio di utilizzo del Bridge Pattern:

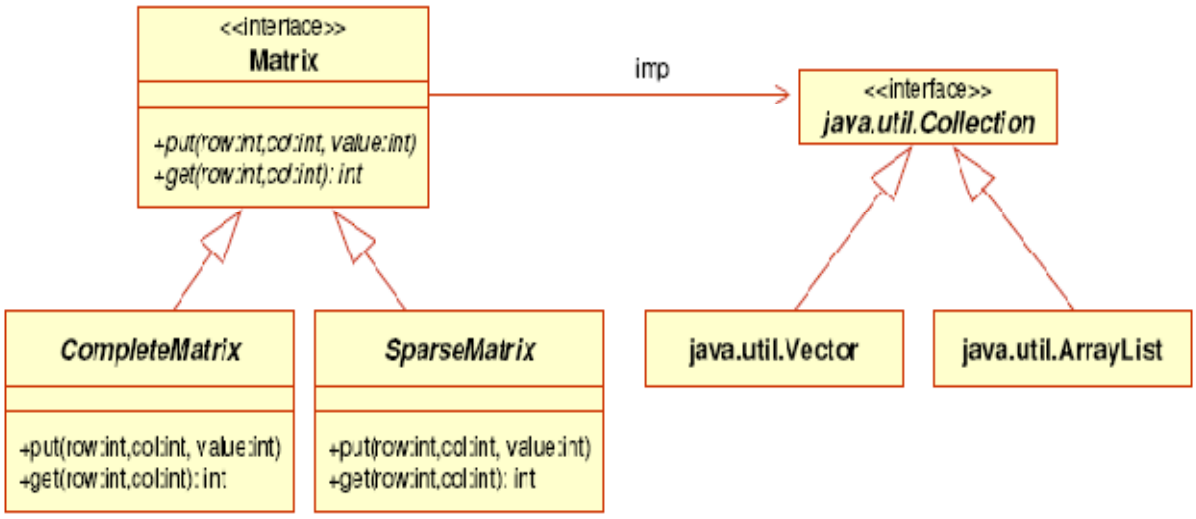


Fig. 28 Esempio di utilizzo del Bridge Pattern

Nell'esempio in fig. 26 il livello astratto è rappresentato dalle classi "CompleteMatrix" e "Sparse Matrix" mentre il livello di implementazione è rappresentato da un vettore o da una lista di array.

3.1.3. Adattamento al Bridge Pattern del DMM

Come già accennato, abbiamo bisogno di separare le funzionalità della suite (classificazione, regressione) dall'implementazione dei modelli. Il livello funzionale del DMM sarà rappresentato dalle funzionalità di classificazione e regressione, mentre il livello di implementazione dai modelli di apprendimento automatico. Più specificatamente avremo un'interfaccia che generalizza ogni modello con supervisione implementato dall'intera suite (chiamata "DMMInterface"); ogni modello avrà una classe con responsabilità mirate esclusivamente a quel modello, mettendo a disposizione tutte le funzionalità del modello stesso.

Logicamente il livello di astrazione è così strutturato:

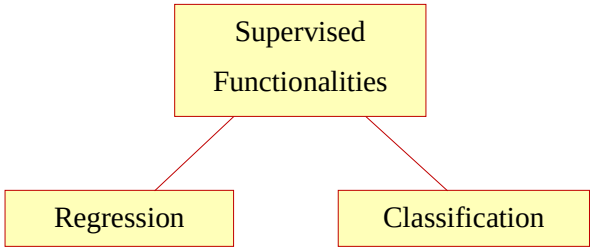


Fig. 29 Livello di astrazione : le funzionalità della suite DAME

Mentre il livello di implementazione sarà strutturato nella maniera seguente:

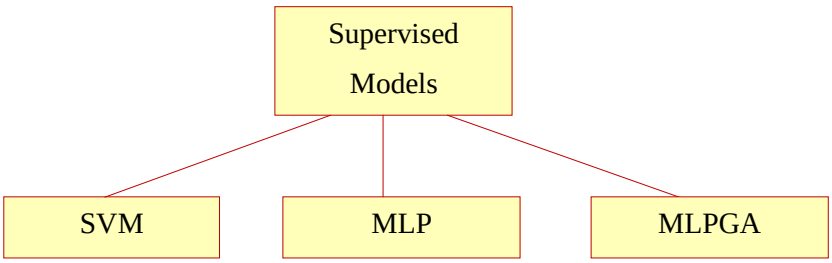


Fig. 30 Livello di implementazione : I modelli

il risultato prodotto da questa struttura a due livelli tramite l'utilizzo del bridge pattern è il seguente:

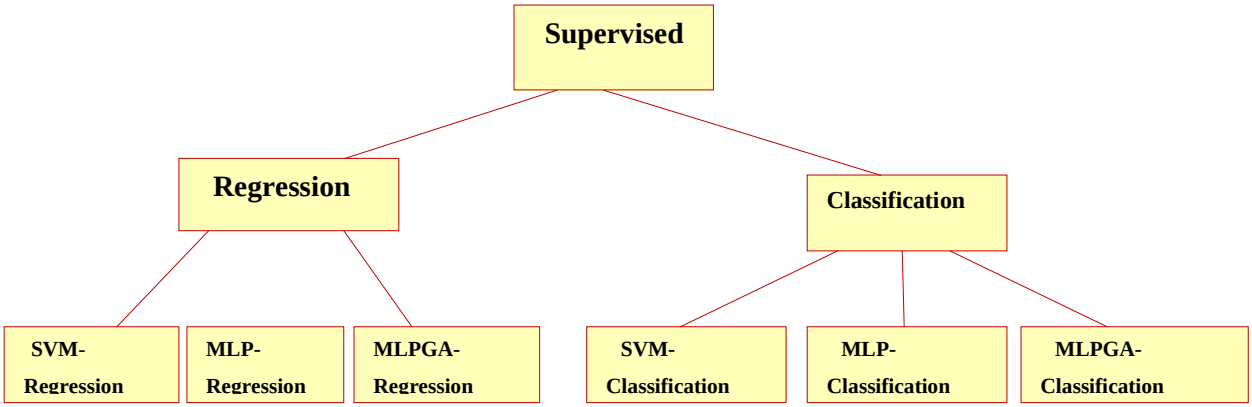


Fig. 31 Risultato prodotto dalla struttura a due livelli

3.1.3.1. Il livello funzionale

Come già accennato, il livello Funzionale consiste, per adesso, in una classe astratta che generalizzerà tutte le funzionalità dei modelli con supervisione. Chiamiamo questa classe astratta "Supervised" che sarà estesa da due classi concrete: "Classification" e "Regression" come mostra la figura sottostante:

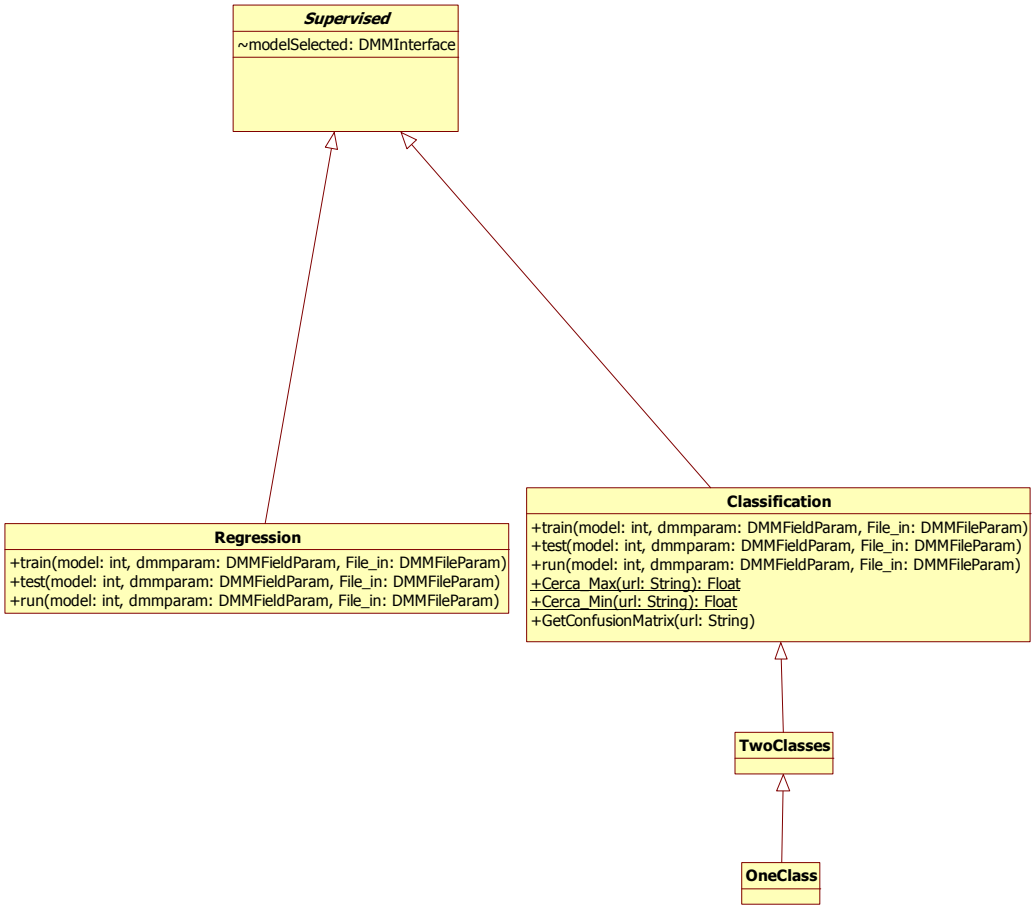


Fig. 32 Diagramma UML livello delle funzionalità

Come si può notare la classe “Classification” ha due livelli di specializzazione:

- La classe “TwoClasses” : in cui verranno inseriti metodi e attributi specifici di una classificazione con due classi.
- La classe “OneClasses” : in cui verranno inseriti metodi e attributi specifici nel caso si volesse effettuare una classificazione con una sola classe.

