I N D I C E

CAPITOLO 1

INTRODUZIONE AL DATA MINING

La società contemporanea è caratterizzata dalla disponibilità di una gran quantità di dati: il crescente e sempre più pervasivo utilizzo dei computer e di Internet nella vita e nell'attività degli individui e delle organizzazioni (scientifiche, industriali, commerciali), e la disponibilità di memorie di massa di dimensioni crescenti a prezzi sempre più bassi, ha portato ad un accumulo crescente dei dati in file, database e data warehouse. E' stato stimato che ogni 20 mesi raddoppia la quantità di dati accumulati.

Ma i semplici dati, senza alcuna interpretazione, raramente possono essere di aiuto: ciò che realmente occorre è l'informazione nascosta nei dati.

Le enormi quantità di dati in nostro possesso non sono di utilizzo immediato, ma richiedono un lavoro di estrazione delle informazioni: i dati sono il materiale grezzo da cui si estrae l'informazione.

L'abilità di analizzare e sfruttare database enormi è in ritardo rispetto l'abilità di raccogliere e memorizzare i dati. Recentemente però la ricerca, particolarmente nei settori dell'informatica e della statistica, ha condotto allo sviluppo di procedure flessibili e scalabili per l'analisi di grandi basi di dati.

Tradizionalmente l'analisi dei dati era un processo “manuale” e l’analista doveva avere familiarità sia con i dati sia con i metodi della statistica, mentre l'elaboratore elettronico era solo un sostegno per il calcolo. Ma l'enorme crescita degli ar-

chivi di dati (sia nel numero dei record che nel numero di attributi contenuti in un database) ha reso completamente impraticabile questo tipo di analisi dei dati. Di conseguenza si è costituita ed é cresciuta costantemente una comunità di ricercatori e professionisti interessati al problema dell'analisi automatica di grandi quantità di dati, denominata "Knowledge Discovery in Databases (KDD) " (scoperta di conoscenza nei database). La prima serie di incontri con argomento il KDD si è tenuta nel 1989 e da allora si sono succedute diverse altre conferenze. Il termine KDD viene utilizzato per descrivere l'intero processo di *estrazione della conoscenza da un database*: dall'individuazione degli obiettivi fino all'applicazione delle informazioni trovate. Il problema base cui si rivolge il processo di KDD è quello di trasformare i dati grezzi in altre forme che possono essere più compatte, più astratte o più utili. Al centro di tale processo c'è l'applicazione dei metodi di *data mining* (estrazione dei dati), che utilizza raffinate analisi statistiche e tecniche di modellazione per la scoperta e l'estrazione di pattern e relazioni nascoste nei database: i dati descrivono un insieme di fatti, e il *pattern* è un'espressione, in un qualche linguaggio, che descrive un sottoinsieme dei dati (o un modello applicabile a questo sottoinsieme), le associazioni tra essi, le sequenze ripetute o le regolarità nascoste nei dati; in altre parole un pattern indica una rappresentazione sintetica e ad alto livello di un sottoinsieme dei dati.

Quindi, il termine data mining viene impiegato per descrivere la fase del processo di KDD nel quale gli algoritmi di apprendimento automatico (*machine learning*) vengono applicati ai dati. Infatti, molte delle metodologie impiegate nel data mining traggono origine principalmente da due filoni di ricerca: quello sviluppato dalla comunità scientifica dell'apprendimento automatico e quello sviluppato dagli statistici.

La formalizzazione di questa terminologia è stata avanzata per la prima volta da Usama Fayyad nei lavori della "First International Conference on Knowledge Discovery and data mining" a Montreal nel 1995, per indicare un insieme integrato di tecniche di analisi, ripartite in varie fasi procedurali, volte a estrarre conoscenze non note a priori da grandi masse di dati, apparentemente non contenenti regolarità o relazioni importanti.

La natura del KDD è fondamentalmente interdisciplinare: esso è nato e si evolve dall'incrocio di numerose ricerche in diversi campi di studio come machine learning, riconoscimento dei pattern, statistica, intelligenza artificiale, ragionamento sotto incertezza, sistemi esperti per acquisizione di conoscenza, visualizzazione dei dati, machine discovery, scientific discovery, recupero di informazione, elaborazione ad alte prestazioni e chiaramente anche dall'ambito della gestione dei database: i sistemi software per KDD inglobano teorie, algoritmi e metodi estratti da tutti questi campi. Le teorie e gli strumenti per la gestione dei database forniscono le infrastrutture necessarie per memorizzare, accedere e manipolare i dati.

Le aree di ricerca che si occupano dell'inferenza di modelli dai dati, come il riconoscimento statistico dei pattern, la statistica applicata, machine learning e reti neurali, sono state la spinta per molti dei precedenti lavori di scoperta di conoscenza. Il processo di KDD si fonda ampiamente sui metodi di queste aree, soprattutto nel passo di Data Mining, ma differisce da esse perché mira all'intero processo di scoperta di conoscenza, si preoccupa cioè anche di come avviene la memorizzazione e l'accesso ai dati, di come gli algoritmi possano essere scalati per trattare insieme massivi di dati, di come interpretare e visualizzare i risultati, eccetera.

La statistica fornisce al processo di KDD il linguaggio e gli strumenti per quanti-

ficare l'incertezza nell'inferenza di pattern di valore generale estratti da campioni particolari della popolazione. Il KDD è un modello di più ampio respiro rispetto alla statistica poiché fornisce gli strumenti per l'automatizzazione dell'intero processo di analisi dei dati.

1.1) Il processo di KDD

La scoperta di conoscenza è un processo che richiede un certo numero di passi, necessari per assicurare l'efficace applicazione del data mining. Non è possibile sperare di utilizzare un algoritmo per il data mining, come le reti neurali o gli alberi di decisione, direttamente sui dati e pretendere di ottenere dei risultati significativi.

Il processo di KDD può definirsi come:

> *Il processo non banale di identificazione di validi, originali, potenzialmente utili e in definitiva comprensibili pattern nei dati.* [03]

Analizzando tale definizione, si evince che i pattern ricavati dovrebbero essere:

- *validi*: il loro valore e la loro efficacia dovrebbe restare inalterata, o quasi, per nuovi dati, ossia per dati diversi da quelli che si sono utilizzati per ricavare i pattern in questione;
- *originali:* dovrebbero contenere qualche elemento innovativo, ossia dovrebbero fornire nuove informazioni, o un nuovo modo di vedere e utilizzare informazioni già note;
- *potenzialmente utili*: dovrebbero fornire risultati e informazioni non ovvie e utili nel campo di applicazione interessato dall'analisi;
- *comprensibili*: i risultati e le informazioni ottenute dall'analisi dovrebbero es-

sere fruibili con chiarezza anche da utenti non esperti in statistica ed informatica (se non immediatamente, almeno dopo qualche post-elaborazione).

Il processo di KDD è iterativo e interattivo, con molte decisioni prese dall'utente, e comporta l'esecuzione di molti passi raggruppabili in pochi tipi di operazioni fondamentali: la preparazione di dati, la ricerca dei pattern, la valutazione della conoscenza, i raffinamenti; tutte queste operazioni vengono iterate un certo numero di volte.

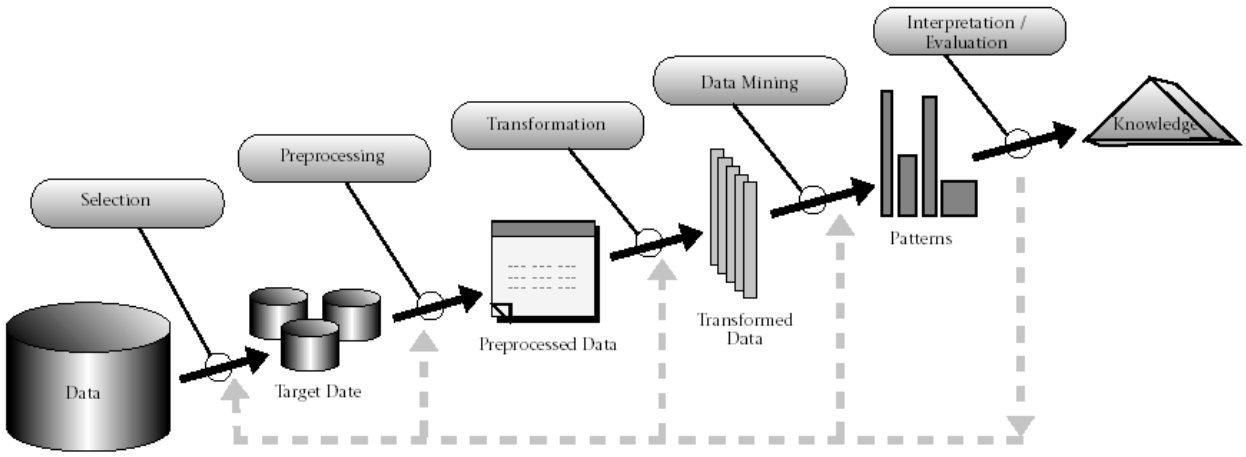


Figura 1.1 Il processo di Knowledge Discovery in Databases (KDD)

I passi fondamentali del processo di scoperta di conoscenza dai database (Knowledge Discovery in Databases, KDD) sono:

A) Identificazione del problema

Per il miglior uso del data mining bisogna saper esprimere chiaramente i propri obiettivi. Infatti una cattiva descrizione del problema rallenta lo sviluppo dell'intero processo e può causare una cattiva scelta del modello da utilizzare. Prerequisito fondamentale è l'avere una buona conoscenza dei dati e della realtà

che essi descrivono. Senza questo background, anche se (fortunatamente) si ottengono buoni risultati, non si comprenderebbe pienamente il loro significato, né si saprebbe come sfruttarli al meglio.

B) Preparazione dei dati

Spesso questo è il passo più impegnativo: la preparazione dei dati prende di solito tra il 50% e l'80% del tempo e degli sforzi dell'intero processo di KDD. Solitamente in questo passo possono distinguersi 5 fasi differenti, non sempre tutte necessarie:

B.1) Raccolta

Bisogna individuare le sorgenti dei dati che si vogliono analizzare. Per di più è possibile che alcuni dati di cui si ha bisogno non siano mai stati collezionati, per cui bisogna procedere ex novo alla raccolta.

B.2) Accertamento

Anche il data mining, come gran parte delle tecniche che lavorano sui dati, è soggetto alla regola del GIGO (Garbage In, Garbage Out), cioè se i dati in ingresso sono scorretti, in uscita si ritroveranno risultati scorretti. Una ispezione dei dati ne identifica le caratteristiche che principalmente influiranno sulla qualità dell'analisi.

I dati di cui si ha bisogno risiedono in uno o più database: le sorgenti possono trovarsi nel database dell'organizzazione, nel data warehouse o in data mart realizzati per scopi specifici, oppure in altri sistemi esterni. Nasce quindi il problema di consolidare i dati in un singolo database, per cui si dovranno affrontare i tipici inconvenienti di tale operazione: le inconsistenti definizioni dei dati, le differenti codifiche dei dati, eccetera.

Anche quando i dati provengono da un solo database bisogna ugualmente esaminarli, per individuare eventuali problemi come la mancanza di alcuni dati o la presenza di valori che violano i vincoli di integrità. Talvolta è necessario effettuare ulteriori verifiche, ad esempio sulla provenienza dei dati, sul tipo di memorizzazione, sulle unita di misura usate, su come e quando i dati sono stati raccolti e sotto quali condizioni, ed altro ancora.

In pratica bisogna assicurarsi, nel miglior modo possibile, che tutti i dati misurino la stessa cosa e nella stessa maniera.

B.3) Consolidamento e Pulizia

É la fase nella quale si costruisce il database dal quale prelevare i dati da elaborare. Occorre consolidare i dati e rimediare, per quanto possibile, ai problemi che si sono identificati nella fase precedente, evitando però di ricorrere a soluzioni superficiali che potrebbero addirittura peggiorare le cose. Ad esempio ignorare tuple con rilevanti mancanze di dati può portare alla costruzione di modelli di predizione non ottimi o addirittura falsi, in quanto si basano su dati che in parte non hanno significato.

B.4) Selezione

Una volta raccolti e consolidati i dati, si passa alla selezione di quelli di interesse per il modello che si vuole costruire. Ad esempio, per i modelli predittivi si separano le variabili (colonne) indipendenti da quelle dipendenti, e si scelgono le tuple (righe) da utilizzare per il *training* (addestramento) del modello.

La conoscenza del dominio del problema permette di selezionare i dati correttamente. Gli strumenti grafici che aiutano a visualizzare i dati e le loro relazioni sono molto utili a questo punto dell'analisi: ad esempio aiutano

nell'identificazione di variabili indipendenti importanti e rivelano variabili che sono correlate. Per quanto riguarda la presenza di eventuali valori anomali (outliers), in qualche caso essi contengono informazioni importanti (anche per la costruzione del modello), ma spesso possono o devono essere ignorati: ad esempio possono essere il risultato di uno scorretto inserimento dei dati oppure la registrazione di eventi che accadono molto raramente.

Spesso quando il database è molto grande è necessario campionare i dati. In tal caso è importante che la campionatura sia fatta in modo rigoroso, per assicurare l'effettiva casualità della selezione, in modo da evitare che si abbia una grossa perdita di informazione. Di fronte alla scelta di analizzare pochi modelli costruiti su tutti i dati oppure analizzare tanti modelli ma costruiti su dati campionati, quest'ultimo approccio di solito è il migliore, e consente la realizzazione di modelli più robusti e accurati.

B.5) Trasformazione

Dopo aver selezionato i dati si potrebbe presentare l'esigenza di operare ulteriori trasformazioni su di essi. Ad esempio per utilizzare grandezze derivate (come il rapporto tra due variabili), per normalizzare qualche grandezza, per discretizzarla, oppure per trasformare una variabile categorica in una variabile dicotomica multipla.

C) Costruzione e validazione del modello

È fondamentalmente un processo iterativo: occorre esaminare modelli alternativi per trovare quello migliore per il problema da risolvere. Tale ricerca comporta spesso la riconsiderazione dei passi precedenti, ad esempio modificando

la selezione dei dati o anche la descrizione del problema stesso.

La scelta di quale metodo utilizzare nella fase di analisi dipende essenzialmente dal tipo di problema oggetto di studio e dal tipo di dati disponibili per l'analisi. Il data mining è un processo guidato dalle applicazioni, per cui i metodi utilizzati possono essere classificati in base allo scopo per il quale l'analisi viene effettuata. In conformità a tale criterio si possono distinguere essenzialmente quattro grandi classi di metodologie: [01]

- metodi esplorativi;
- metodi descrittivi;
- metodi previsivi;
- metodi locali.

Il procedimento di costruzione del modello può essere ad apprendimento *supervisionato* (come nella classificazione e nella regressione) o *non supervisionato* (come nel clustering e nella scoperta di associazioni e sequenze). Nel primo caso la costruzione del modello avviene basandosi, in ciascuna tupla, sui valori assunti dalle variabili indipendenti (o variabili esplicative) rispetto ai corrispondenti valori assunti dalle variabili dipendenti (o target, o risposta), essendo questi ultimi noti: le realizzazioni delle variabili dipendenti costituiscono un supervisore del problema trattato. Nel secondo caso, invece, il modello è costruito solo in base ai valori assunti, in ciascuna tupla, dalle variabili indipendenti, perché le variabili dipendenti o i loro valori non sono noti a priori (scoperta di associazioni e sequenze), oppure perché non esistono variabili dipendenti (clustering).

La seguente descrizione si concentra sull'apprendimento supervisionato che ha una più chiara definizione rispetto all'apprendimento non supervisionato.