Avendo a disposizione modelli di apprendimento automatico non supervisionati sarà aggiunta una nuova classe astratta che si chiamerà “Unsupervised”, che generalizza tutte le funzionalità non supervisionate.

3.1.3.2. Il livello di Implementazione dei modelli

Viceversa, il livello di implementazione, consiste in una interfaccia chiamata “DMMInterface” che rappresenta la generalizzazione tutti i modelli con supervisione implementati dalla suite (in realtà il nome non è totalmente appropriato dato che si tratta

di una generalizzazione dei soli modelli con supervisione), ed ha 3 metodi corrispondenti a 3 funzionalità su cui si basano i modelli di apprendimento automatico con supervisione, il quale riceve in input dei parametri e dei File di I/O:

- **public void train(DMMFieldParam[] dmmparam, DMMFileParam[] File_in) throws IOException**:
  è la firma esatta del metodo che corrisponde all'addestramento del modello genrico (vedi paragrafo 2.4.4.1).

- **public void run(DMMFieldParam[] dmmparam, DMMFileParam[] File_in) throws IOException:** è la firma esatta del metodo che corrisponde all'utilizzo del modello (vedi paragrafo 2.4.4.2) .

- **public void test(DMMFieldParam[] dmmparam, DMMFileParam[] File_in) throws IOException:** è la firma esatta del metodo che si occupa della valutazione dell'apprendimento del modello sulla base dei parametri di input (vedi paragrafo 2.4.4.2).

Il caso d'uso "Full" (vedi paragrafo 2.4.4.4) viene visto come la sequenza esatta dei tre metodi appena descritti e viene implementato all'interno del Framework.
L'interfaccia "DMMInterface" è implementata dalle classi relative ai modelli (che abbiamo chiamato "SVM", "MLP", "MLPGA") come mostra la figura seguente:

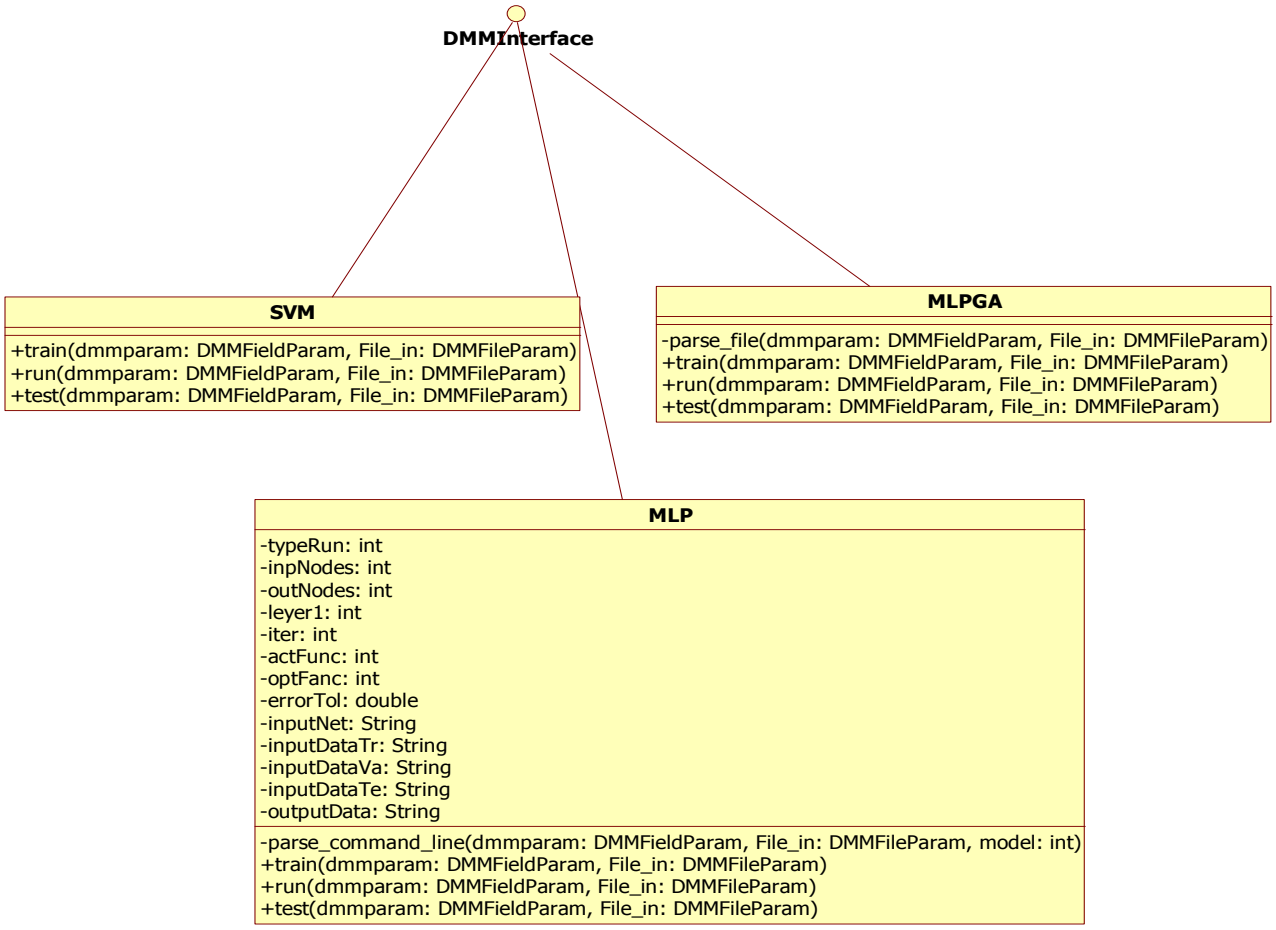


Fig. 33 UML del livello dei modelli

Chiaramente, con questa struttura, l'inserimento di un nuovo modello all'interno della suite risulta estremamente semplice e rapido in termini di tempo.

3.1.4. Il Pacchetto DMM: Diagramma UML

Di seguito viene riportato il diagramma UML (secondo lo standard UML 2.0) dell'intero pacchetto DMM, con l'introduzione di nuove classi non ancora descritte che hanno responsabilità ben precise:

- **Visualization**: ha al suo interno un metodo chiamato "plot2d" che prende in input un file in formato ASCII , e produce in output il suo grafico.
- **Statistics**: classe che si occupa di statistiche analizzando le colonne dei Dataset in input.