Una volta deciso il tipo di modello da costruire, occorre scegliere l'algoritmo che costruirà tale modello; tale scelta influisce anche sulle operazioni da effettuare nella fase della preparazione dei dati: ad esempio una rete neurale richiede che le variabili categoriche siano trasformate in gruppi di variabili binarie. Quando l'insieme dei dati da analizzare (*dataset*) è pronto si procede all'addestramento del modello.

Essenzialmente l'apprendimento supervisionato si effettua mediante una fase di addestramento (*training*) del modello su una porzione del dataset, seguita da una fase di verifica (*testing*) del modello sulla restante parte del dataset. Il modello può ritenersi costruito quando il ciclo di training e testing è terminato con esito soddisfacente. Qualche volta c'è bisogno di un terzo insieme di dati, detto insieme di convalida (*validation dataset*), perché i dati di test potrebbero influenzare le caratteristiche del modello; l'insieme di convalida esegue una misura indipendente dell'accuratezza del modello (ad esempio si potrebbero usare i dati storici contenuti nel data warehouse per il training e il testing, e i dati presi in tempo reale dal database transazionale per la validation).

Le fasi di training e di testing richiedono che il dataset venga spezzato in almeno due parti: la prima sarà usata durante il training per la stima dei parametri del modello, la seconda sarà usata durante il testing per la verifica dell'accuratezza del modello. Se questi gruppi non sono distinti, l'accuratezza del modello sarà sovrastimata, perché una parte dei dati di test è servita anche ad addestrare il modello. Una volta addestrato il modello, il tasso di accuratezza che si ottiene fornendogli l'insieme di testing risulta una buona stima della accuratezza che si può ottenere fornendo al modello dei dati nuovi. Non garantisce la correttezza del modello, ma se lo si usa su database con dati non dissimili da

quelli degli insiemi di training e testing, allora l'accuratezza media sarà prossima a quella ottenuta col passo di testing.

Il più semplice metodo di testing è noto come convalida semplice (*simple validation by percentage split*). Si impone che una certa percentuale del dataset (tipicamente dal 5% al 33%) formi l'insieme di test, e questo non verrà usato per la costruzione del modello. La divisione tra dati di testing e di training deve essere casuale affinché entrambi gli insiemi ottenuti riflettano le caratteristiche peculiari dell'intero dataset. Anche nella costruzione di un solo modello la convalida semplice può essere richiesta un dozzina di volte.

Dopo il passo di testing alcuni elementi (tuple) del dataset saranno classificati in modo errato: il rapporto tra il numero di classificazioni scorrette e il numero totale di istanze fornisce il *tasso di errore*, mentre il rapporto tra classificazioni corrette e il numero totale di istanze è il *tasso di accuratezza* (accuratezza + errore = 1).

Se si ha a disposizione una modesta quantità di dati per la costruzione del modello, un metodo di verifica più affidabile è la convalida incrociata (*cross validation*). Tale metodo prevede la divisione casuale del dataset in due (o più) insiemi (*folds*) di pari cardinalità. Il modello è realizzato con l'intero dataset a disposizione, mentre la verifica avviene così: viene costruito un modello sul primo sottoinsieme e viene testato sul secondo ottenendo una certa accuratezza; si esegue poi lo stesso processo scambiando i due sottoinsiemi, e la media delle due (indipendenti) accuratezze dà una buona stima dell'accuratezza del modello costruito con tutti i dati. Se il dataset è stato diviso in più di due gruppi la verifica avviene così: ogni gruppo è visto come insieme di test, per cui viene creato un modello utilizzando i dati dei restanti gruppi, e tale modello viene poi

testato sul gruppo in questione. La media delle accuratezze ottenute è una buona stima dell'accuratezza del modello costruito utilizzando tutto il dataset.

L'inizializzazione (*bootstrapping*) è un'altra tecnica per la stima dell'accuratezza del modello e viene usata principalmente con piccoli insiemi di dati. Come per la cross validation, il modello è costruito su tutto l'insieme. Dopo vengono creati molti insiemi di training campionando l'insieme dei dati in modo casuale (può succedere che uno stesso record sia presente più di una volta nello stesso insieme di training). Vengono creati almeno 200 insiemi di training, e tipicamente la stima finale dell'accuratezza è data dalla media delle accuratezze di ogni insieme di bootstrap.

Nessuno strumento o modello è perfetto per tutti i dati, e prima di iniziare è difficile, se non impossibile, accertare quale tecnica lavorerà meglio delle altre; molto spesso si devono costruire molti modelli prima di trovarne uno soddisfacente (ossia sufficientemente accurato).

Per ogni modello costruito si devono iterare le fasi di apprendimento e verifica, e spesso anche la fase di preparazione dei dati; la quantità di tempo usata nella ricerca di un buon modello è significativamente ridotta se l'algoritmo può sfruttare i vantaggi della computazione parallela.

Dopo che si è costruito il modello, si valutano i risultati che si ottengono e se ne interpreta la significatività.

Per problemi di classificazione, la *matrice di confusione* è uno strumento molto utile per capire la qualità dei risultati: essa mostra la distribuzione tra le varie classi dei valori attuali contro quella dei valori predetti dal modello. Non mostra soltanto quanto il modello predice bene, ma presenta anche i dettagli necessari a capire meglio dove le cose non vanno per il verso giusto.

	Attuali		
Predette	**CLASSE A**	**CLASSE B**	**CLASSE C**
CLASSE A	45	2	3
CLASSE B	10	38	2
CLASSE C	4	6	40

Figura 1.2 Matrice di Confusione

Le colonne della matrice riportano le classi attuali mentre le righe le classi predette, cioè nella cella i-j si trova il numero di record che appartengono alla j-esima classe e che vengono assegnati alla i-esima classe dal modello. Ne segue che sulla diagonale principale ci sono i numeri delle predizioni corrette, mentre nelle altre posizione ci sono il numero delle predizioni errate. In figura 1.2 è riportato un esempio di matrice di confusione: in essa il modello ha predetto correttamente 38 dei 46 elementi della classe B (46 = 2 + 38 + 6 = somma degli elementi della colonna della classe B) e ne ha sbagliati 8, ritenendoli 2 della classe A e 6 della classe C. Ciò è molto più significativo che dire che l'accuratezza è pari all'82% (100* (45+38+40)/(45+10+4+2+38+6+3+2+40) = 100*123/150 = 82). In particolare se ci sono costi differenti associati ad ogni tipo di errore (elemento della classe X classificato erroneamente come appartenente alla classe Y), un modello con bassa accuratezza potrebbe essere preferibile ad un altro con migliore accuratezza ma con un costo superiore a causa dei particolari tipi di errore che compie.

Ma anche la comprensibilità del modello è un importante criterio di valutazione: in alcune applicazioni è fondamentale poter spiegare perché una certa deci-

sione è stata presa, in altre invece è d'importanza critica ottenere ottimi valori di accuratezza, per cui la comprensibilità passa in secondo piano. Nel primo caso, quindi, sarà meglio non utilizzare un algoritmo di rete neurale per costruire il modello, mentre nel secondo caso questa potrebbe essere la scelta migliore. In generale gli alberi di decisione e i sistemi basati sulle regole esplicano meglio le ragioni implicite che hanno portato ad un certo risultato. Tuttavia anche un albero o una regola possono diventare talmente complessi da renderli non più interpretabili.

Un altro strumento di grande aiuto per valutare l'utilità di un modello è il grafico di lift (o di guadagno): esso mostra come i risultati cambiano con l'adozione del modello in esame. Ad esempio si può inviare della pubblicità via posta a un certo numero di persone scelte in modo casuale, e allo stesso numero di persone selezionate in base al modello, e verificare gli scostamenti dei risultati.

Inoltre è importante accertarsi anche dell'effettivo valore delle soluzioni trovate dal modello. Ad esempio, può capitare che un pattern è interessante, ma tentare di sfruttarlo costa più dei vantaggi che può generare. O magari, teoricamente i risultati del modello possono essere sfruttati in modo proficuo, ma in pratica non esistono gli strumenti idonei per beneficiare della conoscenza acquisita.

Infine, in qualsiasi maniera venga stimata l'accuratezza del modello, non si ha la garanzia che esso rifletta ciò che accade nel mondo reale. Un modello formalmente valido non è necessariamente un modello corretto: questa discrepanza è causata principalmente dalle assunzioni implicite fatte durante la costruzione del modello. Ad esempio, il tasso di inflazione può essere escluso dal modello di previsione della propensione agli acquisti dei consumatori, ma forte sbalzi di esso certamente influiscono sul comportamento delle persone.

Pertanto è importante testare il modello sul mondo reale (*validazione esterna*). Conviene selezionare prima un piccolo ambito dove applicare il modello, verificarne l'efficacia in questo spazio ristretto, e solo dopo aver ottenuto riscontri positivi estenderlo all'intero dominio di applicazione.

D) Utilizzo del modello

Dopo che si è costruito e "validato" un modello per data mining, lo si può impiegare in due modi differenti.

Un primo modo consiste in un uso diretto da parte degli analisti, che semplicemente visionano il modello e i suoi risultati. Il modello è applicato a dati differenti da quelli usati per costruire il modello e per validarlo, oppure è usato per selezionare insiemi di records del database al fine di sottoporli ad ulteriori analisi con altri tipi di strumenti, come ad esempio l'OLAP. A seguito di questi impieghi, l'analista suggerisce le eventuali azioni da intraprendere per il raggiungimento degli obiettivi fissati.

Un utilizzo diverso consiste nell'integrare il modello in parte di un processo più complesso, come l'analisi dei rischi o la rivelazione di frodi; in tal caso il modello è incorporato in una applicazione di più vasta portata. Ad esempio un modello di predizione può essere integrato in una simulazione impiegata nella programmazione delle promozioni. Oppure il modello può essere incluso in un sistema di gestione di un inventario, che automaticamente genera delle ordinazioni quando prevede che i livelli delle provvigioni scendano al di sotto di determinate soglie.

I modelli di data mining sono spesso applicati a un evento o transazione per volta. La quantità di tempo necessaria per processare tutte le nuove transazioni,

e la velocità con la quale queste arrivano, determineranno se è necessario far ricorso ad algoritmi paralleli. Così, mentre le applicazioni per la valutazione dei rischi connessi alla concessione di un credito possono essere eseguite facilmente su computer di modeste dimensioni, il monitoraggio delle operazioni con carte di credito al fine della rivelazione di eventuali frodi potrebbe richiedere l'uso di un sistema parallelo.

Quando si ha a che fare con applicazioni complesse, il data mining è spesso solo un piccola parte, sebbene di importanza critica, del prodotto finale. Ad esempio, la scoperta di conoscenza attraverso il data mining può venire combinata con la conoscenza di esperti del settore e applicata ai dati nel database. In un sistema di rivelazione delle frodi, le forme note di frodi (pattern) vengono confrontate coi pattern scoperti utilizzando i modelli costruiti con il data mining.

E) Monitoraggio del modello

Quando si utilizza un modello è necessario monitorarlo, al fine di valutare la qualità del lavoro da esso svolto: bisogna continuamente controllarlo, anche quando funziona bene, perché col passare del tempo tutti i sistemi evolvono e variano i dati che essi producono.

Ad esempio, i commercianti sanno che i gusti della gente cambiano; inoltre a lungo andare il tasso di inflazione può influire sui comportamenti degli acquirenti; oppure variabili esterne correttamente trascurate in precedenza possono diventare influenti.

Di tanto in tanto il modello va ritestato, ed eventualmente riaddestrato o addirittura ricostruito daccapo. I diagrammi che confrontano i risultati previsti con

quelli osservati sono degli eccellenti indicatori del comportamento del modello. Tali diagrammi sono facili da usare e da capire, non sono computazionalmente dispendiosi e il software che li produce può essere incorporato col software che implementa il modello.

1.2) Il Data Mining

"Per data mining si intende il processo di selezione, esplorazione e modellazione di grandi masse di dati, al fine di scoprire regolarità o relazioni non note a priori, e allo scopo di ottenere un risultato chiaro e utile al proprietario del database". [01]

Traducendo dall'inglese, *to mine* significa "scavare per estrarre": questo verbo rende l'idea di come alla base del data mining vi sia una ricerca in profondità, nella massa dei dati disponibili, per trovare informazioni aggiuntive, non precedentemente note né ovvie.

Il problema dell'estrazione di conoscenza da database enormi va risolto tramite un processo di elaborazione articolato e complesso, formato da molti passi, che possono andare dalla manipolazione dei dati sino a sofisticate tecniche di inferenza statistica, di ricerca e di ragionamento artificiale.

Il KDD indica l'intero processo di scoperta di conoscenza dai dati dei database; il Data Mining è un particolare passo in questo processo: l'applicazione di specifici algoritmi per l'estrazione di pattern. Gli altri passi del processo di KDD, come ad esempio la preparazione, la selezione, la pulizia dei dati (data cleaning), l'incorporamento di conoscenza già acquisita, l'interpretazione dei risultati che scaturiscono dal data mining stesso, sono essenziali per garantire che la conoscenza estratta sia effettivamente valida e utile. Al contrario, l'applicazione alla cieca dei metodi

di Data Mining (giustamente criticata come "data dredging" nella letteratura statistica) può risultare una attività rischiosa, che facilmente porta alla scoperta di pattern non significativi o addirittura erronei.

Il data mining, che come detto è uno dei passi del processo di KDD, trova i pattern e le relazioni tramite la costruzione di modelli. I modelli sono delle rappresentazioni astratte della realtà: non si dovrebbe mai confondere il modello con la realtà, però un buon modello è un'utile guida alla comprensione dei problemi con i quali si ha a che fare, e suggerisce le azioni che possono essere intraprese per raggiungere i propri scopi.

Esistono due tipi principali di modelli in data mining. Il primo, quello dei modelli predittivi, usa dati che rappresentano fatti o eventi dei quali sono noti l'evoluzione finale e i risultati per costruire un modello che viene usato per predire esplicitamente l'evoluzione e i risultati per altri dati. Ad esempio un modello può essere costruito con la storia dei pagamenti delle persone alle quali è stato fatto un prestito, allo scopo di aiutare l'identificazione di chi probabilmente non restituirà i prestiti

Il secondo tipo, quello dei modelli descrittivi, descrive i pattern esistenti nei dati. La differenza fondamentale tra i due tipi di modelli è che i modelli predittivi fanno delle predizioni esplicite (come ad esempio la probabilità di una mancata restituzione di un prestito), mentre i modelli descrittivi sono usati per aiutare a costruire un modello predittivo o, semplicemente, per ottenere una più profonda conoscenza dei dati e della realtà che essi rappresentano.

Chiaramente, ogni organizzazione che conosce molto bene il proprio ambiente e i propri problemi, già è a conoscenza di molti importanti pattern che i suoi operatori hanno osservato negli anni. Ciò che il data mining può fare è non solo di con-

fermare queste osservazioni empiriche, ma anche trovare pattern nuovi e più sottili. Questa nuova conoscenza può produrre ottimi risultati creando costanti miglioramenti.

Il data mining è un insieme di strumenti, non è un sistema completamente automatico per l'estrazione d'informazioni dai database. Non viene eliminato il bisogno di conoscere l'ambiente di lavoro, di capire cosa rappresentano i dati e come funzionano i metodi analitici d'elaborazione; il data mining assiste l'analista nella ricerca dei pattern e delle relazioni nei dati, ma non dice quanto essi valgano per chi li estrae. Il data mining non sostituisce gli abili analisti ma piuttosto dà loro un nuovo potente strumento per migliorare il lavoro da svolgere. Ma per usare efficacemente uno strumento di data mining è importante comprendere il funzionamento delle metodologie e degli algoritmi che sono alla sua base. Ad esempio, le scelte fatte nell'inizializzazione degli strumenti di data mining e le ottimizzazioni selezionate influenzano l'accuratezza e la velocità dei modelli.