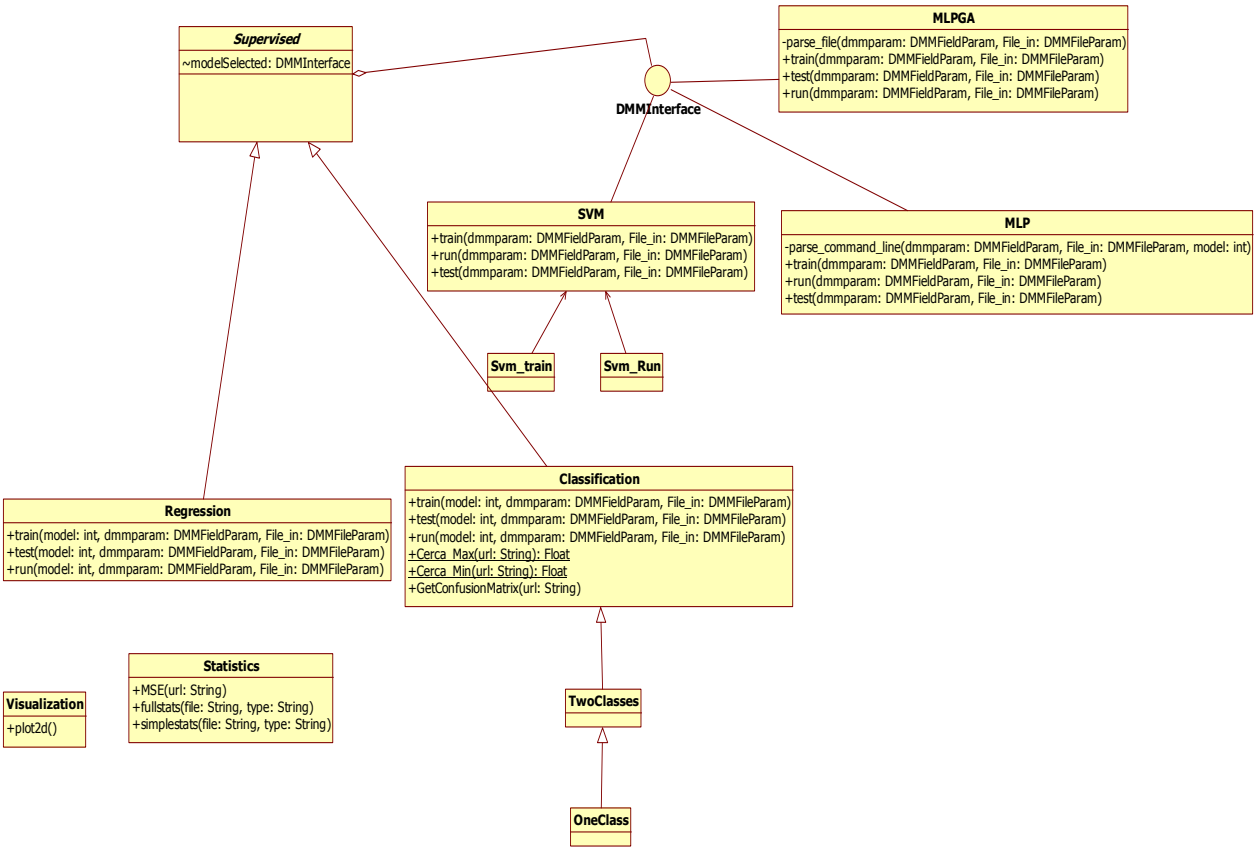


Fig. 34 Pacchetto DMM

3.2. Implementazione del componente DMM

Dopo la fase di progettazione del componente DMM, di seguito sarà descritta la fase di implementazione. Il linguaggio scelto, come già accennato, è Java 1.5 ed ho utilizzato l'ambiente di sviluppo Neatbeans in diverse versioni (prima 6.5, poi 6.7).

I motivi per cui è stato scelto tale linguaggio sono i seguenti:

- **Compatibilità multipiattaforma**: infatti grazie alla Java Virtual Machine le applicazioni Java sono compatibili a quasi tutte le piattaforme esistenti; tale caratteristica è fondamentale per la nostra web application perché, nei requisiti è compresa la possibilità di lavorare su diverse piattaforme computazionali (GRID/CLOUD e in generale HPC, High Performance Computing), sia in ambiente distribuito, sia in modalità Stand Alone(vedi paragrafo 2.3.4).
- **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, facendo risparmiare agli sviluppatori tempi di realizzazione.
- **Alta integrazione con il web**: altra caratteristica fondamentale per la nostra applicazione, dato che sarà installata su un server e sarà accessibile tramite web.
- **Sicurezza**: l'essere fortemente tipizzato e la gestione automatica della memoria, producono un codice più robusto, eliminando il rischio di operazioni dannose per il sistema come l'uso improprio di puntatori.

3.2.1. Implementazione livello funzionale

L'implementazione del livello funzionale riguarda la dichiarazione della classe astratta che generalizza tutte le funzionalità con supervisione, chiamata "Supervised" nonché della creazione delle due classi Classification e Regression con la responsabilità di fornire alla suite le funzionalità di classificazione e regressione, utilizzando i modelli messi a disposizione dal livello d'implementazione.

3.2.1.1. La classe Classification

La classe "Classification" ha la responsabilità di tutto ciò che riguarda la funzionalità di classificazione. Per fare ciò utilizza metodi e oggetti appartenenti alle classi presenti al livello di implementazione (SVM, MLP, MLPGA).
"Classification" ha il compito principale di effettuare le operazioni di addestramento, valutazione e utilizzo ("train", "test" "run") di tutti i modelli presenti nel livello di implementazione. Come visto nel diagramma UML del paragrafo 3.1.3.1. la classe "Classification" estende la classe astratta "Supervised", e di conseguenza, deve ridefinire i tre metodi presenti in essa. Per la semplicità di lettura, questa parte di codice della classe "Classification" viene proposto di seguito:

```
package org.dame.dmm;
import org.dame.dmm.DMMInterface.*;
public abstract class Supervised {

DMMInterface modelSelected;


}

public class Classification extends Supervised{



   public void train(int model,DMMFieldParam [] dmmparam, DMMFileParam [] File_in) throws
IOException{
     // the param model is a flag to select a model
     // 0 SVM
     // 1 MLP
     // 2 MLPGA

       if (model==0) {
          modelSelected=new SVM();
          modelSelected.train(dmmparam, File_in);
       }
       if (model==1) {
          modelSelected=new MLP();
          modelSelected.train(dmmparam, File_in);
       }

       if (model==2)  {
          modelSelected=new MLPGA();
          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 SVM
     // 1 MLP
     // 2 MLPGA

       if (model==0) {
          modelSelected=new SVM();
          modelSelected.test(dmmparam, File_in);
       }
       if (model==1) {
          modelSelected=new MLP();
          modelSelected.test(dmmparam, File_in);
       }

       if (model==2)  {
          modelSelected=new MLPGA();
          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 SVM
    // 1 MLP
    // 2 MLPGA

     if (model==0) {
         modelSelected=new SVM();
         modelSelected.run(dmmparam, File_in);
      }
      if (model==1) {
         modelSelected=new MLP();
         modelSelected.run(dmmparam, File_in);
      }

      if (model==2)  {
         modelSelected=new MLPGA();
         modelSelected.run(dmmparam, File_in);
      }


    }

…….
```

Come si può notare i modelli sono stati associati a degli interi in maniera sequenziale (0 per il modello SVM, 1 per MLP, 2 per MLPGA). Ciò è stato fatto, come si può intuire, per dare la possibilità al Framework (più precisamente alla sottocomponente creata apposta per la comunicazione con il DMM, chiamata "DMPlugin") di indicare al DMM quale modello, tra quelli a disposizione, sia stato selezionato dall'utente, e quindi di richiamare il metodo analogo al modello scelto.

Gli altri due vettori di input sono oggetti della componente "DmmParams" (vedi paragrafo 3.1.1.): il primo contiene tutti i parametri di configurazione del caso d'uso, l'altro contiene oggetti di tipo File di I/O come il Training e Test set; essi sono praticamente i parametri da dare in input alle librerie che implementano i modelli.

Oltre a questi tre metodi, la classe "Classification" fornisce un metodo per la rappresentazione dei risultati ottenuti dalla fase di "test" di un modello sulla base di un Dataset dato in input (vedi paragrafo 2.4.4.2). Questo metodo chiamato "GetConfusionMatrix" produce in uscita un file contenente la **matrice di confusione** basata sul risultato ottenuto dalla fase di "test". Ecco un esempio:

	A	B	C	Totale	
A	60	14	13	87	69,0%
B	15	34	11	60	56,7%
C	11	0	42	53	79,2%
Totale	86	48	66	200	68,0%

Fig. 35 Esempio di una matrice di confusione

L'elemento sulla riga *i* e sulla colonna *j* è il numero assoluto oppure la percentuale di casi della classe "vera" *i* che il classificatore ha classificato nella classe *j*.

Sulla diagonale principale ci sono i casi classificati correttamente, gli invece altri sono errori. Nel training set ci sono 200 casi. Nella classe A ci sono 87 casi:

- 60 classificati correttamente come A
- 27 classificati erroneamente, dei quali 14 come B e 13 come C

Sulla classe A l'accuratezza è 60 / 87 = 69,0%.

Sulla classe B è 34 / 60 = 56,7% e sulla classe C è 42 / 53 = 79,2%.

L'accuratezza complessiva è (60 + 34 + 42) / 200 = 136 / 200 = 68,0%.

Gli errori sono il 32%, cioè 64 casi su 200. Il valore di questa classificazione dipende non solo dalle percentuali, ma anche dal costo delle singole tipologie di errore. **Se C è la classe che è più importante classificare bene, ciò è positivo.**

Il metodo "GetConfusionMatrix" prende in input una stringa rappresentante l'URI del file su cui calcolare la matrice di confusione, e produrrà in output un file nominato con la stessa URI, aggiungendovi la stringa ".matrix" alla fine.

Questo metodo sarà utilizzato all'interno dei plugin di ogni modello e il suo utilizzo verrà spiegato meglio in uno dei prossimi paragrafi dove si parlerà dei plugin dei modelli.

Esempio del file prodotto dal metodo:

UriFile.matrix

```
0 line not employed

0       2
2       0
EFFICIENCY:0.0 (0/4)
COMPLETENESS OF CLASS 0:0.0 (0/2)
COMPLETENESS OF CLASS 1:0.0 (0/2)
```

3.2.1.2. La classe Regression

La classe “Regression”, ha la responsabilità di fornire alla suite DAME la funzionalità di regressione.

La classe “Regression”, per adesso, contiene esclusivamente, i metodi già visti all’interno della classe “Classification” ad eccezione del metodo “GetConfusionMatrix” che ovviamente è strettamente legato alla funzionalità di classificazione.

A supporto della funzionalità di regressione è stata creata un’altra classe chiamata “Visualization”, il quale contiene un metodo chiamato “plot2d”, che è in grado di dare in output un grafico di un qualsiasi file di dataset (opportunamente configurato) preso in input. Tale metodo utilizza la libreria STILTS[3] (Starlink Tables Infrastructure Library Tool Set) e supporta dati di input di tipo ASCII, dando la possibilità di scegliere l’estensione del file di output (png, gif, jpeg, eps, eps-gzip).

Ecco un esempio di output:

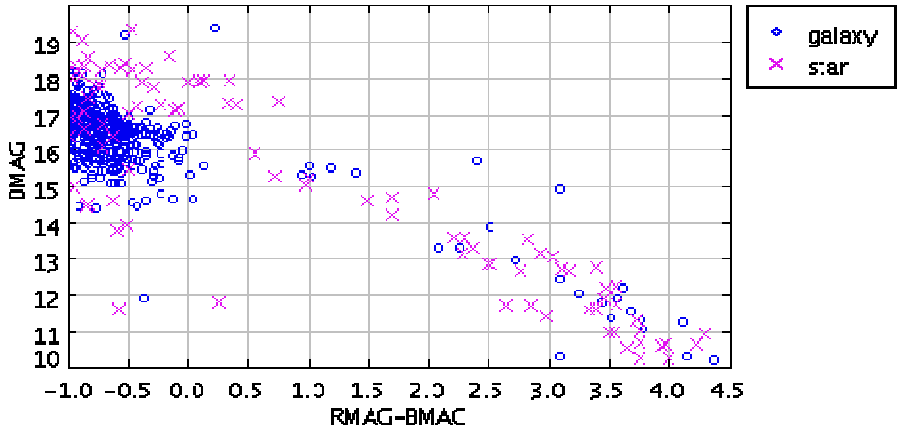


Fig. 36 Immagine di output utilizzando STILTS

3.2.2. Il Livello di implementazione: I modelli

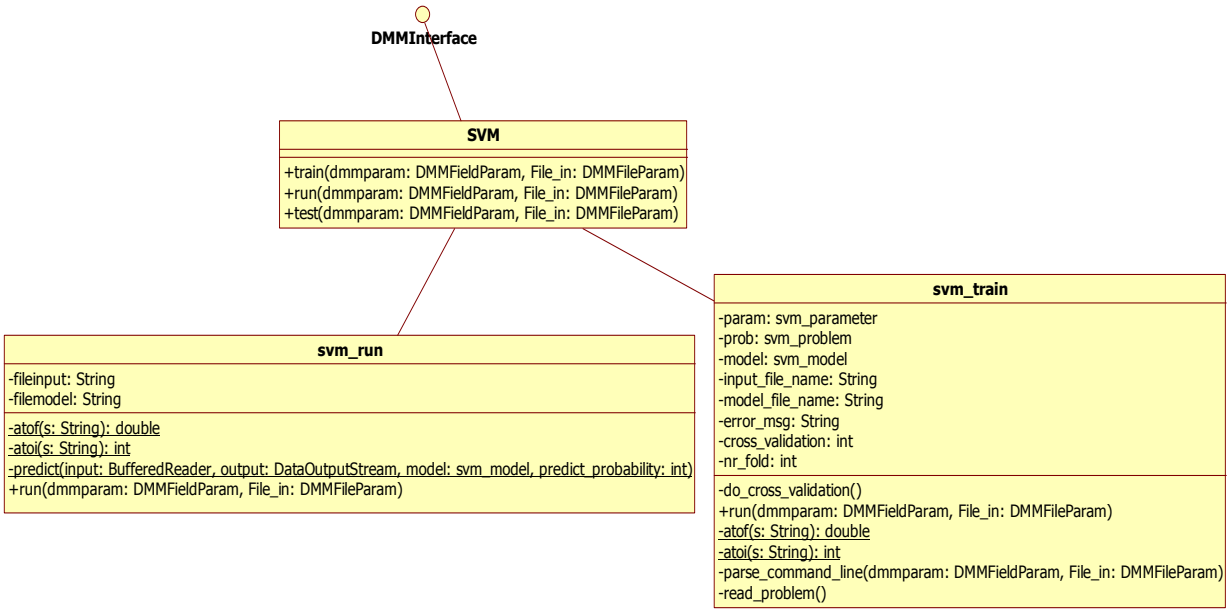
L’implementazione vera e propria dei modelli di apprendimento automatico consiste in un’interfaccia chiamata “DMMInterface” che generalizza tutti modelli con supervisione a disposizione della suite DAME. Tale interfaccia è implementata da tutte le classi che sono responsabili dei modelli, è cioè SVM, MLP e MLPGA; essa prevede l’implementazione dei tre modelli già descritti nel livello delle funzionalità, ovverossia la fase di addestramento, di test, e di utilizzo del modello stesso (train, test e run).

[3] Vedi anche www.star.bris.ac.uk/~mbt/stilts/

In questo caso però, i 3 metodi dovranno gestire tutti i parametri in input, configurandoli nella maniera più opportuna, in modo tale che possano essere utilizzabili dalle librerie.
Nei paragrafi che seguono è descritto dettagliatamente come vengono configurati questi i file e i paramrtri di I/O delle librerie utilizzate.
In futuro DAME avrà a disposizione tutti i modelli di apprendimento automatico, e quindi dovremmo creare una nuova interfaccia che generalizza tutti i modelli senza supervisione.

3.2.2.1. SVM

SVM è stato il primo modello che ho trattato. Esso utilizza la libreria Java libsvm-2.89[4] creata da Chih-Chung Chang and Chih-Jen Lin.
Come prima cosa ovviamente ho creato la classe SVM che implementa l'interfaccia DMMInterface:



In seguito ho implementato i tre metodi dichiarati nell'interfaccia con la seguente firma:

- public void train(DMMFieldParam [] dmmparam, DMMFileParam [] File_in) throws IOException
- public void test(DMMFieldParam [] dmmparam, DMMFileParam [] File_in) throws IOException
- public void run(DMMFieldParam [] dmmparam, DMMFileParam [] File_in) throws IOException

i tre metodi sopraelencati utilizzeranno oggetti e classi della libreria nella maniera più opportuna, configurando adeguatamente i parametri di input.
Per quanto riguarda la fase di addestramento, essa è identificata nel metodo chiamato "train", che a sua volta crea un'istanza della classe "svm_train", classe di supporto dedicata esclusivamente alla fase di addestramento del modello.

[4] Vedi anche http://www.csie.ntu.edu.tw/~cjlin/libsvm/

Per la sua semplicità, possiamo scrivere il codice relativo al metodo “train”:

```
public class SVM implements DMMInterface {

  public void train(DMMFieldParam [] dmmparam, DMMFileParam [] File_in) throws IOException{
          svm_train t = new svm_train();
          t.run(dmmparam,File_in);
    }
……
  }
```

Quindi per capire in che modo viene utilizzata la libreria è necessario analizzare meglio la classe chiamata svm_train.

Tale classe ha il compito di effettuare l’addestramento del modello utilizzando il dataset in input contenuto nel vettore di oggetti “DMMFileParam”, che a sua volta è un oggetto della classe DmmParams(vedi paragrafo 3.1.1).

La configurazione dei parametri viene fatta dal metodo “parse_command_line” scorrendo il vettore di oggetti “DMMFieldParam” chiamato dmmparam; questo vettore contiene i valori dei parametri del modello da settare, fondamentali per un addestramento corretto.

Ovviamente l’utente può decidere di non voler settare tutti i parametri, quindi i parametri non settati non verranno inseriti dal Framework nel vettore; ciò implica il fatto che i parametri non hanno un ordine ben preciso all’interno del vettore, perciò devono essere in qualche modo riconoscibili. Di conseguenza, i parametri sono stati etichettati appositamente per poterli riconoscere (l’etichettatura avviene al momento della definizione del Plugin di ogni modello, che sarà descritta nei prossimi paragrafi). Tale operazione di etichettatura avviene ovviamente anche all’interno del vettore contenente i file di I/O.

Vediamo un esempio di addestramento del modello SVM a partire da un Dataset molto banale:

ecco com’è strutturato un dataset che viene dato in input :

File Prova.txt

Nome Classe	N° Colonna :Valore

1	1:3.5 2:4 3:6
2	1:2.2 2:4.5 3:6.4

File Prova.txt.*model*

```
svm_type c_svc
kernel_type rbf
gamma 0.5
nr_class 2
total_sv 4
rho 0.0
label 1 0
probA 0.9460653745633817
probB -1.5426484195590797E-16
nr_sv 2 2
SV
1.0 1:1.0 2:0.0
1.0 1:0.0 2:1.0
-1.0 1:0.0 2:0.0
-1.0 1:1.0 2:1.0
```

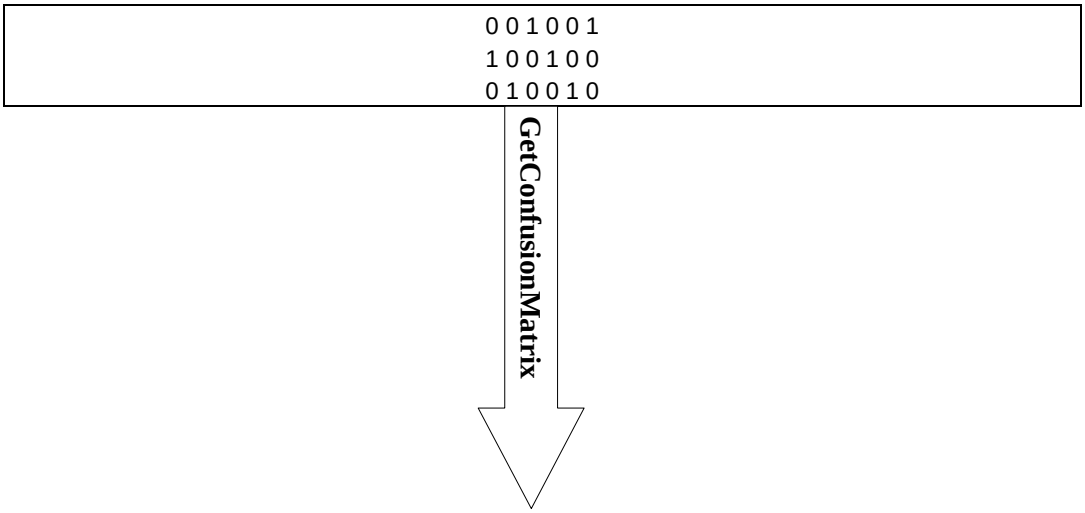
Come si può notare, dopo l'esecuzione del metodo "train" della classe SVM, viene creato un file con lo stesso nome del file di input e l'estensione *".model"*, che la libreria libsvm-2.89 è in grado di interpretare; il file viene costruito sulla base dei dati di input passati dalla componente Framework (cioè i valori dei parametri settati dall'utente ed il training set).