Bisogna inoltre avere una buona conoscenza dei dati: la qualità dei risultati, ad esempio, sarà spesso influenzata dalla presenza di valori che si discostano molto dai valori tipici nel database (*outliers*), dalla presenza di campi irrilevanti o di campi correlati tra loro, dal modo nel quale i dati sono codificati, dai dati presi in considerazioni o da quelli esclusi.

1.3) Data Warehousing, OLAP e Data Mining

La tecnologia delle basi di dati è finalizzata alla gestione efficiente e affidabile di dati “in linea” (*On Line Transaction Processing*, OLTP). Tramite questa tecnologia, le imprese accumulano grandi moli di dati relativi alla loro gestione operativa quotidiana. Questi dati però possono rivelarsi utili non solo per la gestione

dell'impresa, ma anche per la pianificazione e il supporto delle decisioni. Per molti anni lo sviluppo della tecnologia ha trascurato l'analisi dei dati, pensando che i linguaggi di interrogazione (come ad esempio SQL) fossero sufficienti sia per la gestione operativa che per l'analisi. Con l'inizio degli anni Novanta, parallelamente allo sviluppo delle reti e dei prodotti per la distribuzione dei dati, si sono imposte nuove architetture, caratterizzate dalla *separazione degli ambienti*: a fianco dei sistemi per OLTP si sono sviluppati sistemi dedicati esclusivamente all'elaborazione e all'analisi dei dati (*On Line Analytical Processing*, OLAP). L'elemento principale dell'architettura OLAP, che svolge il ruolo di server (analogo al DBMS per OLTP), è il *data warehouse*; in questa architettura i sistemi OLTP tradizionali svolgono il ruolo di fonti di dati (*data source*) per l'ambiente OLAP.

Sebbene il data warehousing abbia contenuti e pratiche distinti dal data mining, è strettamente associato ad esso: infatti, la diffusione del data warehousing è uno dei principali fattori dell'accresciuto interesse per il data mining. Tuttavia un data warehouse non è un prerequisito essenziale per la efficace applicazione del data mining, e molti strumenti per il data mining lavorano anche su più semplici file, eventualmente estratti dal sistema OLTP.

Una definizione di *data warehouse* è stata fornita da Inmon nel 1992:

> *"Un data warehouse è un data base relazionale subject oriented, integrato, non volatile, time variant, progettato per il supporto alle decisioni."* [02],[07]

- Subject oriented (orientato ai soggetti): i data warehouse sono progettati per aiutare l'azienda ad analizzare i propri dati per i propri scopi: ad esempio, i dati sono definiti e organizzati in termini affaristici, raggruppati sotto eti-

chette orientate ai soggetti ("Clienti", "Vendite", ecc.).

- Integrato: gli oggetti del data warehouse sono definiti in modo tale che risultino validi per l'impresa e le sue sorgenti operazionali e le sorgenti esterne; ciò comporta la risoluzione dei conflitti tra nomi e dei problemi derivanti dal fatto che i dati si trovano espressi in unità di misure differenti. Ad esempio tutte le occorrenze di elementi comuni di dati, come il codice identificativo di un cliente, devono essere rappresentate consistentemente per permettere la realizzazione di resoconti consistenti dal data warehouse.
- Non volatile: una volta entrati nel data warehouse, i dati non vengono più modificati. Così diventano una risorsa stabile per la realizzazione di rapporti e per le analisi comparative.
- Time variant (variabile nel tempo): ogni dato nel data warehouse è marcato temporalmente con l'indicazione del momento d'ingresso nel data warehouse stesso: così si realizza una registrazione cronologica che permette l'analisi storica. Mentre infatti nei sistemi OLTP viene normalmente descritto solo lo “stato corrente” di una applicazione, i dati presenti nel warehouse sono di tipo *storico-temporale*. I meccanismi di importazione dei dati sono normalmente di tipo asincrono e periodico, in modo da non penalizzare le prestazioni dei data source, specie se si tratta di sistemi OLTP con prestazioni particolarmente critiche, per cui il data warehouse non contiene dati perfettamente aggiornati rispetto al flusso di transazioni che operano nei sistemi OLTP (un disallineamento controllato dei dati è in genere ritenuto accettabile per molte applicazioni di analisi).

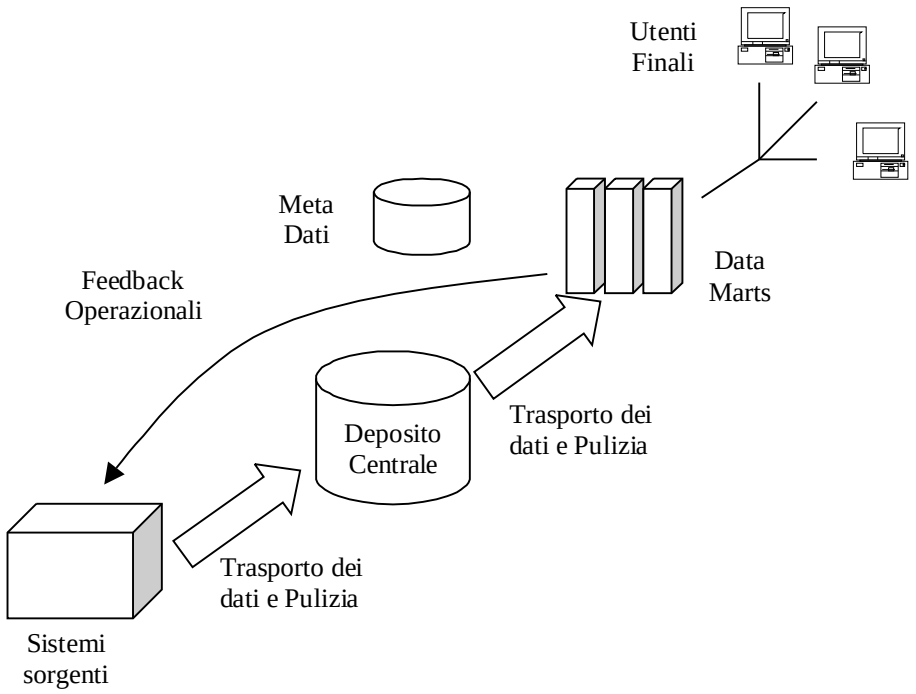


Figura 1.3 Struttura di un data warehouse

L'architettura di un data warehouse è costituita essenzialmente dai seguenti componenti principali:

- *sistemi sorgenti*: un data warehouse contiene dati che vengono estratti da uno o più sistemi, detti *sistemi sorgenti* o *data source*; questi includono una vasta tipologia di sistemi, comprendenti anche raccolte dati non gestite tramite DBMS;
- un componente (a volte diviso in più moduli) che si occupa del *trasporto* e della *pulizia* dei dati: muove i dati fra i diversi dispositivi di memorizzazione del data warehouse e si occupa del filtraggio dei dati per assicurarne correttezza e consistenza;
- *deposito centrale*: è il principale dispositivo di memorizzazione del data warehouse;
- un componente contenente i *metadati*, che descrivono cosa è disponibile, in

che forma e dove;

- *data mart*: forniscono accessi veloci e specializzati ai dati per gli utilizzatori finali e per le applicazioni (un data mart è un data warehouse più piccolo che funziona indipendentemente o può essere interconnesso con altri data mart per formare un warehouse integrato). Un data mart memorizza un sottoinsieme dei dati del data warehouse, normalmente in forma molto aggregata, utile ad un particolare dipartimento direzionale;
- componente per i *feedback operazionali*: integra le decisioni suggerite dai dati all'interno del sistema;
- *utenti finali*: sono gli utilizzatori del data warehouse.

Un problema importante nella gestione di un data warehouse è quello della *qualità dei dati*. I sistemi sorgenti forniscono al data warehouse i dati in forma grezza (ad esempio in un grosso supermercato una vendita viene registrata dalla cassa con un codice dell'articolo, l'ora, eccetera): questi dati sono stati progettati per il funzionamento del sistema, non per il supporto alle decisioni, e spesso adattare un data warehouse ai sistemi sorgenti presenta diverse difficoltà.

Gli strumenti per i trasporto e la pulizia dei dati si riferiscono fondamentalmente al software usato per muovere i dati e adattarli alla struttura del data warehouse.

Il deposito centrale è la parte più tecnicamente avanzata del data warehouse. È il database che contiene i dati ed è caratterizzato da tre fattori:

- hardware scalabile,
- sistema di database relazionale,
- modello logico dei dati.

La scalabilità si riferisce al fatto che l'hardware deve avere la capacità di crescere virtualmente senza limiti. Un unico hardware deve supportare molti utenti, molti

dati, molti processi, e ciò è stato possibile negli ultimi anni grazie ai miglioramenti nella tecnologia parallela. Le macchine parallele possono crescere aggiungendo più dischi, più memoria, più processori e più banda passante tra i componenti stessi. Tutto questo è di fondamentale importanza dato il rapido aumento della quantità dei dati da memorizzare e analizzare in un data warehouse.

I database relazionali sono ormai maturi per approfittare delle piattaforme hardware scalabili in tutte quelle operazioni che coinvolgono un gran numero di dati: caricamento dei dati, costruzione degli indici, esecuzione di copie di salvataggio, esecuzione di query.

I metadati sono spesso una componente ignorata del data warehouse. Un buon sistema di metadati potrebbe includere le seguenti cose:

- annotazioni sul modello logico dei dati, che spiegano il significato delle entità, degli attributi, quali sono i vincoli e i valori validi, eccetera;
- le relazioni tra il modello logico e i sistemi sorgenti;
- lo schema fisico;
- viste e formule più comuni per l'accesso ai dati;
- informazioni per la sicurezza e l'accesso.

I data mart servono a superare i problemi del deposito centrale nel dover servire migliaia di utenti con bisogni differenti. Un data mart è un sistema specializzato che raccoglie insieme i dati necessari ad un certo settore di utenti (dipartimento) e di applicazioni.

Molti sono i vantaggi quando si ha a disposizione un data warehouse.

- l'integrazione dei dati permette all'analista di osservare e accedere ai dati facilmente e rapidamente. Senza i dati integrati invece si spende molto tempo per la pulizia e l'aggregazione dei dati prima di iniziare il processo di data mi-

ning. Il problema della pulizia dei dati per un data warehouse e per il data mining sono molto simili;

- nel data warehouse i dati sono presenti in forma dettagliata e "sommarizzata" (*summarized*), cioè sotto forma di riassunti (ad esempio la media e la deviazione standard di un certo campo) e di descrizioni sintetiche di sottoinsiemi di dati, perciò l'analista che deve usare il data mining si risparmia di compiere tali operazioni (spesso necessarie per avere una visuale chiara dei dati con cui si sta lavorando);
- i dati sono ben individuati nel tempo (dati storici), e ciò permette lo studio nel tempo delle grandezze di interesse e di individuare pattern sul comportamento a lungo termine;
- la struttura di un data warehouse consente di effettuare interrogazioni (query) molto complesse (talvolta coinvolgono milioni di record e migliaia di attributi).

Malgrado questi vantaggi, la creazione di un ampio data warehouse che fonde i dati da tante sorgenti, che risolve i problemi d'integrità e che carica i dati in un database per le interrogazioni, può risultare un compito oneroso, richiedendo talvolta alcuni anni di lavoro e costi elevati.

L'OLAP (*On Line Analytical Processing*) è una metodologia che si pone l’obiettivo di fornire un supporto efficiente all’analisi multidimensionale dei dati, intesa come analisi delle informazioni effettuata prendendo in considerazione più dimensioni contemporaneamente. La sigla mette in evidenza che l’analisi dei dati avviene “in linea”, cioè tramite strumenti interattivi. Il KDD, invece, focalizza la propria attenzione principalmente sull'automatizzazione dell’analisi dei dati.

Secondo una ben nota definizione, "*OLAP è il nome dato all'analisi dinamica dell'impresa necessaria per creare, manipolare, animare e sintetizzare informazioni dagli Enterprise Data Models*" (Codd). Questo processo consente:

- di scoprire nuove relazioni tra le variabili;
- di identificare i parametri necessari alla gestione di grosse quantità di dati;
- di creare un numero illimitato di dimensioni;
- di specificare condizioni ed espressioni che coinvolgono, contemporaneamente, più dimensioni.

Uno strumento OLAP, per essere considerato tale, deve rispettare diverse caratteristiche, racchiuse nelle cinque parole chiavi: Fast Analisys of Shared Multidimensional Information (FASMI), ovvero analisi veloce di informazione multidimensionale condivisa.

La rapidità è una delle caratteristiche richieste ad uno strumento OLAP, ma non è ottenibile facilmente quando si ha a che fare con grandi quantità di dati, in particolar modo quando si richiedono elaborazioni complesse. Si ricorre quindi a numerose tecniche per raggiungere questo obiettivo, tra le quali: forme di memorizzazione specializzate, estese pre-elaborazioni e stringenti requisiti hardware.

La multidimensionalità dei dati è la richiesta principale per uno strumento OLAP. I dati sono organizzati in ipercubi, ognuno dei quali ha un certo numero di dimensioni. Una dimensione è un attributo o un attributo strutturato formato da una lista di membri (nomi o etichette, in generale valori discreti), che sono considerati dall'utente tutti dello stesso tipo. Spesso una dimensione ha una struttura gerarchica, ad esempio una dimensione temporale può essere strutturata in anni, ognuno dei quali è suddiviso in mesi, così come i campi dello stato, delle regioni e delle città, organizzati gerarchicamente, formano la dimensione geografica. Una dimensione

agisce da indice per identificare i valori racchiusi nell'ipercubo; fissando il valore (o l'intervallo di valori) di alcune dimensioni si ottengono dei sotto-ipercubi, e se è fissato il valore per ogni dimensione si ottiene una cella. Una regola fondamentale per la costruzione di un ipercubo è che un record di dati deve essere contenuto in una unica cella. Una cella contiene, oltre al gruppo di dati indicizzati dalle dimensioni, anche dei dati aggregati, in particolare sui campi che non formano una dimensione (attributi aggregati): esempi di dati aggregati sono la media e la somma di un certo attributo, il numero di valori all'interno della cella con certe caratteristiche, eccetera. Selezionando un sotto-ipercubo, si ottengono i dati aggregati per quella parte di record indicizzati dai membri delle dimensioni che identificano il sotto-ipercubo stesso: ad esempio selezionando un certo anno (che raccoglie 12 mesi) e una certa città si può ottenere il numero di nascite per quella città in quell'anno, oppure selezionando un certo mese e una certa regione (che raccoglie più città) si può ottenere il numero di nascite per quella regione in quel mese. L'operazione di aggregazione dei dati, ossia il passaggio da una visuale più dettagliata a una visuale più generale (ad esempio: città → regione) è detta *roll up*; l'operazione inversa, che comporta una disaggregazione dei dati (ad esempio: anno → mese) è detta *drill down*. Altre tipiche operazioni compiute sui dati multidimensionali sono quelle denominate *pivot* (che comporta un riorientamento del cubo) e *Slice & Dice* (che comporta una proiezione dell'ipercubo su un piano, solitamente bidimensionale).

Il concetto di condivisione implica che uno strumento OLAP dovrebbe saper gestire la multiutenza (ad esempio dovrebbe avere una architettura client-server) e dovrebbe soddisfare tutti i requisiti di sicurezza per la riservatezza e la gestione degli accessi concorrenti.

Data mining e OLAP sono due strumenti spesso ritenuti simili, ma in effetti presentano numerose differenze e possono completarsi a vicenda.