Questo file, serve per addestrare il modello, che deve essere in grado di classificare input di cui non si conosce l'output (vedi paragrafo 2.4.4). Il metodo **run**, infatti, utilizza il file con estensione *".model"* , come riferimento per poter effettuare classificazione (o regressione, dipende dall'input), ricevendo in input un file ASCII ; il risultato di tale metodo viene salvato all'interno di un file chiamato "UriDataset.run" dove UriDataset sta ovviamente per l'Uri del dataset dato in input.

Per quanto riguarda la fase di **test**, essa viene semplicemente implementata chiamando a sua volta il metodo "run" appena descritto, dando in input un dataset diverso da quello utilizzato per l'addestramento (trainset per il training, testset per il test); il metodo produrrà analogamente un file chiamato "UriTestset.test", dove però non è possibile apprezzare l'esito della fase di valutazione dato il tipo di formattazione del file. Per fare

ciò è possibile chiamare il metodo della matrice di confusione (“GetConfusioneMatrix”) utile proprio per poter visualizzare l’esito di una fase di valutazione di un modello.
Ecco un esempio:

Prova.txt.test

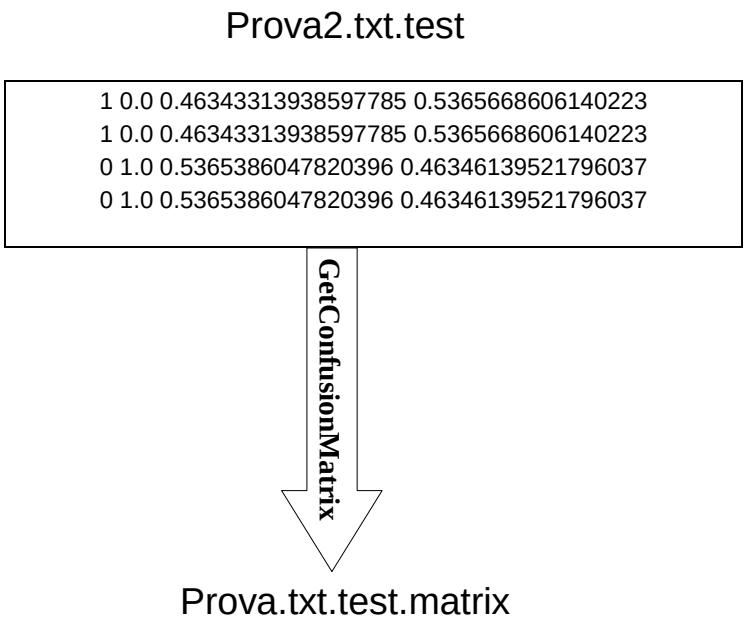


Prova.txt.test.matrix

```
0 line not employed

1       0       0
0       1       0
0       0       1
EFFICIENCY:1.0 (3/3)
COMPLETENESS OF CLASS 0:1.0 (1/1)
COMPLETENESS OF CLASS 1:1.0 (1/1)
COMPLETENESS OF CLASS 2:1.0 (1/1)
```

In questo caso il test ha avuto esito positivo poiché abbiamo un’efficienza del 100%; lo si può notare anche dalla matrice poiché vi sono tutti 1 sulla diagonale principale, che significa che nessun elemento è stato classificato male.




```
0 line not employed

0       2
2       0
EFFICIENCY:0.0 (0/4)
COMPLETENESS OF CLASS 0:0.0 (0/2)
COMPLETENESS OF CLASS 1:0.0 (0/2)
```

Come si può notare in questo caso la fase di "test" ha avuto un esito negativo poiché sulla diagonale principale abbiamo tutti zero. Questo accade quando il trainset e valori dei parametri non sono stati scelti con cura, oppure che sulla base dei dati a disposizione (trainset) non è possibile fare classificazione o regressione.

3.2.2.2. MLP

La procedura per l'inserimento del modello MLP nella suite Dame, risulta molto diversa rispetto a quella del modello SVM poiché la libreria che supporta questo modello è scritta nel linguaggio C++.

La libreria si chiama FANN (Fast Artificial Neural Network)[5] che appunto mette a disposizione l'implementazione del modello MLP.

Per inserirla all'interno della nostra suite per prima cosa è stata costruita la classe MLP di cui segue il Diagramma UML:

[5] Vedi anche http://leenissen.dk/fann/

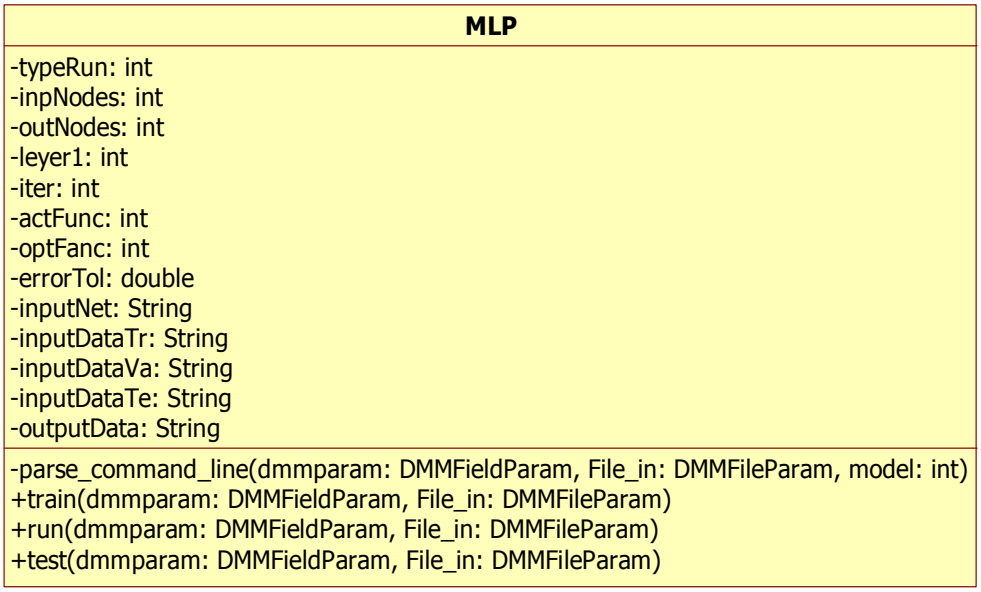


Fig. 37 UML della classe MLP

Come si può notare, oltre ai 3 metodi che, come già detto, deve implementare perché dichiarati all'interno della interfaccia "DMMInterface", utilizza un'ulteriore metodo chiamato "parse_command_line" che si occupa del settaggio di tutti i parametri del modello, iterando sul vettore di input "dmmparam", e inoltre, si occupa di passare al metodo della libreria le uri dei file di I/O.

Per utilizzare la libreria FANN scritta in C++, all'interno di una suite Java abbiamo bisogno di una classe che permetta l'esecuzione di un qualsiasi file, cioè come se lo dovessimo far avviare da riga di comando: ecco la porzione di codice relativa all'utilizzo di questa classe:

```
Runtime r=Runtime.getRuntime();
    try {
            parse_command_line(dmmparam,File_in,2);



            typeRun=1;
            String path = "/media/disk/DAME/DMM/src/MLP/FANNMLPSRC/fannmain";
            String cmd = path+" "+typeRun+" "+inpNodes+" "+outNodes+" "+leyer1+" "+iter+"
            "+actFunc+" "+optFanc+" "+errorTol+" "+inputNet+" "+inputDataTr+" "+inputDataVa+"
            "+inputDataTe+" "+outputData;

             Process p = r.exec(cmd);
……….
```

Come si può notare prima viene creato l'oggetto di tipo "Runtime",poi vengono settati i parametri attraverso il "metodo parse_command_line"; vengono settate la path in cui è situata la libreria, e la stringa "cmd" che è che il comando da dare per far eseguire la libreria tramite linea di comando. Si avvia l'esecuzione del metodo passando la stringa "cmd" al metodo "exec" dell'oggetto Runtime.
Di seguito viene illustrata una tabella con la descrizione di tutti i parametri da passare alla libreria:

Posizione	Nome variabile	Tipo	Descrizione
argv[1]	typeRun	Int	1 Training 2 Test 3 complete 4 run
Argv[2]….argv[9]	inpNodes, outNodes, layer1, iter, actFunc, optFunc, errorTol, inputNet.	Int Int Int Double String	Parametri relativi al settaggio del modello per il caso d'uso specifico.
Argv[10]	inputDataTr	String	Uri del trainset
Argv[11]	input DataVa	String	Uri del Validation set
Argv[12]	input DataTe	String	Uri del testset
Argv[13]	output Data	String	Nome del file di output (settato di default "out.txt")

3.2.2.3. MLPGA

L'ultimo modello da me trattato è l'MLPGA, che consiste in una variante del MLP. La libreria utilizzata si chiama "MlpGas"[6] ed è stata implementata da Massimo Brescia. Come per gli altri modelli, ho cominciato con la creazione della classe MLPGA:

[6] Vedi anche www.voneural.na.infn.it

Anche questa libreria essendo scritta in C++ necessita la stessa tipologia di inserimento nella suite DAME. Ma, a differenza dell'MLP, la libreria "MlpGas" necessita in input dei seguenti file che l'utilizzatore deve fornire (non tutti, dipende dal caso d'uso):

- Dataset di input

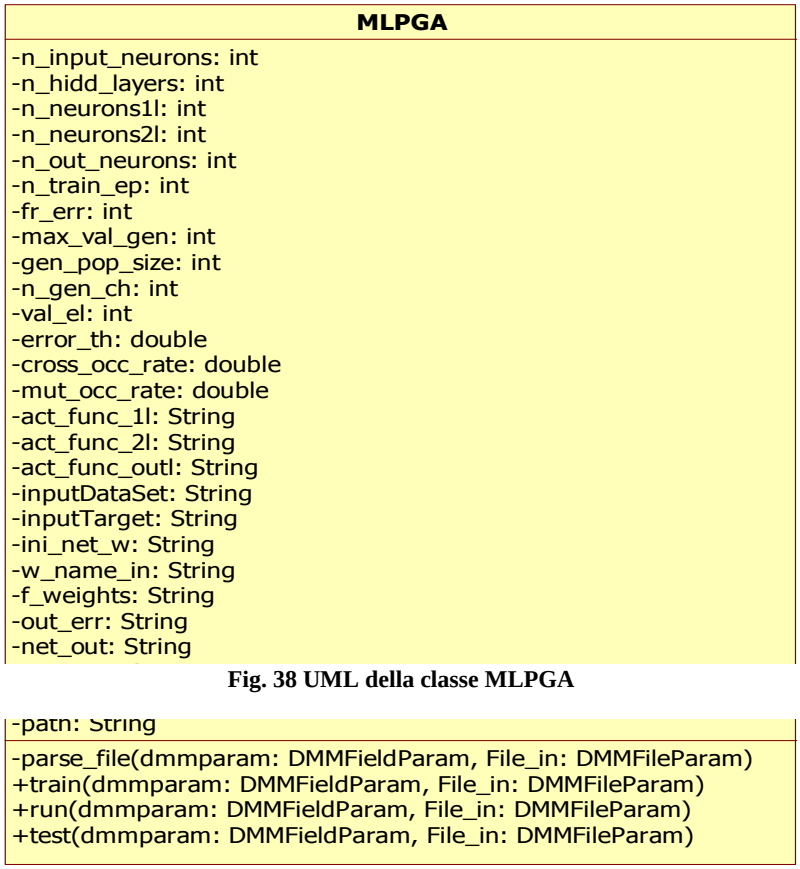


Fig. 38 UML della classe MLPGA

- File dei Target
- File di configurazione dei parametri
- File di configurazione per l'esperimento (train/test)
- File di configurazione per il caso d'uso (train/test/full)

Questi file verranno descritti nei prossimi paragrafi.

3.2.2.3.1. Dataset di input

Rappresenta il Dataset di input che può essere utilizzato nella fase di train e/o nella fase di test. Il file, come per tutte le librerie utilizzate deve essere un file ASCII con le colonne separate da spazi.

3.2.2.3.2. File dei Target

Il file dei Target è di tipo ASCII e viene utilizzato nella fase di apprendimento. Esso è richiesto dalla libreria solo nel caso in cui si voglia effettuare l'addestramento del modello sul Dataset di input.

3.2.2.3.3. File di configurazione dei parametri

Questo file contiene la configurazione dei parametri dell'architettura interna della rete neurale. Anche questo file è di tipo ASCII, ed è configurato con 8 righe, ognuna per ogni parametro del modello, e 1 colonna.

3.2.2.3.4. File di configurazione dell'esperimento

Questo file contiene la configurazione di ulteriori parametri relativi al caso d'uso scelto, oltre che alle path del Dataset di Input e del File dei Target. La sua configurazione cambia a seconda del caso d'uso da effettuare:

- **TRAIN:** in questo caso avremo un file con 16 righe e 1 colonna (16 parametri da configurare)
- **TEST:** in questo caso avremo un file con 3 righe e 1 colonna(3 parametri da configurare)

3.2.2.3.5. File di configurazione del caso d'uso

Questo file contiene i dati specifici del caso d'uso da effettuare, cioè quello che l'utente ha scelto tra il Train, il Test e il Full. In generale il file deve essere configurato in questo modo:
riga1:<String> Path del File di configurazione dei parametri
riga2:<String> esattamente la parola relativa al caso d'uso in maiuscolo (TRAIN, TEST, FULL)
riga3:<String>Path del file di configurazione dell'esperimento

3.2.2.3.6. File di output

La libreria produce in output a seconda del caso d'uso scelto i seguenti file di output:

- File di output della fase di train

- File dei pesi finali(output della fase di train, input della fase di test)
- File dei log errori della fase di test
- File di output della fase di test

Una volta noti i file di I/O della libreria, di seguito verrà descritto come configurare questi file nella maniera più opportuna in tutti i casi d'uso.
Come si può notare dal diagramma UML della classe, c'è un metodo privato chiamato "parse_file" che si occupa della configurazione del *"File di configurazione dei parametri"* , il quale, come già detto, è inserito in tutti i casi d'uso. Questo metodo, oltre a questa funzione ha un'ulteriore utilità fondamentale: dato che il *"File dei Target"* e il *"Dataset di input"* arrivano dal Framework in un unico file, il metodo "parse_file" si occupa di dividerli, avendo a disposizione alcuni parametri che indicano quante colonne deve avere il *"Dataset di input"*.
Gli altri 3 metodi si occupano della configurazione del *"File di configurazione dell'esperimento"* e il *"File di configurazione del caso d'uso"*, oltre che ovviamente di mandare in esecuzione la libreria MlpGas con lo stesso metodo visto per la classe MLP.
I parametri scelti dall'utente vengono ovviamente presi dal vettore dmmparam, usando praticamente la stessa tecnica di configurazione utilizzata per gli altri modelli, solo che in questo caso, li scrive all'interno di un file.
Ecco la porzione di codice relativa al lancio dell'eseguibile della libreria MlpGas:

```
Runtime r=Runtime.getRuntime();
Process pr=r.exec(path+"m 1");
```

dove "path" è una stringa che contiene la path nella quale è salvato il file eseguibile (chiamato "m") della libreria MlpGas. Come si può notare la libreria richiede un solo parametro (un intero) che rappresenta il caso d'uso da utilizzare (1 per il train, 2 per il test, 3 per il full). La fase di test e la fase di run sono implementate dallo stesso codice, quindi per poter effettuare la fase di run basta utilizzare il caso d'uso test con un Dataset che contiene valori di cui non si conosce l'output.
Supponendo di avere il file di input iniziale già diviso in due file chiamati input e target. Ecco un esempio di utilizzo:

Fase di Train

File di configurazione dei parametri (chiamato sample_NNParams)

35
1
10

4
10
activation_sigmoid
activation_sigmoid
activation_linear

File di configurazione dell'esperimento (Chiamato sample_train_conf)

input
target
RANDOM
none
0.1
30000
50
0.7
0.6
1
20
2
3
weights_saved_file
error_file
train_output_file

File di configurazione del caso d'uso:

sample_NNParams
TRAIN
sample_training_conf

Fase di Test

File di configurazione dell'esperimento (chiamato sample_test_conf)

input
weights_saved_file
test_output_file

File di configurazione del caso d'uso

sample_NNParams (stesso della fase di train)
TEST

sample_test_conf

I file di configurazione dell'esperimento devono avere un nome preciso:

- TRAIN: "train_experiment_configuration.txt"
- TEST: "test_experiment_configuration.txt"
- FULL: "full_experiment_configuration.txt"

Inoltre questi file devono essere memorizzati nella stessa path dell'eseguibile della libreria MlpGas.