L'OLAP fa parte dell'insieme degli strumenti per il supporto alle decisioni. Tradizionalmente gli strumenti d'interrogazione (*query*) e per realizzare dei rapporti (*report*) descrivono cosa c'è nel database. L'OLAP va oltre, in quanto fornisce la risposta sul "*se*" e sul "*perché*" certi fatti siano veri: l'utente formula un'ipotesi (riguardo ad una relazione) e la verifica con una serie di query sui dati. Ad esempio un analista potrebbe voler determinare quali fattori portano alla mancata restituzione di un prestito: egli può inizialmente ipotizzare che la gente con basso reddito è ad alto rischio, quindi analizza il database con l'OLAP per verificare (o confutare) quest'assunzione; poi può rivolgere l'attenzione ai debiti elevati, ritenendoli causa di rischio, e così via, utilizzando anche combinazioni di fattori.

In altre parole l'analista OLAP genera una serie d'ipotetici pattern e usa delle query sul database per verificarli o confutarli. Ma quando il numero di variabili sale a decine o a centinaia diventa molto più difficoltoso formulare buone ipotesi (richiedendo tra l'altro molto più tempo) e analizzare il database con l'OLAP.

Il data mining si differenzia dall'OLAP perché, piuttosto che verificare degli ipotetici pattern, usa gli stessi dati per scoprire tali pattern. Per esempio se l'analista usa gli strumenti del data mining per identificare i fattori di rischio per la restituzione di un prestito potrà scoprire che le persone con debiti elevati e bassi guadagni sono ad alto rischio, ma potrà anche scoprire altri pattern che non aveva ipotizzato, come ad esempio che l'età della persona è un fattore di rischio.

E' qui che il data mining e l'OLAP si integrano l'un l'altro. Prima di sfruttare il pattern scoperto (nell'esempio precedente, usandolo per decidere chi potrà accedere al credito), l'analista ha bisogno di conoscere quali saranno le implicazioni.

Gli strumenti OLAP permettono all'analista di rispondere a questo tipo di questioni.

In più, l'OLAP è complementare anche nelle fasi del processo di scoperta di conoscenza precedenti al passo del data mining, poiché aiuta a comprendere i propri dati, concentrando l'attenzione sulle variabili importanti, identificando le eccezioni, gli outliers, i clusters, eccetera. Ciò è importante perché più si conoscono i dati e più risulta efficace il processo di KDD.

1.4) Applicazioni del Data Mining

Il data mining è diffuso tra molte organizzazioni grazie ai concreti apporti che fornisce; può essere usato nel controllo (ad es. dei costi, dei processi) e può contribuire al miglioramento dei risultati.

Ricordiamo che il data mining, che per definizione è il passo della scoperta di conoscenza nei database durante il quale si applicano gli algoritmi, è applicabile ovunque vi sia un database, e in particolare un database massivo.

Esistono numerosi esempi di campi in cui è stato applicato con successo il processo di KDD. Tra i principali figurano:

- **Gestione del mercato (market management)**

  Questa è l'area di applicazione dove il data mining è più stabilmente utilizzato. L’area di applicazione più conosciuta è il Database Marketing: l’obiettivo è quello di condurre mirate ed efficienti campagne promozionali, attraverso l’analisi dei dati contenuti nei database dell'impresa.

  Gli algoritmi di data mining setacciano i dati, cercando categorie di consumatori che hanno le stesse caratteristiche ed applicando per ciascuna categoria una specifica strategia di approccio. In questo modo il cliente non è disturbato da

una eccessiva pubblicità, e nello stesso tempo gli addetti del marketing limitano i costi, facendo poche ma efficaci pubblicità.

Un'altra area di applicazione del data mining nella gestione del mercato è quella che si occupa di determinare nel tempo i pattern sugli acquisti dei clienti (Market Basket Analysis). Il data mining viene applicato anche per le campagne di vendite incrociate (cross-selling): si ha un servizio di vendite incrociate quando un venditore al dettaglio o un fornitore di servizi cerca di rendere allettante la proposta di acquisto di un prodotto o di un servizio per un cliente che ha già acquistato un altro prodotto o servizio collegato al primo.

- **Gestione dei rischi (Risk Management)**

  Il Risk Management non comprende soltanto il rischio associato alle assicurazioni e agli investimenti, ma una categoria più ampia di rischi derivanti dalla competitività delle aziende, dalla scarsa qualità dei prodotti e dalla perdita dei clienti (attrition).

  Il rischio è l'aspetto essenziale nell'attività assicurativa, ed il data mining è adatto a predire le linee di condotta da seguire. Le predizioni sono di solito espresse sotto forma di regole che vengono applicate ad esempio al potenziale sottoscrittore di una polizza assicurativa.

- **Prevenzione delle frodi (Fraud detection and management)**

  Molte organizzazioni appartenenti a diverse aree, come vendite al dettaglio, servizi di carte di credito, assistenza sanitaria e società di telecomunicazioni, utilizzano il data mining per rilevare e per prevenire le frodi. L'approccio usuale è il seguente: servendosi dei dati storici si costruisce un modello di comportamento fraudolento o potenzialmente fraudolento, e successivamente lo si utilizza per identificare comportamenti simili al modello costruito.

- **Investimenti finanziari**

  Molte applicazioni per l'analisi finanziaria adottano tecniche con modelli predittivi (ad esempio reti neurali e regressione statistica) allo scopo di creare e ottimizzare i portafogli o costruire modelli di scambio. Per mantenere il vantaggio competitivo gli utenti e gli sviluppatori di tali applicazioni, che rimangono in uso per diversi anni, raramente rendono noto i loro precisi dettagli e la loro efficacia.

- **Gestione delle reti**

  Un area di applicazione dove la componente temporale è dominante risulta essere la gestione delle reti di telecomunicazione. Queste reti estese e complesse producono quotidianamente molti allarmi, e le sequenze di essi contengono informazioni implicite sul comportamento della rete. Col data mining si può estrarre conoscenza preziosa sull'intero sistema e sulle sue prestazioni. Le regolarità e l'ordine degli allarmi sono usati nel sistema di gestione dei guasti per filtrare gli allarmi ridondanti, localizzare i problemi nella rete e predire i guasti pericolosi.

- **Produzione industriale e manifatturiera**

  Il controllo e la pianificazione dei processi tecnici di produzione rappresentano una delle aree di applicazione del data mining con maggiori opportunità di sviluppo e di profitto. Infatti, sebbene durante un processo di produzione siano spesso rilevati e immagazzinati grandi quantità di dati, il più delle volte tali dati sono scarsamente sfruttati.

  Il principale vantaggio dell'applicazione dei metodi della scoperta di conoscenza in questa area è il risparmio sui costi che si ottiene quando i risultati sono utilizzati per il controllo di processi costosi. Un altro tipico esempio di utilizzo

del data mining in questo settore è dato dal sistema CASSIOPEE, realizzato dalla General Electric e dalla SNECMA: tale sistema è stato utilizzato da tre delle maggiori compagnie aeree Europee per diagnosticare e predire eventuali problemi per il Boeing 737. [03]

- **Data mining applicato ai dati scientifici**

  Gli strumenti scientifici (satelliti, microscopi, telescopi) possono facilmente generare terabytes e petabytes di dati a velocità dell'ordine dei gigabytes per ora. Ne segue un rapido allargamento del gap tra la capacità di raccogliere i dati e l'abilità di analizzarli. L'approccio tradizionale consiste nell'andare a caccia tra i dati grezzi di fenomeni (spesso ipotizzati) e delle strutture sottostanti. Un ricercatore riesce a lavorare efficacemente con poche migliaia di osservazioni, ognuna con un piccolo numero di variabili (di solito non più di una decina); posto di fronte a milioni di record, ognuno con centinaia o migliaia di variabili misurate, la tecnica tradizionale non è più applicabile. Tuttavia, quando il problema da trattare è ben conosciuto e il ricercatore sa cosa sta cercando, l'enorme quantità di dati può essere ridimensionata sottoponendola efficacemente alla data reduction (riduzione dei dati). Per data reduction in ambiente scientifico si intende l'individuazione delle variabili essenziali, ossia di maggior interesse, tra quelle presenti nelle osservazioni grezze; tale procedimento richiede operazioni di trasformazione, di selezione e di normalizzazione. Quando questo non si può fare, oppure anche ridimensionando i dati essi non sono comunque analizzabili coi metodi tradizionali, si può ricorrere al data mining. Ma il data mining può essere efficacemente usato anche in fase di data reduction, grazie al ricorso degli algoritmi di classificazione e di clusterizzazione. Noti esempi di applicazione del data mining in campo scientifico sono:

- Catalogazione degli oggetti celesti: è una delle principali applicazioni del data mining; un notevole successo, ad esempio, è stato ottenuto da SKICAT, un sistema usato dagli astronomi per eseguire analisi di immagini, classificazione e catalogazione di oggetti celesti; nella sua prima applicazione il sistema è stato usato per processare 3 terabytes di dati in forma di immagini provenienti dal Secondo Osservatorio Astronomico di Palomar: è stato stimato che il numero di oggetti celesti distinguibili in tali immagini è dell'ordine di 10^9. [03]
- Ricerca dei vulcani su Venere: anche in questo caso si fa ricorso al data mining per analizzare in maniera automatizzata le immagini provenienti da sonde, satelliti e osservatori astronomici.
- Ricerca delle biosequenze nel genoma: l'identificazione delle regioni di codice genetico nelle sequenze di DNA è una delle ultime e più promettenti applicazioni del data mining.

Oltre all'enorme quantità di dati, esistono altri problemi che spingono all'applicazione del data mining nell'ambito scientifico. Tra essi ricordiamo: particolarità dei tipi di dato (immagini, suoni), dati incompleti, sparsi o poco affidabili, eccetera: in tutti questi casi il data mining è in grado di aiutare fortemente il lavoro dei ricercatori su dati altrimenti difficili da analizzare.

- **Area medica**

Le applicazioni in medicina rappresentano un'altra area feconda: il data mining può essere usato nella predizione dell'efficacia di procedure chirurgiche, di test medici o di medicazioni. Può individuare il comportamento nel tempo di alcune malattie ereditarie (anche se purtroppo l'utilizzo del data mining in questo campo è limitato dal fatto che i dati storici non risalgono molto indietro nel

tempo, salvo rarissime eccezioni). Inoltre alcune ditte farmaceutiche usano il data mining in grandi database di composti chimici e di materiali genetici alla scoperta di sostanze che potrebbero essere candidate per successive ricerche allo scopo di produrre nuovi principi per il trattamento delle malattie.

- **Text mining**

  Due aree di sviluppo emergenti per il data mining sono il text mining ed il web mining, e si basano entrambe sull'idea di utilizzare "agenti" intelligenti per navigare attraverso ambienti ricchi di informazioni (ambienti in cui l'informazione è sovrabbondante o ridondante).

  Il text mining è l'applicazione dei tipici algoritmi del data mining ai testi e ai database di testi. L'obiettivo non è una critica letteraria: piuttosto si vuole rendere disponibile alla rapida analisi e alla comprensione la quantità sempre crescente di informazione testuale. Tipiche applicazioni sono l'indicizzazione automatica dei documenti e la creazione di mappe in cui documenti simili o affini sono disposti l'uno vicino all'altro (le parole sono analizzate nel loro contesto: ad esempio, se la parola Quirinale appare in un articolo di architettura, l'articolo è disposto vicino agli articoli di architettura e non di politica). Se da un lato i sistemi di ricerca tradizionali sono generalmente vincolati da una strategia di confronto letterale sulle parole e dalla ingegnosità dell'utente, dall'altro il text mining apre la possibilità di apprendere i collegamenti che l'utente non ha inizialmente specificato o non ritiene possibili.

- **Web Mining**

  Il web mining tocca due delle aree dove attualmente è concentrata la maggiore attenzione: Internet e data mining. Un utilizzo tipico è basato sull'idea di applicare il data mining ai registri dei Web server (activity log): osservando gli

spostamenti di molti utenti, si sviluppano previsioni sulle comportamento di essi nella rete, cioè su quali potrebbero essere i futuri siti di interesse ad un certo punto della navigazione. Gli sviluppatori dei siti mettono a disposizione in modo statico i link più probabili nella navigazione ricorrendo a congetture. Con il data mining, analizzando la storia delle navigazioni, si cerca di suggerire in modo dinamico al navigatore i link che possono interessargli.

Oltre che nella ricerca dei siti di interesse, il web mining può essere utilmente impiegato nell'estrazione delle informazioni utili, anche facendo ricorso alle strategie tipiche del text mining. Esistono, ad esempio, programmi in grado di estrarre i titoli e gli autori di articoli disponibili in rete in vari formati; altri in grado di individuare file di FAQ (Frequently Asked Questions) e di estrarre da essi le risposte che interessano.

1.5) Difficoltà e problemi

Come per qualsiasi innovazione di successo, anche per il data mining i problemi e le difficoltà non mancano. I principali sono:

- **Archivi di dati ad elevata dimensionalità**

  I database di grandi dimensioni, con milioni di record e con gran numero di campi per ciasun record, sono sempre più diffusi. Questi archivi creano degli spazi di ricerca che crescono in modo combinatorio, per cui aumenta il rischio che gli algoritmi di data mining trovino pattern spuri, privi di significato o non validi. Per ovviare a questi problemi è necessario far ricorso a soluzioni innovative: algoritmi molto efficienti, campionamento dei dati, metodi di approssimazione, scalabilità ed elaborazione parallela spinta, tecniche di riduzione dimensionale e incorporazione di conoscenza precedentemente acquisita.

- **Interazione con l'utente e uso della conoscenza di base**

  Esiste l'esigenza di dare risalto più all'interazione tra utente e macchina piuttosto che alla completa automatizzazione del processo di data mining, allo scopo di facilitare il lavoro di chi non è esperto di data mining ma che comunque se ne deve servire. Molti degli attuali metodi e strumenti per il data mining non sono veramente interattivi e non incorporano facilmente la conoscenza di base sul problema, tranne che in forma semplice, sebbene l'uso di tale conoscenza è fondamentale in gran parte dei passi del processo di KDD e nel data mining in particolare.

- **Super-adattamento (overfitting)**

  Un algoritmo che sta cercando i migliori parametri per un modello usando un piccolo dataset potrebbe super-adattarsi a questi dati, cioè funzionare troppo bene su questi dati e male sui restanti, per cui l'accuratezza del modello sarebbe certamente sopravvalutata. Per evitare, almeno in parte, l'insorgere di questo problema si usano metodi quali la cross-validation ed altre strategie statistiche.

- **Dati mancanti o errati**

  Questo è un problema grave soprattutto per i database nel campo economico. Alcuni attributi o valori possono mancare perché il database non è stato realizzato allo scopo di estrarre conoscenza utile, oppure per errori degli operatori, per cadute di sistema, per errori nelle misurazioni, eccetera. Soluzioni possibili richiedono strategie statistiche più sofisticate che identificano variabili e dipendenze nascoste (al fine di compensare la mancanza).

- **Trattamento di dati e di conoscenze che cambiano nel tempo**

  Dati che cambiano rapidamente possono rendere i pattern precedentemente scoperti non più validi. In aggiunta, potrebbero esserci variazioni non solo sui

record, ma anche negli attributi dei database. Per evitare ciò si possono usare metodi incrementali per l'aggiornamento dei modelli e trattare le variazioni dei dati come l'innesco per ulteriori ricerche di possibili variazioni dei pattern.

- **Integrazione**

  Un sistema per la scoperta di conoscenza è meno efficace quando non è integrato nel sistema globale dell'organizzazione. Problemi tipici di integrazione sono: integrazione con i DBMS, integrazione con fogli di calcolo e con strumenti di visualizzazione, adattamento a sensori che leggono i dati in tempo reale.

- **Dati non standard, multimediali e orientati agli oggetti**

  Attualmente i database tendono a contenere non più solo dati numerici o in formato testo, ma anche grandi quantità di dati non standard e multimediali (grafici, testi multilingua, immagini digitalizzate, video, audio, eccetera). Attualmente questi tipi di dati sono oltre la portata della tecnologia che supporta il processo di KDD.

CAPITOLO 2

MODELLI E METODOLOGIE PER IL DATA MINING

Lo scopo del data mining è produrre nuova conoscenza che l'utente possa utilizzare, costruendo un modello del mondo reale che si basa sui dati raccolti da varie sorgenti.

Trattare i problemi del data mining in modo gerarchico dà un vigoroso aiuto alla loro soluzione. Al livello più alto della struttura c'è il problema espresso nei termini tipici dell'ambiente a cui si riferisce: ad esempio in astronomia un problema può essere quello di inserire una stella in uno dei gruppi tipici in cui si raggruppano le stelle stesse, come giganti rosse, nane bianche, eccetera.

Al livello successivo c'è il tipo di modello che si vuole utilizzare: si può costruire un modello di classificazione oppure un modello di clustering per raggruppare le stelle.

Più in basso c'è il livello dell'algoritmo che costruisce il modello. Ad esempio per la classificazione delle stelle si possono usare alberi di decisione, reti neurali, o si possono anche sfruttare algoritmi della statistica tradizionale.

L'ultimo livello della gerarchia è occupato dal prodotto (tool) usato per la realizzazione del modello. Prodotti differenti generalmente realizzano differenti implementazioni di un particolare algoritmo. Queste differenze implementative si riferiscono a caratteristiche tecniche ed operative, come l'uso della memoria e l'immagazzinamento dei dati, e influiscono quindi sulle prestazioni, come la velo-

cità di costruzione del modello e l'accuratezza.

2.1) Modelli

Un modello per il data mining generalmente è inquadrato in uno dei seguenti tipi: classificazione, regressione, serie temporali, clustering (raggruppamento), analisi delle associazioni e scoperta delle sequenze.

Spesso per molti dei problemi da affrontare il migliore approccio è costruire diversi tipi di modelli e provare più di un algoritmo per modello, perché a seconda dei dati e del problema alcuni algoritmi e modelli funzionano meglio di altri. Per di più è quasi impossibile determinare analiticamente quale algoritmo è il migliore per il modello che si sta costruendo.

Modelli di classificazione, regressione e serie temporali sono principalmente usati per la predizione, mentre i modelli di clustering, di associazione e di scoperta di sequenze sono soprattutto utilizzati per la descrizione. I più implementati per il data mining sono la classificazione e la regressione.

Nei modelli predittivi, i valori o le classi che si predicono sono dette *variabili dipendenti* o *obiettivo*, mentre i valori impiegati per realizzare la predizione sono detti *variabili indipendenti* o *predittrici*.

La classificazione, la regressione e le serie temporali sono modelli ad apprendimento supervisionato (*supervised learning*), perché per costruirli c'è bisogno di dati che siano già classificati (ossia di tuple con valori noti per le variabili dipendenti), in quanto il modello in base agli esempi apprende come comportarsi con eventi nuovi (ancora non classificati). Al contrario, per il clustering, le associazioni e la scoperta di sequenze non sono disponibili risultati già noti e gli algoritmi non vengono addestrati: ciò è spesso indicato come apprendimento non super-

visionato (*unsupervised learning*).

2.1.1) Classificazione

La classificazione permette di trovare le caratteristiche comuni tra un insieme di oggetti in un database e li raggruppa in classi differenti (in accordo con il modello di classificazione). Per costruire tale modello, una parte del database è utilizzata come insieme di addestramento, nel quale ogni tupla oltre agli altri attributi possiede un attributo categorico (etichetta) che indica la classe alla quale appartiene. L'obiettivo della classificazione è, in una prima fase, di analizzare i dati di addestramento e sviluppare il modello che descrive accuratamente ogni classe usando le caratteristiche insite nei dati. Tali descrizioni delle classi sono poi utilizzate per classificare le altre tuple del database o future nuove tuple. Ne segue che la classificazione permette sia di comprendere meglio i dati analizzati, sia di predire quale sarà il "comportamento" di nuovi dati.

Quindi il data mining crea i modelli di classificazione dall'esame di dati già classificati, e per induzione predice il comportamento di futuri nuovi dati. I casi già classificati hanno origine da database storici o da data warehouse; oppure tali casi possono provenire da esperimenti nei quali un campione estratto dal database è testato nel mondo reale, e i risultati usati per la costruzione del classificatore. Ad esempio si può effettuare una campagna promozionale spedendo la pubblicità ad un gruppo di persone estratte casualmente dal database dell'azienda, e utilizzare le reazioni di tali persone per la realizzazione di un modello di classificazione da applicare all'intero database; in alternativa, un esperto classifica un campione del database e tale classificazione viene usata per creare un modello da applicare all'intero database.

I modelli di classificazione sono realizzati nella maggior parte dei casi tramite alberi di decisione e reti neurali.

2.1.2) Regressione

La regressione sfrutta valori già noti per predire altri valori non ancora noti. Si usano un gruppo di variabili divise tra variabili dipendenti e indipendenti. Dato un certo insieme di tuple delle quali sono noti i valori delle variabili di entrambi i tipi, si costruisce il modello di regressione. Successivamente, avendo a disposizione un altro insieme di tuple delle quali sono noti solo i valori delle variabili indipendenti, si usa il modello per predire i valori delle variabili dipendenti.

Insieme alla classificazione la regressione si occupa di predizione, con la differenza che la prima predice variabili categoriche mentre la regressione predice variabili numeriche.

Nei casi più semplici si adoperano le tecniche standard della statistica come la regressione lineare. Sfortunatamente però in molti problemi reali i valori ignoti non sono delle semplici proiezioni lineari dei valori già noti; ciò accade soprattutto quando la variabile dipendente è legata a molte variabili indipendenti, o quando la regressione lineare è pesantemente influenzata da valori anomali (outliers). Quindi sono necessarie tecniche più complesse per predire i valori futuri, come la regressione non lineare, le funzioni a base radiale (radial basis function, RBF), le reti neurali RBF.

Le stesse tecniche spesso possono essere usate sia per la regressione sia per la classificazione. Ad esempio, il CART (Classification And Regression Trees) costruisce alberi di classificazione che classificano variabili categoriche dipendenti, e alberi di regressione che predicono variabili continue dipendenti. Ed anche le

reti neurali possono creare sia modelli di classificazione che di regressione.

2.1.3) Clustering (analisi dei gruppi)

Il clustering divide un database in gruppi distinti, senza il ricorso a dati di addestramento. L'obiettivo è di trovare gruppi che siano il più possibile differenti l'uno dall'altro, ma nel contempo i membri di ogni gruppo devono essere il più possibile simili tra loro. Diversamente dalla classificazione, all'inizio del processo di clusterizzazione non si conosce che cosa rappresenteranno i cluster o quali attributi avranno maggior influenza sulla costituzione dei raggruppamenti stessi.

Esistono moltissime tecniche per realizzare il clustering:

- *algoritmi scissori*: si basano sulla partizione dell'insieme iniziale dei dati in due sottoinsiemi, e sulle successive suddivisioni delle partizioni, in maniera ricorsiva; tali suddivisioni continuano fintantoché è soddisfatto un certo criterio di ottimizzazione;
- *algoritmi aggregativi*: i gruppi si formano aggregando le tuple "più vicine" in funzione di un qualche criterio di misura della distanza tra le tuple (k-mean, algoritmo agglomerativo);
- *Kohonen feature map*: reti neurali che hanno un funzionamento alquanto complesso.

Raggruppare è una ottima maniera di iniziare qualsiasi analisi dei dati, in quanto i cluster forniscono quella conoscenza sintetica sui dati che permette il miglior uso di essi nelle successive analisi.

Tra i vantaggi del clustering ci sono:

- apprendimento non supervisionato,
- buon comportamento con dati categorici, numerici e di testo,

- facilità di applicazione.

Tra gli svantaggi:

- sensibilità ai parametri iniziali,
- difficoltà nella scelta del criterio di misura della distanza tra le tuple,
- difficoltà nello stabilire il numero ottimale di cluster.

2.1.4) Link analisys (analisi dei legami)

Contrariamente ai modelli predittivi e alle operazione di segmentazione (clustering), che puntano a caratterizzare il contenuto di un database nel suo complesso, l'analisi dei legami cerca di stabilire i legami tra record individuali o insiemi di record. Queste relazioni sono spesso chiamate *associazioni*. Una tipica applicazione dell'analisi dei collegamenti è proprio la scoperta di associazioni.

Esistono tre specializzazioni per l'analisi dei collegamenti:

- **Scoperta di associazioni**

  Lo scopo della scoperta di associazioni è di trovare insiemi di oggetti che implicano la presenza di un altro insieme di oggetti nello stesso evento o nella stessa transazione. Un esempio classico è dato da un database in cui sono memorizzati gli acquisti dei clienti di un supermercato. Applicando la scoperta di associazioni a questo insieme di transazioni si scopriranno le affinità tra insiemi di oggetti; queste affinità sono rappresentate tramite le *regole di associazione*.

  Generalmente una regola ha la forma: "se X allora Y" e si rappresenta con $X \Rightarrow Y$, dove X è un insieme di oggetti ed è detto *corpo* o *antecedente* o *parte sinistra*, Y è anch'esso un insieme di oggetti, distinti da X, ed è detto *testa* o *conseguente* o *parte destra*.

Gli algoritmi di associazione sono veloci ed efficienti nel ricavare le regole. Le difficoltà, piuttosto, nascono quando si deve giudicare la validità e l'importanza delle regole. A questo riguardo sono importanti due parametri: il supporto e la confidenza.

Il *supporto* indica il numero relativo di volte che appare una regola scoperta nell'insieme totale delle transazioni:

$$\text{supporto} = \frac{\text{numero di transazioni che contengono sia il corpo che la testa}}{\text{numero totale di transazioni}}$$

Il supporto è indice dell'importanza della regola rispetto alle altre estratte dallo stesso database.

La *confidenza* indica la forza della regola di associazione ed è definita dal rapporto:

$$\text{confidenza} = \frac{\text{numero di transazioni che contengono sia il corpo che la testa}}{\text{numero di transazioni che contengono solo il corpo}}$$

Le regole sono tanto più significative e importanti quanto più elevati sono il supporto e la confidenza. Di solito le regole che si ottengono appartengono a una delle tre seguenti categorie:

- regole utili,
- regole triviali,
- regole inesplicabili.

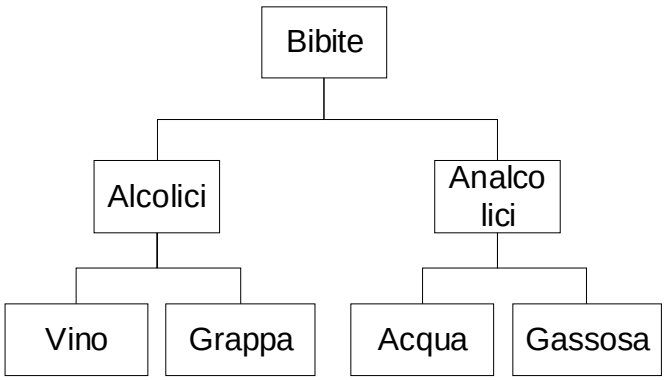
Le regole utili sono quelle contenenti informazione di buona qualità e non sono difficili da giustificare; ad esempio: "quando si compra molta farina spesso si compre anche lo zucchero".

Le regole triviali sono quelle ovvie o già ampiamente acquisite; ad esempio: "se un paziente soffre di vene varicose allora è molto probabile che il paziente

sia donna".

Le regole inesplicabili sono quelle che non hanno giustificazioni sensate e che non suggeriscono niente; ad esempio: "quando si comprano dei lacci per le scarpe spesso si comprano anche dei floppy disk".

Spesso accade che gli oggetti nel database sono moltissimi e ognuno di essi appare in relativamente poche transazioni (supporti bassi) e spesso si è interessati non a un singolo particolare oggetto (ad es. il caffè marca X) ma a categorie di oggetti (ad es. caffè). In questi casi, se la natura degli oggetti lo permette, si può ricorrere all'uso di tassonomie. Una *tassonomia* [02] generalizza e gerarchizza gli oggetti, nel senso che diversi oggetti possono essere raggruppati sotto un unico concetto, che a sua volta può essere raggruppato sotto concetti più generali, e così via, ottenendo in tal modo una gerarchia di concetti. Ad esempio una tassonomia per i prodotti in vendita nel reparto bevande in un supermercato può avere la seguente struttura:



Facendo ricorso a tali generalizzazioni, nello stesso numero di transazioni sono contenuti un minor numero di "oggetti", quindi si trovano meno regole ma più significative (con supporto più elevato).

Nel campo della vendita al dettaglio, l'utilizzo dell'elettronica per le registra-

zioni di cassa (lettori ottici dei codici a barre) ha permesso la memorizzazione in grande quantità di transazioni, e quindi ha reso possibile la scoperta di associazioni per quanto riguarda gli acquisti e le vendite: tale attività viene detta in inglese *market-basket analysis*.

La tecnica più diffusa per la scoperta di regole di associazione è l'algoritmo *Apriori*. Alcune implementazioni di algoritmi per la scoperta di regole di associazioni supportano un solo elemento nella parte destra o nella parte sinistra.

Tra i vantaggi della scoperta delle regole di associazione ricordiamo:

- spesso i risultati sono chiari e comprensibili;
- il modello non necessita di apprendimento;
- l'algoritmo è di facile comprensione.

Tra gli svantaggi:

- il numero di operazioni da eseguire varia esponenzialmente con la quantità di dati;
- è difficile decidere il numero appropriato di oggetti con cui lavorare;
- viene sminuita l'importanza degli oggetti che si presentano raramente.

Per quanto riguarda quest'ultimo inconveniente, è da sottolineare che la scoperta di associazioni funziona meglio quando tutti gli oggetti compaiono quasi con la stessa frequenza nel database usato per la ricerca.

Una estensione alle regole di associazione sono le *regole di dissociazione*, le quali coincidono con le prime, ma in più in esse si usa anche il connettore logico "e non" ("and not"): "se X e (non Y) $\Rightarrow$ W e (non Z)". Ad esempio: "se si compra farina e non zucchero allora spesso si comprano anche pelati".

Le regole di dissociazione sono generate con semplici adattamenti dagli stessi algoritmi usati per le regole di associazione, considerando come nuovi oggetti

gli inversi degli oggetti stessi: se l'oggetto X è presente in 30 transazioni su 100, allora l'oggetto "non X" sarà presente nelle restanti 70 transazioni.

- **Scoperta di sequenze**

Si tratta di un processo strettamente legato alla scoperta di associazioni; la differenza è che le relazioni tra gli oggetti devono rispettare una successione temporale, cioè si cerca di individuare insiemi di oggetti che sono seguiti temporalmente da altri insiemi di oggetti. Risulta quindi fondamentale conoscere, oltre agli oggetti di una transazione, anche il tempo nel quale è avvenuta la transazione (*campo temporale*) e chi l'ha effettuata (*campo soggetto*).

In definitiva una transazione è costituita da un campo soggetto, un campo temporale e diversi campi oggetto (*items*).

Le sequenze non sono limitate a solo due insiemi di oggetti, come per le regole di associazione ($A \Rightarrow B$), ma ne possono contenere di più: $A \rightarrow B \rightarrow C$, dove il simbolo "$\rightarrow$" sta per "segue".

Per le sequenze è definito il *supporto*: data una certa sequenza, il supporto è il rapporto tra il numero di soggetti per i quali sussiste la sequenza e il numero totali di soggetti.