3.2.3. I Plugin dei modelli implementati

La componente DMM comunica esclusivamente con il Framework attraverso un sottocomponente chiamato DMPlugin. Come già accennato questo componente permette, oltre alla comunicazione tra DMM e Framework, di dare la possibilità a programmatori esterni di poter aggiungere modelli alla suite DAME, avendo a disposizione una GUI molto semplice da utilizzare:

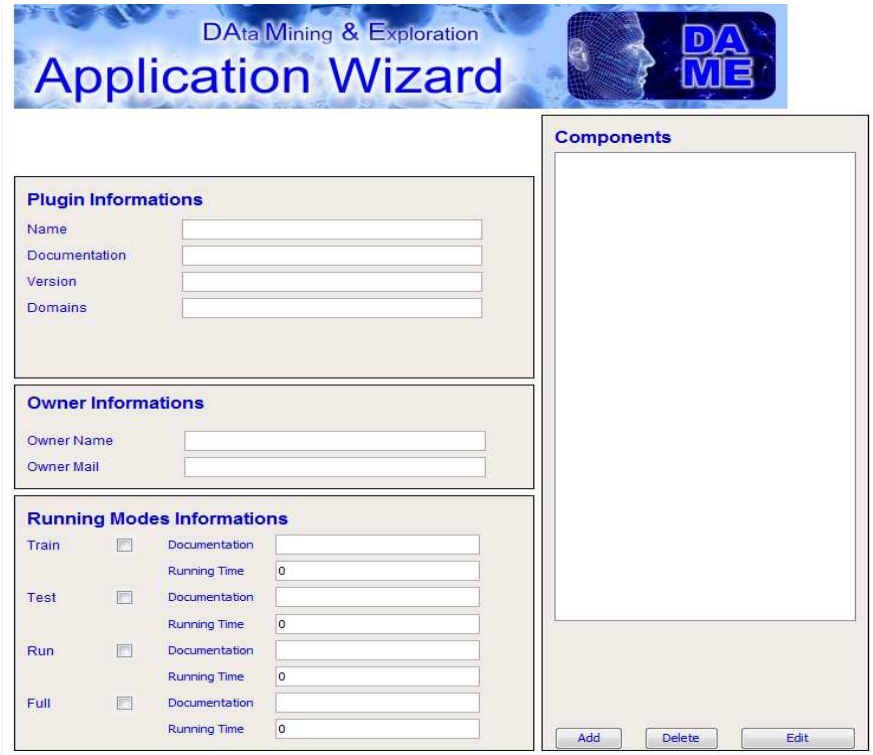


Fig. 39 GUI DMPlugin

Come già ampiamente descritto i modelli dovranno essere utilizzati per fornire alla suite le funzionalità di classificazione e regressione. Per poter fare ciò, bisogna creare delle classi che comunichino con la componente DMM, ottenendo così i file di output desiderati dall'utente. Ecco il diagramma UML del Pacchetto del "DMPlugin":

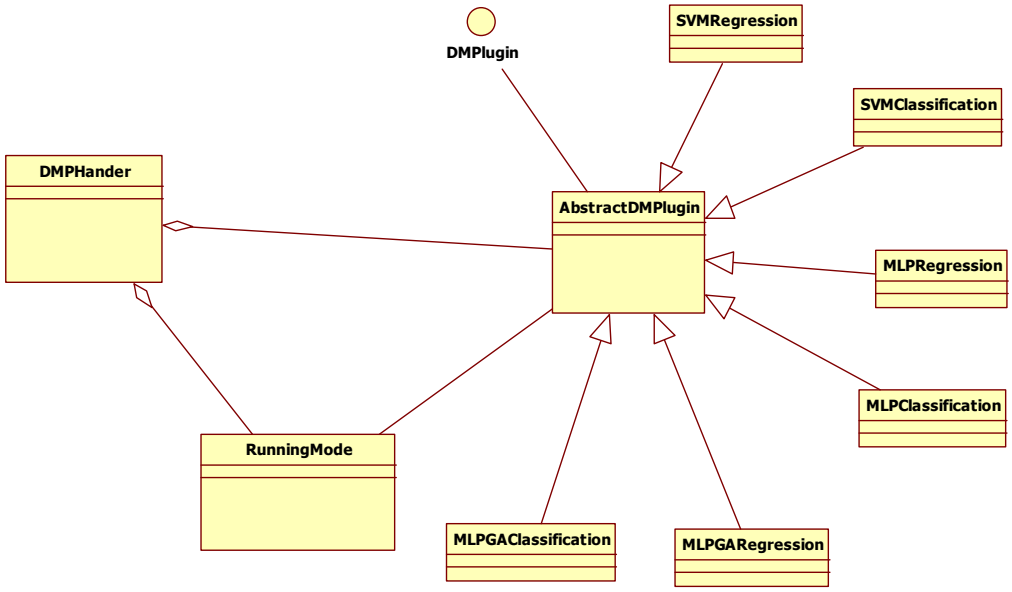


Fig. 40 UML del DMPlugin

Il mio compito è stato quello di progettare ed implementare le sei classi che specializzano la classe "AbStractDMPlugin" come mostra il diagramma UML sopra riportato.
Per prima cosa, dobbiamo compilare in maniera opportuna tutti i campi della GUI del DMPlugin mostrata in fig 35:

- **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 relativa al plugin
- **Version:** versione del plugin, se è così il plugin precedente viene sovrascritto.
- **Domains:** indica quale funzionalità fornire al plugin (classificazione o regressione)
- **Owner Name:** nome del proprietario
- **Owner Mail:** e-mail del proprietario

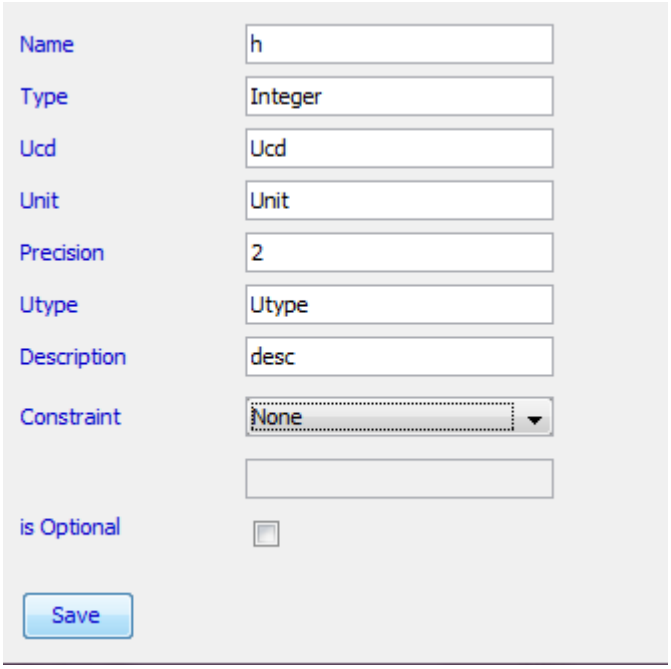
Dopo aver riempito questi campi bisogna riempire i campi sotto la sezione "Running Modes Information", che richiede quale caso d'uso (train, test, run, full) il plugin implementerà. Nel mio caso ovviamente tutte e 4, e dopo aver spuntato tutte le caselle e

inserito anche qui le url delle documentazioni relative, avremo una situazione simile alla figura che segue:

L'esempio ovviamente si riferisce al plugin relativo al modello MLPGA che fornisce la funzionalità di classificazione.

Come si può notare sono comparse 4 icone nel riquadro a destra sotto la scritta "Components"; esse sono relative ad ogni caso d'uso e riguardano, la definizione di tutti i modelli, e i file di I/O che servono per ogni caso d'uso.

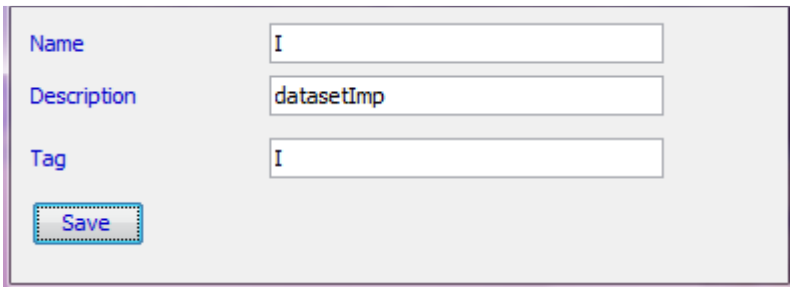
Selezionando la voce "Fields" e cliccando su bottone "Add" è possibile definire tutti parametri relativi al modello MLPGA.



La figura mostra il form relativo all'aggiunta di un parametro con alcuni parametri importanti come:

- **Name:** è l'etichetta del parametro, attraverso la quale il parametro si rende riconoscibile alla classe MLPGA: per essere più precisi la classe MLPGA (come gli altri due modelli) analizza la prima lettera del Name, perciò almeno la prima lettera del campo deve essere univoca.
- **Type:** tipo del parametro
- **Ucd, Unit, Precision, Utype, Description:** campi relativi al parametro che possono essere utili successivamente la creazione del plugin.

Per quanto riguarda i file di I/O la procedura è molto simile:



Dopo aver completato l'inserimento di tutti i parametri e di tutti i file del modello, si preme il bottone "Generate Code" posto sul menù a tendina "File":

A questo punto l'applicazione crea una classe Java all'interno del pacchetto DMPlugin in cui è stato automaticamente generato il codice relativo al costruttore della classe, ed il mio compito era quello di implementare i 4 metodi relativi ai 4 casi d'uso:

- **trainRun**
- **testRun**
- **runRun**
- **FullRun**

Le funzionalità principali di questi metodi sono:

- **creare oggetti della classe Classification o Regression** richiamando a loro volta i metodi che tali classi mettono a disposizione. Infatti l'applicazione genera automaticamente anche i due vettori da dare in input a tutti i metodi implementati dalle due classi.
- **spostare i file di output di ogni caso d'uso** dalla cartella di esecuzione alla cartella dell'utente relativa alla sessione dell'esperimento tramite una semplice invocazione di un metodo del Framework la cui firma completa è: "reportStatus(DMMOutputFileParam file, String status)". Come si può facilmente intuire, esso prende in input il file di output dell'esperimento e lo trasferirà nella cartella utente relativa all'esperimento appena eseguito;
- **Fornire ulteriori servizi all'utente** come la rappresentazione tramite matrice di confusione (nel caso d'uso del test della classificazione) oppure plottare la funzione risultante con il metodo "plot2d" della classe Regression (nel caso d'uso del test della Regressione).