Degli esempi di sequenze sono:

- quando una persona compra un martello, nei successivi tre mesi compra almeno una volta i chiodi nel 79% dei casi;
- se viene eseguita un certa operazione chirurgica ai polmoni, successivamente sopraggiunge la febbre e una infezione nel 45% dei casi;
- se il titolo A sale e l'indice MIBTEL scende, allora il titolo A salirà ulteriormente nel 68% dei casi.

Molti programmi per il data mining trattano le sequenze come associazioni per

le quali gli eventi sono collegati nel tempo.

La scoperta di sequenze ha gli stessi vantaggi e svantaggi delle analisi delle associazioni, e in più bisogna considerare che:

- occorre un elevato numero di record per assicurare che per ogni soggetto ci siano un numero significativo di transazioni;
- è richiesto un campo supplementare per rappresentare i soggetti, ma spesso i database delle varie organizzazioni, specialmente quelle che si occupano di vendite, non memorizzano informazioni sui soggetti;
- le tecniche di ricerca di sequenze funzionano meglio se i dati sono anticipatamente ordinati per soggetto e per tempo.

➢ **Scoperta di sequenze temporali simili**

La scoperta di sequenze temporali simili si occupa di trovare sequenze simili tra loro, o simili ad una fissata sequenza, in un database con dati temporali.

Per *sequenza temporale* intendiamo un insieme ordinato di valori di una variabile su un periodo di tempo. I modelli per la scoperta di sequenze temporali devono tenere in considerazione le proprietà caratteristiche del tempo, come ad esempio la gerarchia dei periodi temporali (ora, giorno, mese, stagione, anno), i periodi particolari (ad esempio giorni feriali e giorni festivi, oppure giorni in cui avvengono particolari eventi), l'aritmetica delle date.

Su un grafico bidimensionale si può rappresentare una sequenza temporale ponendo sull’asse delle ascisse il tempo per unità discrete costanti (come ad esempio settimane, mesi oppure anni) e sull’asse delle ordinate i valori della variabile (ad esempio vendite di prodotti, costi di mutui, quotazioni di titoli).

Supponiamo che una società di distribuzione al dettaglio abbia un database con le vendite effettuate. Si potrebbero individuare i prodotti o i gruppi di prodotti

che hanno un andamento delle vendite periodico, o quelli le cui vendite sono in fase, in ritardo o in anticipo rispetto alle vendite di un altro gruppo di prodotti. Sulla base di tali informazioni si potrebbero ottimizzare i rifornimenti del magazzino per l'anno o per la stagione successiva.

Fondamentale per la previsione di serie temporali è "quando" e "come" considerare due serie uguali. A causa dell'intrinseca casualità dei dati reali, il concetto di uguaglianza deve essere aggiustato per ottenere una significativa e utilizzabile definizione di similarità. Due approcci comuni a questo problema sono l'introduzione di un margine per l'errore e di un ammissibile gap di disaccordo. Il *margine per l'errore* è il numero massimo di oggetti in posizione corrispondente in due sequenze che possono differire pur considerando uguali le sequenze: tale valore assicura la tolleranza per l'intrinseca variabilità delle serie di dati temporali. Il gap di disaccordo è il numero di unità temporali consecutive per le quali sono ignorati i valori in disaccordo: tale valore assicura che brevi sottosequenze non concordi in due sequenze non influiscano sulla loro evidente somiglianza.

Uno svantaggio della scoperta di sequenze temporali simili sta nel fatto che bisogna impostare con cura un certo numero di parametri, e la cosa potrebbe non risultare agevole per un utente inesperto. Specificando un margine di errore troppo piccolo, sicuramente non si ottiene nessun risultato; specificandolo troppo grande, si potrebbero ottenere troppe sequenze considerate simili ma che in realtà non lo sono. Imponendo un gap di disaccordo troppo piccolo le sequenze simili potrebbero restare nascoste; imponendolo troppo grande, potrebbero venire fuori delle soluzioni non valide.

Un vantaggio è che molti pattern possono essere scoperti senza nessuna condi-

zione particolare da imporre, tranne che i dati devono essere quantitativi e dipendenti dal tempo.

2.2) Algoritmi e metodologie

Dopo aver discusso degli utilizzi del data mining e dei modelli che adotta, si introdurranno alcune tecniche per realizzarli e per risolvere i problemi connessi.

Molti produttori di software per il data mining usano delle varianti di algoritmi e di strumenti riportati nelle pubblicazioni specialistiche di statistica e di informatica, adattati agli obiettivi che si prefiggono. Ad esempio molte società informatiche vendono versioni del CART o del CHAID (due algoritmi per la classificazione ad albero) adattate per lavorare su computer paralleli. Qualche ditta ha anche sviluppato dei propri algoritmi, che non sono estensioni o perfezionamenti di approcci pubblici.

E' da sottolineare che non esiste uno strumento unico in grado di risolvere ogni tipo di problema. La natura stessa dei dati condizionerà la scelta degli strumenti: avere a disposizione una buona varietà di tecnologie e tools agevola la ricerca del migliore modello possibile.

Molti degli strumenti descritti nel seguito possono essere pensati come generalizzazioni del metodo standard di modellazione: il modello di regressione lineare. Numerosi sforzi sono stati compiuti per oltrepassare le limitazioni di questo modello base.

La caratteristica comune di molte nuove tecnologie è che il meccanismo di ricerca dei patterns è diretto dai dati più che dell'utente: le relazioni sono scovate dal software stesso, basandosi soprattutto sui dati esistenti e non soltanto sulle capacità di un analista di specificare le forme e le interazioni tra variabili.

2.2.1) Alberi di decisione (decision trees)

Gli alberi di decisione sono un modo per rappresentare una serie di regole che portano alla definizione di una classe o di un valore. E', probabilmente, il modello più semplice da comprendere. La figura 2.1 mostra un piccolo albero binario per la classificazione delle persone che chiedono un prestito (le possibili classi sono: affidabile e non affidabile).

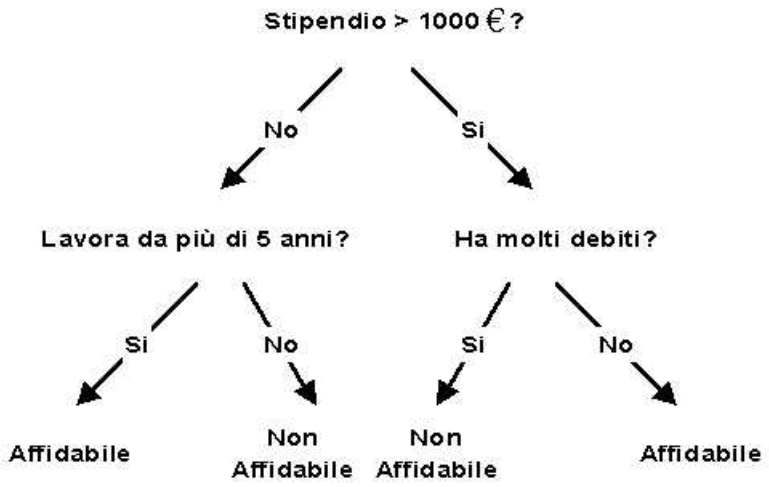


Figura 2.1 Esempio di albero di classificazione

L'albero è composto fondamentalmente da tre elementi: i nodi di decisione, i rami e le foglie. In base al tipo di algoritmo, ogni nodo può avere due o anche più rami. Il primo componente dell'albero è il nodo radice o nodo di decisione superiore, che specifica la condizione che deve essere testata. L'albero viene così diviso in due o più rami, che rappresentano le possibili risposte.

Ogni ramo porta o ad un altro nodo di decisione, o al fondo dell'albero, cioè a un nodo foglia. Al livello dei nodi foglia è nota la classe alla quale una tupla è assegnata, tupla che ha rispettato tutte le condizioni dei nodi nel cammino che va dalla

radice alla foglia stessa.

Navigando nell'albero, partendo dalla radice, si classificano le tuple sottoponendole alla condizione di ciascun nodo di decisione incontrato, finché non si raggiunge una foglia, cioè una classe. Ogni nodo usa i dati della tupla per scegliere il ramo appropriato.

Il data mining è usato per creare l'albero dall'esame dei dati e per indurre le regole che formano l'albero. Sono molti gli algoritmi che possono essere impiegati nella costruzione di un albero, e tra gli altri ricordiamo il CHAID (Chi-squared Automatic Interaction Detection), CART (Classification and Regression Tree), Quest, C5.0, C4.5.

Gli alberi di decisione si sviluppano per mezzo di divisioni iterative dei dati in gruppi discreti, con lo scopo di massimizzare la "distanza" tra i gruppi ad ogni divisione (una delle differenze principali tra i vari metodi è appunto come misurare questa distanza), mantenendo minima la distanza tra gli elementi appartenenti allo stesso gruppo. Quindi, nella ricerca delle condizioni, per ogni nodo bisogna trovare quella condizione che più mette in risalto le differenze tra i nuovi gruppi che verrebbero a formarsi, in modo tale che in ogni gruppo ci siano possibilmente solo gli elementi di una certa classe.

Gli alberi di decisione usati per la predizione di variabili categoriche sono detti *alberi di classificazione*, in quanto collocano le tuple in classi o categorie, mentre gli alberi usati per la predizione di variabili continue sono detti *alberi di regressione*.

L'albero, se è sufficientemente piccolo, risulta di immediata comprensione; però è molto difficile ottenere un albero piccolo quando viene costruito usando un database con moltissimi attributi: sebbene rimanga chiaro il ruolo di ogni nodo, l'inte-

ro albero risulta incomprensibile perché troppo complesso. Tuttavia il significato di ogni cammino radice-foglia è immediato, per cui l'albero di decisione può "spiegare" le sue predizioni (al contrario delle reti neurali), e questo è un grosso vantaggio. Questa chiarezza però può risultare ingannevole. Ad esempio la rigidità delle condizioni di decisione spesso non trova corrispondenza nel mondo reale: la differenza tra una persona che guadagna 1000 € e una che ne guadagna 1001 non è certo abissale, tuttavia nell'albero di decisione vengono considerate in modo differente. E inoltre, poiché gli stessi dati, utilizzando algoritmi diversi o lo stesso algoritmo con parametri diversi, possono essere rappresentati da alberi differenti (magari con eguale accuratezza), quali interpretazioni possono essere date per le differenti regole?

Per quanto riguarda la costruzione, gli alberi richiedono pochi passaggi sui dati (non più di un passaggio per ogni livello dell'albero), e anche in presenza di molte variabili indipendenti non sorgono difficoltà, per cui i modelli sono costruiti molto rapidamente, e ciò li rende particolarmente adatti ai grossi insiemi di dati.

Gli alberi che sono lasciati crescere senza limiti richiedono più tempo per la costruzione e diventano inintelligibili, ma soprattutto creano problemi di overfitting dei dati. La dimensione dell'albero, allora, può essere controllata tramite le regole d'arresto (*stopping rules*), che ne limitano la crescita usando determinati criteri: una semplice e diffusa regola d'arresto è la limitazione del numero massimo di livelli (profondità dell'albero); un'altra regola d'arresto consiste nel fissare il numero minimo di tuple in un nodo (se effettuando la divisione si scenderebbe al di sotto di tale valore l'algoritmo lascia intatto il gruppo rappresentato da quel nodo).

Una alternativa alle regole d'arresto è la potatura (*pruning*) dell'albero: all'albero è permesso di crescere fino alla sua massima grandezza, e poi gli vengono tolti al-

cuni rami (ricorrendo a interventi euristici integrati nell'algoritmo o ad interventi dell'utente), riducendone la dimensione senza comprometterne eccessivamente l'accuratezza. Ad esempio può essere rimosso un cammino o un sottoalbero che si ritiene insignificante perché raccoglie pochissime tuple.

Una critica comune agli alberi di decisione è che essi scelgono un criterio di divisione ricorrendo ad un algoritmo "*greedy*": nel decidere su quale variabile basare ciascuna divisione non si tiene conto degli effetti che tale scelta può avere sulle future divisioni (in altre parole, la decisione sulla divisione è presa e non è mai più rivista). In più tutte le divisioni sono eseguite in successione e quindi ognuna di esse dipende dalle precedenti. Ne segue che le soluzioni finali possono essere molto differenti tra loro se la condizione al nodo radice varia anche di poco. La ricerca sta sviluppando dei metodi per guardare in avanti, ma questi sono computazionalmente molto onerosi, per cui attualmente non si hanno a disposizione implementazioni commerciali.

Inoltre, gli algoritmi usati per la divisione si basano su condizioni espresse rispetto ad una sola variabile, cioè considerano una sola variabile indipendente per volta: questo approccio è uno dei motivi della rapidità nella costruzione del modello, ma limita fortemente la possibilità di scoprire le relazioni tra variabili apparentemente indipendenti.

Esistono algoritmi che effettuano la divisione lavorando su più variabili indipendenti: gli alberi che permettono combinazioni lineari delle variabili sono noti come *alberi obliqui*. Un criterio lineare è ad esempio: "Stipendio < 0.15 * Debiti". Un altro tipo di divisione su più variabili prevede la combinazione logica di più condizioni semplici: (Salario > 1000) OR (Anni Lavorativi > 5).

Gli alberi di decisione trattano efficacemente dati non numerici. La capacità di

operare su dati categorici riduce la quantità di trasformazioni da effettuare sui dati ed elimina l'esplosione di variabili indipendenti tipica delle reti neurali, dovuta alla necessità di trasformare una variabile categorica in una variabile dicotomica multipla.

Qualche albero di classificazione è progettato per lavorare meglio quando le variabili di ingresso sono categoriche e in questo caso le variabili continue vanno discretizzate. Ci sono poi alcuni algoritmi che non supportano variabili continue in uscita (cioè non possono costruire alberi di regressione), e in tal caso vanno discretizzate le variabili dipendenti nell'insieme di addestramento.

2.2.2) Clustering demografico

Nel clustering demografico l'algoritmo di data mining crea nuovi segmenti (*cluster*) secondo il seguente criterio: quando si presenta in input un record, si verifica che esso sia compatibile con almeno uno dei cluster già esistenti, e se ciò non accade viene creato un nuovo segmento dove il record viene inserito. Tale verifica è eseguita secondo una misura di distanza tra il record in questione ed i record appartenenti ai diversi segmenti.

La tecnica del clustering demografico si basa su un semplice principio di votazione, chiamato *Condorcet*, che misura la distanza tra i record per poterli assegnare ai cluster appropriati. Ogni record è contenuto in uno ed un solo cluster.

Coppie di record vengono comparate secondo i valori dei campi individuali: il numero di campi che hanno valori simili determina il loro livello di similarità, mentre il numero di campi che hanno valori diversi determina il loro livello di diversità. Per variabili non categoriche tale similarità è espressa attraverso un certo numero di intervalli: se due campi hanno valori all'interno dello stesso intervallo

allora sono simili, altrimenti no. Quando una coppia di record ha valori simili in corrispondenza dello stesso campo, quest'ultimo prende il voto +1, altrimenti prende –1.

Il punteggio complessivo, infine, è calcolato come la somma dei punteggi a favore e contro l'inserimento del record in un dato cluster. L'idea di base è che un record viene assegnato ad un certo cluster se il suo punteggio complessivo è maggiore rispetto al punteggio che avrebbe avuto assegnandolo in qualunque altro cluster. Se il punteggio complessivo risulta negativo per tutti i cluster, il record è candidato per essere posto in un nuovo cluster, che viene creato. Quando si verifica ciò, bisogna ricalcolare il punteggio di tutti gli altri record rispetto a questo nuovo cluster e, se qualche record risulta avere un punteggio maggiore rispetto al cluster cui era stato assegnato in precedenza, viene spostato nel nuovo cluster.

Il clustering demografico fornisce un veloce e naturale partizionamento di grandi database. Una volta che si sono formati dei cluster, la decisione di dove mettere un nuovo record non viene presa comparando ogni campo del record con ogni campo di tutti i record di ciascun cluster: per ciascun cluster si assegnano dei valori, che rappresentano le distribuzioni dei valori di ciascun campo calcolate su tutti i record contenuti nel cluster, e il confronto avviene tra i campi del nuovo record e questi valori.

A differenza del clustering neurale, che è adatto soltanto a dati numerici, il clustering demografico è particolarmente adatto per dati categorici, specialmente se il numero di categorie è basso. Inoltre esso può anche trattare variabili non categoriche, ed in tal caso l'analista deve stabilire a priori delle tolleranze. L'algoritmo usa tali tolleranze per determinare le similarità o le differenze tra due variabili: valori all'interno delle tolleranze provocano un voto a favore per l'inserimento nel

cluster, mentre valori al di fuori delle tolleranze provocano un voto contrario. La misura di similarità in questo caso non è un semplice valore binario, ma varia da 0 a 1: zero indica valori lontani, 1 indica valori identici.

2.2.3) Reti neurali

Lo sviluppo delle reti neurali avviene prevalentemente nell'ambito dell'intelligenza artificiale. Esse traggono spunto dalla biologia, e intendono simulare le reti neurali del cervello umano (anche se le reti biologiche sono incomparabilmente più complesse delle attuali corrispondenti artificiali). Le reti neurali artificiali offrono i mezzi necessari per modellare ampi e complessi problemi, con migliaia di variabili indipendenti che interagiscono tra loro in molti modi (il *teorema di Hecht-Nielsen* [02] afferma che una qualsiasi funzione vettoriale Y=F(X) può essere computata con elevata accuratezza mediante una rete neurale non ricorrente a soli tre strati, avente un opportuno numero di nodi nello strato intermedio, senza alcuna connessione all'interno dello strato ma totalmente connessa tra uno strato e l'altro; tale teorema ha validità più teorica che pratica).

Le reti neurali sono usate nei problemi di classificazione (quando l'output è composto da variabili categoriche), per la regressione (quando l'output è continuo) e per il clustering (reti neurali con funzioni caratteristiche di Kohonen).

Una rete neurale artificiale è composta da un insieme di *nodi* (neuroni); ogni nodo è raggiunto in ingresso da diverse connessioni, mentre l'uscita è unica, anche se da essa possono partire più di una connessione ad altri nodi (fig. 2.2).

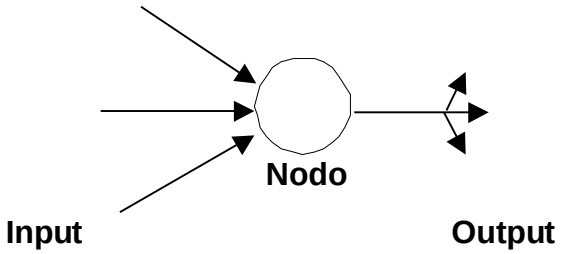


Figura 2.2 Nodo (o neurone) artificiale

Ogni nodo è un piccolo centro di elaborazione delle informazioni pervenutegli: praticamente applica una certa funzione di trasferimento ai segnali di ingresso fornendo un segnale di uscita. I segnali di ingresso ad un nodo provengono da altri nodi o sono i segnali di ingresso della stessa rete neurale, mentre il segnale di uscita raggiunge altri nodi o è uno dei segnali di uscita della rete . Sia i segnali di ingresso che di uscita sono dei numeri reali.

L'elaborazione effettuata dal nodo i-esimo consiste nel calcolo di una somma degli ingressi pesata con i pesi attribuiti a ciascuna connessione; quello che si ottiene è il *valore di attivazione* del nodo:

$$A_i = \sum_{j \in I_i} W_{ij} * O_j$$

dove I_i è l'insieme degli indici delle connessioni che entrano nel nodo i-esimo, O_j è l'uscita (output) del nodo j-esimo, W_{ij} è il peso della connessione che va dal nodo j-esimo al nodo i-esimo.

I *pesi* W_{ij} delle connessioni sono dei valori reali che indicano quanto il nodo j influisca fortemente sul nodo i: se $W_{ij} = 0$ vuol dire che non esiste connessione tra i nodi i e j, mentre se $W_{ij} < 0$ il nodo j influisce negativamente sul nodo i. I pesi sono i parametri comportamentali della rete: variando i pesi si varia il comporta-

mento della rete.

Una volta noto il valore di attivazione A_i del nodo i-esimo, ad esso viene applicata una certa funzione, detta *funzione di attivazione*, che produce l'uscita O_i:

$$O_i = T(A_i)$$

Le funzioni di attivazione sono funzioni a soglia o simili (vedi fig. 2.3).

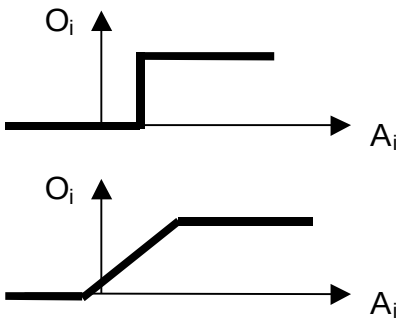


Figura 2.3 Esempi di funzioni di attivazione

I nodi che ricevono segnali dall'esterno (ingresso) saranno detti *nodi di input* della rete, i nodi che forniscono i risultati della rete all'ambiente esterno saranno detti *nodi di output* della rete.

L'input della rete è un insieme X di numeri reali (vettore) e l'output è anch'esso un insieme Y di numeri reali. Ma le reti neurali sono applicabili anche in presenza di variabili categoriche: in tal caso bisogna procedere alla trasformazione di ogni variabile categorica in una variabile dicotomica multipla, cioè occorre trasformare i valori categorici in vettori numerici formati da tutti 0 e un solo 1 (il vettore contiene tanti elementi quanti sono i possibili valori che può assumere la variabile categorica). Lo stesso ragionamento vale quando l'uscita è categorica: la rete in effetti produrrà in uscita dei vettori di tutti 0 e un solo 1, e ognuno di essi corrisponderà ad un particolare valore categorico di uscita.

Il compito della rete è di fornire un output quando gli si presenta un input: in altre parole la rete realizza una funzione vettoriale F tale che Y = F(X).

Il calcolo di F avviene in questo modo: inizialmente la rete è in stato di quiete, quindi tutti i nodi sono inattivi e forniscono un output O_i nullo. Quando pervengono dall'esterno dei segnali di input X, i nodi di input vengono sollecitati e mandano un segnale agli altri nodi. Ogni nodo della rete verrà dunque attivato in maniera diversa e propagherà stimoli agli altri nodi. Le uscite fornite dai nodi di output saranno trasmessi all'esterno formando l'insieme Y dei valori di output della rete. Le cose si complicano quando nella rete ci sono connessioni cicliche e alcuni nodi si stimolano a vicenda. In tal caso possono verificarsi tre situazioni:

- la reciproca stimolazione porta ad attivare le unità di output con valori sempre differenti, e la rete fornirà un output in perenne cambiamento: *la rete non converge*. Può succedere che l'output sia in continua crescita: in questo caso la rete "esplode" (diverge).
- la reciproca stimolazione porta ad una sequenza di attivazioni che si ripetono ciclicamente, quindi anche l'output della rete sarà periodico: *la rete oscilla*.
- la reciproca stimolazione porta a una combinazione di attivazioni che non cambia più: *la rete converge* verso una soluzione Y = F(X). Questa è la situazione più interessante perchè rende la rete neurale una macchina deterministica.

Le reti si possono classificare in base alla loro struttura e alle loro caratteristiche di funzionamento:

- Reti *non ricorrenti*: reti in cui le connessioni vanno in un solo senso, dall'input all'output, cioè sono prive di cicli. Le reti che presentano cicli sono dette *cicliche*.

- Reti *totalmente connesse*: rete nelle quali ogni nodo è connesso con tutti gli altri (generalmente escluso se stesso).
- Reti *a strati*: reti i cui nodi sono organizzati in insiemi separati e disgiunti di cui uno è detto *strato di input*, un altro *strato di output* e gli altri vengono detti *strati nascosti* o *intermedi*.
- Reti *simmetriche*: reti in cui la connessione fra due qualsiasi nodi è uguale in entrambi i sensi: $W_{ij} = W_{ji}$.
- Reti *autoassociative*: reti in cui i nodi di input coincidono con quelli di output. Il compito di queste reti è di ricevere uno stimolo dall'esterno e di farlo evolvere, fornendo come risultato una versione modificata o completata dell'input ricevuto.
- Reti *stocastiche*: reti in cui vi è una certa probabilità che un nodo non venga attivato anche quando riceve stimoli.
- Reti *asincrone*: reti in cui i nodi vengono attivati non tutti contemporaneamente ma uno alla volta secondo un ordine casuale.

Le reti neurali differiscono da molti dei metodi statistici e in molte maniere. Principalmente, una rete neurale di solito ha più parametri di un modello statistico tipico. Per esempio una rete con solo 6 nodi e 9 connessioni presenta 15 parametri. Poiché i parametri sono così numerosi, e poiché le combinazioni tra essi sono numerosissime, il loro significato non è più comprensibile e la rete diventa una "scatola nera" di predizione, ossia non è possibile dare una spiegazione del perché una rete dia un certo output in corrispondenza di un dato input. Inoltre le reti neurali presentano facilmente problemi di overfitting, ossia tendono ad adattarsi troppo bene ai dati di training e meno bene agli altri.

Le reti neurali, soprattutto le back propagation, richiedono molto tempo per l'ap-

prendimento, a meno che il problema da affrontare non sia semplice; però una volta addestrata la rete fornisce le predizioni in tempi rapidissimi.

Un altro obbligo imposto dal ricorso alle reti neurali riguarda la preparazione dei dati, che non è meno onerosa che per gli altri modelli. Le reti neurali hanno la qualità che vi si possono applicare quasi tutti i tipi di dati, ma è necessaria una particolare attenzione nelle fasi di pulizia dei dati, di selezione, di preparazione e di pre-elaborazione.

Infine le reti tendono a lavorare meglio quando il problema è molto grande e il rapporto segnale-rumore (SNR) dei dati è accettabilmente alto: in situazioni di basso SNR, ossia quando i dati utili sono molto "sporchi" (numerosi dati sono errati), le reti troveranno molti falsi pattern.

Un vantaggio delle reti neurali è che possono facilmente essere implementate su computer paralleli su larga scala (massively parallel computers), dove tutti gli elaboratori simultaneamente svolgono i propri calcoli. Sono disponibili in commercio molti chip e schede ad hoc per reti neurali.

Nel seguito verranno descritti due tipi di rete neurale: il primo si usa negli algoritmi di classificazione, il secondo negli algoritmi di clustering. Chiaramente poiché la classificazione necessita della fase di apprendimento supervisionato, allora anche la rete che la realizza apprenderà in modo supervisionato; al contrario, la seconda rete imparerà senza ricorrere a degli esempi, poiché il clustering richiede apprendimento non supervisionato.

- **Reti back propagation**

  La back propagation (propagazione all'indietro) è il tipo di rete neurale che si usa negli algoritmi di classificazione ("classificazione neurale").

  L'*architettura* o *topologia* di una rete neurale è la scelta delle variabili di in-

gresso e di uscita, del numero di strati nascosti, del numero di nodi per ogni strato nascosto e del tipo di connessione. Nel progetto di una rete, o l'utente o il software devono decidere il numero di nodi nascosti e di strati nascosti, le funzioni d'attivazione e i limiti sui pesi.

La rete neurale back propagation, anche nota col termine di *rete feed-forward back propagation* (rete a propagazione all'indietro e alimentata in avanti) viene usata molto diffusamente. Nel seguito viene descritta la sua struttura.

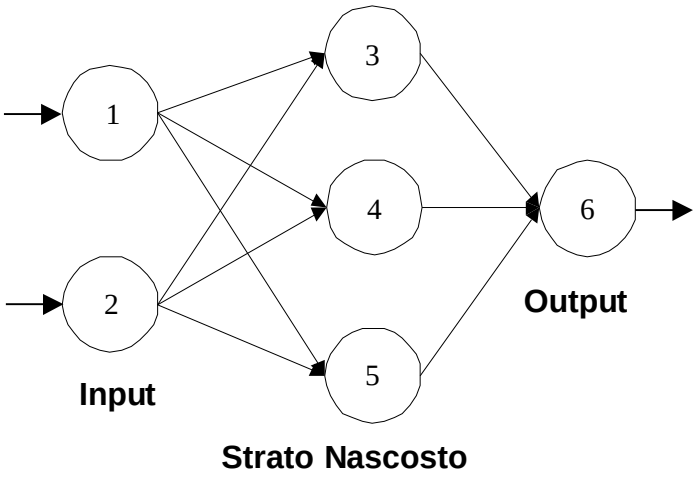


Figura 2.4 Rete neurale back propagation

Una rete neurale back propagation (fig. 2.4), nella versione base, cioè con un solo strato nascosto, inizia con un strato di input, dove ad ogni nodo corrisponde una variabile indipendente (detta anche predittore o input). Questi nodi di ingresso sono connessi ai nodi di uno strato nascosto (ogni nodo di ingresso di solito è collegato a tutti quelli dello strato nascosto). I nodi dello strato nascosto sono connessi allo strato di output. Lo strato di output rappresenta le variabili dipendenti.

La fig. 2.5 mostra come, dopo lo strato di ingresso, ogni nodo riceve un insieme di input, ognuno moltiplicato per W_{ij} (peso della connessione tra il nodo i e il nodo j), li somma, applica la funzione di attivazione a tale somma, infine passa l'output ai nodi dello strato successivo.

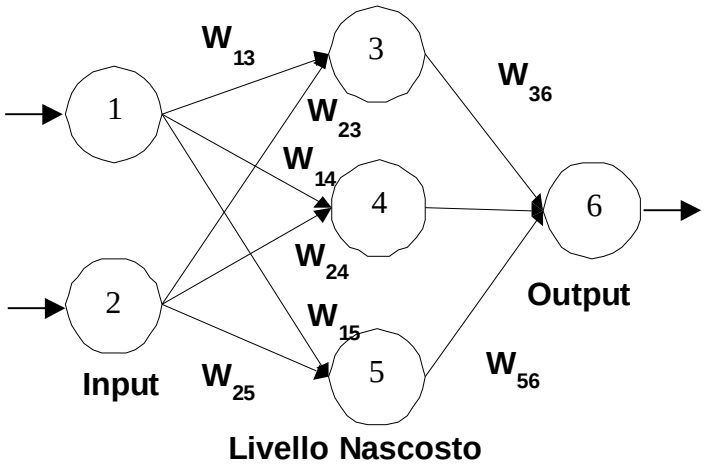


Figura 2.5 Pesi delle connessioni della rete

Le funzioni di attivazione che si usano per le reti back propagation sono le *sigmoidi*:

$$O = \frac{1}{1+e^{-A}}$$

dove A è il valore di attivazione e O è l'uscita.

Ogni nodo è o una variabile di predizione (nodi 1 e 2) o una combinazione di variabili di predizione (nodi da 3 a 6). Il nodo 6 è una combinazione non lineare dei valori dei nodi di ingresso, data la non linearità delle funzioni di attivazione. Al contrario se le funzioni di attivazione sono lineari, la rete neurale è equivalente a una regressione lineare.

La rete neurale per funzionare deve essere addestrata: vengono posti in ingres-

so i valori delle variabili indipendenti e in uscita i corrispondenti valori delle variabili dipendenti. In questa maniera vengono modificati i pesi delle connessioni, ed eventualmente alcuni parametri nelle funzioni di attivazione, che rappresentano la configurazione della rete, ovvero ciò che ha appreso.