Nelle sei classe implementate cambia ovviamente la compilazione del form iniziale e alcuni piccoli accorgimenti sul nome dei file di output.

4. Il testing della componente DMM

In questo capitolo verrà descritta la fase di testing della componente DMM con le strategie adottate. Attualmente lo Unit test della componente DMM si può ritenere completo poiché è già nella sua versione finale.
Il testing è stato effettuato nell'ambiente di sviluppo Neatbeans 6.5 con la libreria JUnit.

4.1. Panoramica della componente DMM

Per la componente DMM ho focalizzato l'attenzione sulla verifica della corretta fornitura alla suite DAME delle funzionalità di Classificazione e Regressione. Questo significa che devono essere testate principalmente le due classi "Classification" e "Regression", ma siccome al momento dal punto di vista di implementazione la classe "Regression" è quasi uguale alla classe "Classification", ci siamo soffermati sul testing di quest'ultima.
La classe "Classification" contiene i più importanti metodi del pacchetto DMM:

- Train
- Test
- Run

Gli input di questi metodi sono:

- Etichetta del modello (0=SVM, 1=MLP, 2=MLPGA)
- Valori dei parametri del modello
- File di Input/Output

La classe "Classification" contiene inoltre il metodo "getConfusionMatrix" che fornisce in output la matrice di confusione della matrice data in input in formato ASCII.
Oltre a questa classe viene testata anche la classe "Statistics" che contiene alcuni metodi che effettuano delle statistiche sulle colonne dei dataset in input.

4.2. Caratteristiche da testare/escludere

Le caratteristiche principali da testare sono:

- **Correttezza dei file di output**
- **Fault Tolerance:** può capitare in certi casi che vi siano errori nella dei file di I/O (path inesistente, file non trovato, etc….) che restituiscano dei fault, ed è per

questo che deve essere testata anche la gestione delle eccezioni, fondamentale per far capire al Framework il tipo di errore occorso.

4.3. Strategie di testing

La componente è stata testata con una strategia Black-Box, verificando che la componente rispetti i requisiti funzionali.
Gli input del metodo "getConfusionMatrix" sono stati divisi in classi di equivalenza nel modo seguente:

- Numero di colonne della matrice di input:
  - N° di colonne <= 2
  - N° di colonne > 2
- Grado di efficienza previsto della matrice di output
  - Efficienza= 1.0 (100%)
  - Efficienza= 0.0 (0%)
  - 0.01 (1%) <= Efficienza <= 0.99 (99%)

Queste classi di equivalenza sono state create sulla base dell'algoritmo della matrice di confusione e verranno combinate tramite la tecnica Worst Case Test (WCT)

4.3.1. Classi e metodi da testare

Di seguito vengono elencate le classi e i relativi metodi da testare:

- Classification:
  - getConfusionMatrix(String uri_matrice)
  - CercaMin(String uri_matrice)
  - CercaMax(String uri_matrice)
- Statistics:
  - fullStats(String file, String type)
  - simpleStats(String file, String type)

4.4. Test Case

Di seguito vengono riportati tutti i casi di test effettuati.

TEST ID	1	
TEST NAME	testCercaMax100	
TEST DESCRIPTION	Test del metodo CercaMax in cui nella matrice di input il massimo valore è 100	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Matrice con valori positivi e negativi in con massimo 100	100	Ok

TEST ID	2	
TEST NAME	testCercaMax0	
TEST DESCRIPTION	Test del metodo CercaMax in cui nella matrice di input il massimo valore è 0	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Matrice con soli zeri o negativi in cui il massimo è 0	**0**	Ok

TEST ID	3	
TEST NAME	testCercaMin0	
TEST DESCRIPTION	Test del metodo CercaMin in cui nella matrice di input il minimo valore è 0	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Matrice con soli valori positivi in cui il minimo è 0	**0**	Ok

TEST ID	4	
TEST NAME	TestCercaMinneg	
TEST DESCRIPTION	Test del metodo CercaMin in cui nella matrice di input il minimo valore è un numero negativo	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**

Matrice con soli numeri positivi o negativi (interi o double) in cui il minimo è negativo	Min negativo	Ok

TEST ID	5	
TEST NAME	testgetConfusionMatrixEff100col2	
TEST DESCRIPTION	Test del metodo getConfusionMatrix in cui nella matrice di input il massimo ci sono 2 colonne e l'efficienza prevista dall'oracolo in output è del 100%	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Matrice con interi e double di due colonne	Efficienza 100%	Ok

TEST ID	6	
TEST NAME	testgetConfusionMatrixEff0col2	
TEST DESCRIPTION	Test del metodo getConfusionMatrix in cui nella matrice di input il massimo ci sono 2 colonne e l'efficienza prevista dall'oracolo in output è del 0%	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Matrice con interi e double di due colonne	Efficienza 0%	Ok

TEST ID	7
TEST NAME	testgetConfusionMatrixEff100colmagg2

TEST DESCRIPTION	Test del metodo getConfusionMatrix in cui nella matrice di input il massimo ci sono più di 2 colonne e l'efficienza prevista dall'oracolo in output è del 100%	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Matrice con interi e double con più di due colonne	Efficienza 100%	Ok

TEST ID	8	
TEST NAME	testgetConfusionMatrixEff0colmagg2	
TEST DESCRIPTION	Test del metodo getConfusionMatrix in cui nella matrice di input il massimo ci sono più di 2 colonne e l'efficienza prevista dall'oracolo in output è del 0%	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Matrice con interi e double con più di due colonne	Efficienza 0%	Ok

TEST ID	9	
TEST NAME	testgetConfusionMatrixEffcolmagg2	
TEST DESCRIPTION	Test del metodo getConfusionMatrix in cui nella matrice di input il massimo ci sono più di 2 colonne	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Matrice con interi e double con più di due colonne	86%	Ok

TEST ID	10

TEST NAME	TestTrainSVM	
TEST DESCRIPTION	Test del metodo Train utilizzando il modello SVM	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Tutti i parametri	Ok	Ok
Nessun Parametro	Ok	Ok

TEST ID	11	
TEST NAME	TestRunSVM	
TEST DESCRIPTION	Test del metodo Run utilizzando il modello SVM	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Tutti i parametri	Ok	Ok
Nessun parametro	Ok	Ok

TEST ID	12	
TEST NAME	TestTestSVM	
TEST DESCRIPTION	Test del metodo Test utilizzando il modello SVM	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Tutti i parametri	Ok	Ok
Nessun parametro	Ok	Ok

TEST ID	13	
TEST NAME	TestTrainMLP	
TEST DESCRIPTION	Test del metodo Train utilizzando il modello MLP	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Tutti i parametri	Ok	Ok
Nessun parametro	Ok	Ok

TEST ID	14

TEST NAME	TestRunMLP	
TEST DESCRIPTION	Test del metodo Run utilizzando il modello MLP	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Tutti i parametri	Ok	Ok
Nessun parametro	Ok	Ok

TEST ID	15	
TEST NAME	TestTestMLP	
TEST DESCRIPTION	Test del metodo Test utilizzando il modello MLP	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Tutti i parametri	Ok	Ok
Nessun parametro	Ok	Ok

TEST ID	16	
TEST NAME	TestTrainMLPGA	
TEST DESCRIPTION	Test del metodo Train utilizzando il modello MLPGA	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Tutti i parametri	Ok	Ok
Nessun parametro	Ok	Ok

TEST ID	17	
TEST NAME	TestRunMLPGA	
TEST DESCRIPTION	Test del metodo Run utilizzando il modello MLPGA	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Tutti i parametri	Ok	Ok
Nessun parametro	Ok	Ok

TEST ID	18	
TEST NAME	TestTestMLPGA	
TEST DESCRIPTION	Test del metodo Test utilizzando il modello MLPGA	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
Tutti i parametri	Ok	Ok
Nessun parametro	Ok	Ok

TEST ID	19	
TEST NAME	Testsimplestats	
TEST DESCRIPTION	Test del metodo simplestast nella classe Statistics	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
File ASCII	Ok	Ok

TEST ID	20	
TEST NAME	TestFullStats	
TEST DESCRIPTION	Test del metodo fullstats nella classe Statistics	
INPUT	**OBTAINED OUTPUT**	**MATCH WITH ORACLE**
File di dimensione superiore ai 500 MB	OK	OK
File di dimensione inferiore ai 500 MB	Ok	Ok

5. Bibliografia

- Progetto S.C.O.P.E. - http://www.scope.unina.it

- Progetto Vo-Neural \ Dame - http://voneural.na.infn.it

- Chih-Wei Hsu, Chih-Chung Chang e Chih-Jen Lin*: A Practical Guide to Support Vector Classification(2007)*

- Java - http://java.sun.com

- IvoaVOTable - http://www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/IvoaVOTable