In origine il metodo più comune di training (apprendimento) era il back propagation; altri metodi di training sono: il metodo del gradiente coniugato, quasi-Newton, Levemberg-Marquardt e algoritmi genetici. Ogni metodo di training ha un insieme di parametri che controllano vari aspetti dell'apprendimento, come ad esempio la regolazione della velocità di convergenza o l'evitare gli ottimi locali.

L'addestramento back propagation è semplicemente una variante del gradiente discendente, un tipo di algoritmo che cerca di minimizzare un valore obiettivo (l'errore, nel caso di reti neurali) ad ogni passo. Il valore del nodo di uscita è calcolato in base ai valori dei nodi d'ingresso e dei pesi iniziali. L'algoritmo procede come segue: i valori dei nodi d'ingresso sono combinati negli strati nascosti, ed i valori di questi nodi sono combinati tra loro per calcolare il valore dell'uscita; questa è la fase di *feed-forward*, cioè di alimentazione in avanti, durante la quale le informazioni viaggiano dallo strato di input verso lo strato di output. Si può avere il feed-forward solo per reti acicliche, in cui il flusso può avere una sola direzione.

L'errore nell'uscita è dato dalla differenza tra l'uscita che si è ottenuta e quella desiderata (cioè, dalla differenza tra i valori attuali e quelli che si trovano nell'insieme di training). L'errore dall'uscita è rinviato ai nodi dello strato nascosto proporzionalmente ai pesi delle connessioni: questa è la fase di *back propagation*, cioè di propagazione all'indietro. Ciò permette di calcolare l'errore, oltre

che per ogni nodo di uscita, anche per ogni nodo nascosto. Infine, l'errore di ogni nodo nascosto e di ogni nodo di uscita viene usato per aggiustare il peso delle connessioni tra di essi, in modo da ridurre gli errori.

Questo processo è ripetuto per ciascuna tupla dell'insieme di training, e più volte sull'insieme di training, finché l'errore non decresce più o diviene accettabile (le iterazioni sono chiamate *epoche*). A questo punto la rete neurale è addestrata e può iniziare a trovare i pattern nell'insieme di testing.

Poiché possono esserci molti parametri negli strati nascosti, una rete neurale con molti nodi nascosti alla fine si adatta sempre all'insieme di training. Il problema, naturalmente, è se la rete si comporterà bene anche con gli altri dati. Il pericolo è una rete neurale "*overfitted*" (super-adattata), che lavora bene solo sui dati d'addestramento; occorre quindi sapere quando interrompere l'addestramento. Alcune realizzazioni valutano la rete neurale sui dati di testing periodicamente durante la fase di addestramento: finché il tasso d'errore sull'insieme di testing decresce, l'addestramento continua. Se il tasso d'errore sui dati di testing cresce, mentre il tasso d'errore sui dati d'addestramento decresce ancora, la rete neurale potrebbe super-adattarsi ai dati; se, proseguendo ulteriormente l'addestramento, non si riduce l'errore sull'insieme di testing, allora si deve tornare a un modello precedente.

- **Kohonen feature maps o reti auto-organizzanti**

  Le Kohonen feature maps (funzioni caratteristiche di Kohonen), dette anche reti auto-organizzanti, sono utilizzate per riconoscere i cluster nei dati. Infatti il loro scopo non è di fornire un output per un certo input, ma di ricevere degli input e classificarli: la rete costruisce in modo autonomo (senza il bisogno di esempi) una struttura che accomuna gli input simili tra loro, individuando re-

golarità e somiglianze. Esse differiscono dalle reti back propagation per due motivi: hanno una differente topologia e non è più applicabile il metodo back propagation per l'apprendimento.

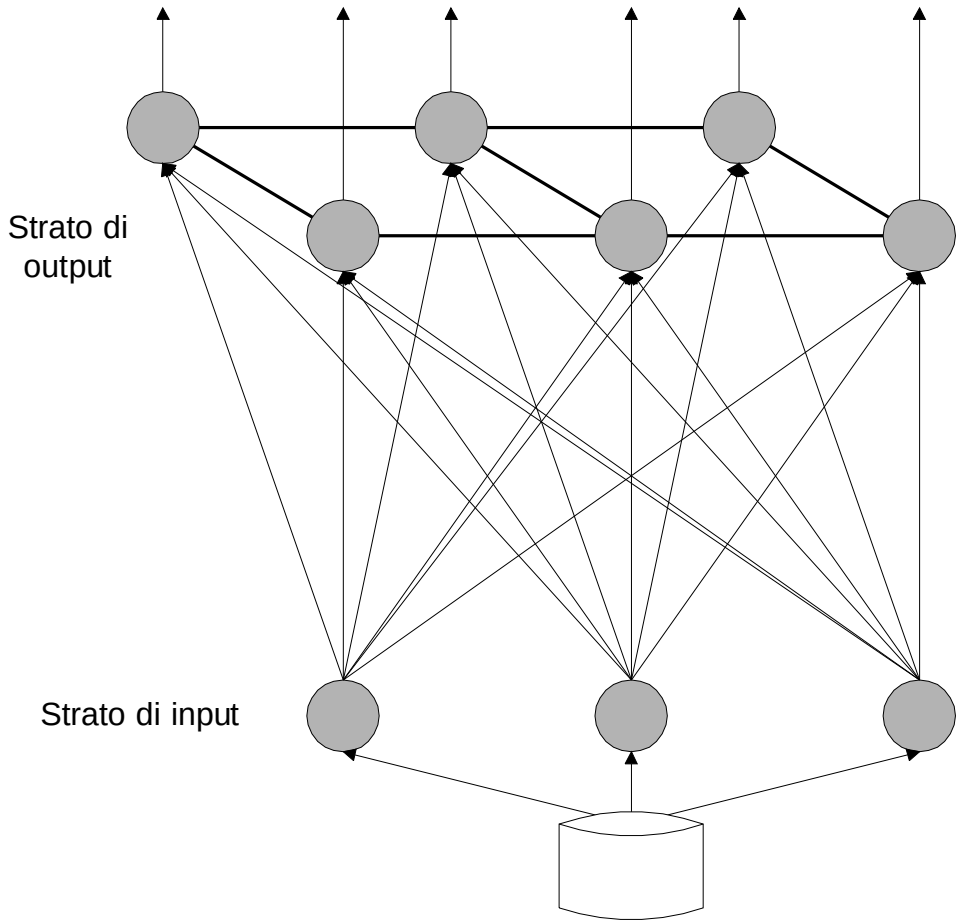


Figura 2.6 Esempio di Kohonen feature map

Una Kohonen feature map è una rete neurale a soli due strati (quello di input e quello di output) in grado di riconoscere i pattern nei dati.

Lo strato di output è formato da molti nodi (al contrario delle reti back propagation che solitamente ne possiedono pochi). Ogni nodo di output è connesso a tutti i nodi di ingresso, e lo strato di output è disposto a griglia (i nodi di output

non sono connessi con tutti gli altri dello stesso livello ma solo con quelli vicini). La struttura a griglia gioca un ruolo importante per l'addestramento della rete.

Quando un record del dataset di training è presentato alla rete, i valori fluiscono attraverso la rete verso i nodi di output. I nodi di output competono tra loro e quello col valore più alto "vince". La "ricompensa" consiste nell'aggiustare i pesi delle connessioni che conducono al nodo vincente favorendo il rafforzamento nella risposta a quel particolare input.

Non solo sono rafforzati i pesi per il nodo vincente, ma anche quelli dei nodi ad esso più vicini (sulla griglia), per favorirne la risposta a quel tipo di input. Queste ultime modifiche sono controllate da un parametro detto *parametro di vicinanza* che regola la distanza degli altri nodi dal nodo vincente, e determina l'entità delle variazioni per i pesi delle connessioni.

In definitiva, ogni volta che viene presentato in ingresso un determinato tipo di input si fortifica sempre la stessa zona di uscita della rete. Man mano che la rete esamina i record, si evidenzia sempre più che tuple simili devono far vincere gli stessi nodi, in quanto appartengono allo stesso cluster. Non tutte le uscite della rete rappresentano un cluster, anzi il numero di cluster è molto più piccolo del numero di nodi di output; qualora il meccanismo di limitazione dell'influenza di un nodo vincente sul suo vicinato non fosse efficiente, si otterrebbe un cluster per ogni nodo di output. I cluster simili tra loro hanno le rispettive uscite più prossime sulla griglia, mentre cluster molto differenti hanno le uscite più distanziate.

Una volta addestrata la rete, la si può utilizzare per clusterizzare nuovi record. Va sottolineato il fatto che la rete identifica l'appartenenza di un record a una

certa classe, ma non spiega perché i membri di un cluster sono ritenuti simili tra loro e perché i cluster sono diversi tra loro. Una maniera per capire cosa accomuna i membri di un cluster è di calcolare, se possibile, la media su essi e ritenere tale media una rappresentazione o una descrizione del cluster.

2.2.4) Algoritmo Apriori

I metodi visti finora sono *globali*, cioè si applicano all'intero dataset a disposizione. Un'altra classe molto importante di metodi computazionali riguarda l'insieme di tutti quei metodi che, invece, sono *locali*, ossia riguardano parti selezionate del dataset, nella forma di sottoinsieme di variabili o di unità statistiche. Hand, Mannilla e Smyth utilizzano per i primi il termine di *model* (modello), e per i secondi il termine di *pattern* (struttura) [01].

Esempi rilevanti di questi ultimi sono la scoperta di regole associative (tra cui figura l'algoritmo Apriori) e le tecniche di ragionamento basato sulla memoria.

Vediamo in dettaglio il funzionamento dell'algoritmo Apriori.

Sia $I = \{ i_1, i_2, ..., i_N \}$ un insieme di oggetti (ogni oggetto è fondamentalmente una etichetta, un valore categorico). Sia D un insieme di *transazioni*, dove ogni transazione T è un insieme di oggetti tale che $T \subseteq I$. Si noti che il numero di volte che un oggetto appare in una transizione è ininfluente, cioè quello che conta è che un dato oggetto sia presente o meno nella transazione (ad esempio, quando si comprano 12 uova e 2 litri di olio, quello che veramente interessa è il fatto che si siano acquistati uova e olio). Ogni transazione è individuata da un codice identificatore.

Indicando con X un insieme di oggetti (*itemset*), si dice che la transazione T contiene X se e solo se $X \subseteq T$. Una regola di associazione è una implicazione del tipo

$X \Rightarrow Y$, dove $X \subset I$, $Y \subset I$ e $X \cap Y = \varnothing$.

Il *supporto* per la regola $X \Rightarrow Y$ si ottiene dividendo il numero di transazioni che soddisfano tale regola per il numero totale di transazioni presenti nell'insieme D.

La *confidenza* per la regola $X \Rightarrow Y$ si ottiene dividendo il numero di transazioni che soddisfano tale regola per il numero di transazioni che contengono l'insieme X.

La confidenza indica la precisione della regola, cioè quante volte, essendo presente X, è presente anche Y; il supporto invece indica l'importanza della regola rispetto alle dimensioni del database. Spesso è desiderabile rivolgere l'attenzione solo a quelle regole che hanno un supporto ragionevolmente alto. Le regole con confidenza e supporto elevati sono dette *regole forti*.

Lo scopo di qualsiasi algoritmo per la scoperta di regole è essenzialmente quello di trovare regole forti in database di grandi dimensioni. Tale obiettivo di solito si raggiunge dividendo il problema in due passi:

- scoperta di *large itemset*, cioè di insieme di oggetti aventi supporto più elevato di un predeterminato supporto minimo s_{min},
- utilizzo dei large itemset per generare le regole di associazione.

L'algoritmo Apriori è iterativo: ripete per un numero finito di volte (al massimo m-1, dove m è il numero di oggetti della transazione più grande) i due passi appena descritti. Ad ogni iterazione costruisce un insieme di large itemset candidati, conta il numero di occorrenze di ogni large itemset candidato tra le transazioni (ossia valuta il supporto di ognuno), determina i large itemset effettivi (quelli il cui supporto è maggiore del supporto minimo s_{min}), estrae le regole da ogni large itemset. Tra tutte le regole ottenute da un large itemset si escludono quelle che non raggiungono una certa confidenza minima decisa in precedenza.

L'algoritmo Apriori è chiaramente estensibile alle tassonomie, ottenendo regole generalizzate e gerarchiche.

Esistono diverse varianti di tale algoritmo, tra le quali: quelle che permettono di ridurre il numero di scansioni del database; quelle che tramite campionamento permettono di analizzare immensi insiemi di transazioni ricorrendo a un compromesso tra accuratezza ed efficienza; quelle che si adattano ad ambienti distribuiti; quelle che permettono la manutenzione delle regole scoperte quando i dati del database vengono modificati (ad esempio per la presenza di nuove transazioni).

2.2.5) Ragionamento basato sulla memoria

Le tecniche di *Memory Based Reasoning* (ragionamento basato sulla memoria) si basano su due momenti strettamente collegati: l'individuazione di osservazioni di interesse (eventualmente sulla base dell'esperienza passata) e, successivamente, la ricerca nel database di tutte le osservazioni simili ad esse.

Anche questo è un esempio di pattern locale, che riguarda la selezione delle osservazioni, anziché delle variabili come invece accade in alcune delle regole associative.

I problemi metodologici di questo metodo sono inerenti all'utilizzo di valide misure di prossimità e di distanza per identificare osservazioni "simili".

I vantaggi collegati all'impiego di queste metodologie sono l'ottenimento di risultati comprensibili e applicabili a diverse tipologie di dati. Tra gli svantaggi c'è l'esigenza di un'ampia capacità di memoria e di calcolo, nonché una certa soggettività legata alla scelta del tipo di funzione da adottare per la ricerca delle somiglianze.

2.2.6) Funzioni a base radiale

Il metodo delle *funzioni a base radiale* (RBF, *radial basis functions*) è una tecnica per la predizione di valori che si è dimostrata più robusta e flessibile del tradizionale approccio della regressione, lineare e non lineare. Il metodo RBF lavora ricorrendo non ad una singola funzione non lineare ma ad una somma pesata di molte funzioni non lineari, dette appunto funzioni a base radiale. Una funzione a base radiale è una funzione non lineare che quantifica la distanza da un centro, ed è per questo motivo che è detta "a base radiale" : essa assume lo stesso valore per ogni punto avente la stessa distanza (o raggio) dal centro. Nello spazio delle variabili indipendenti (input) le RBF distinguono delle regioni, in ognuna delle quali le variabili dipendenti assumono quasi gli stessi valori. Per ogni regione ottenuta viene costruito un centro che predice il valore medio della regione.

Terminato di costruire il modello su valori noti delle variabili dipendenti, lo si può sfruttare per la predizione vera e propria: per ogni nuovo tupla di variabili indipendenti vengono predetti i valori delle variabili dipendenti tramite una media pesata delle predizioni di tutti i centri, dove il peso di ogni centro decade rapidamente se è molto distante dalla tupla in esame.

2.2.7) Algoritmi genetici

Come le reti neurali, anche gli algoritmi genetici sono basati su analogie con i processi biologici, ed in particolare con la teoria evoluzionistica, che lega le possibilità di sopravvivenza principalmente alle capacità di adattamento all'ambiente circostante: i membri di una generazione (di modelli) entrano in competizione per passare le proprie caratteristiche alla generazione successiva, in modo che siano propagati i fattori genetici degli elementi migliori.

Gli algoritmi genetici applicano la stessa idea: lavorano su una popolazione di individui (ciascuno dei quali rappresenta una possibile soluzione del problema), e al termine del processo permettono di individuare il modello migliore.

Quindi gli algoritmi genetici non vengono impiegati nella ricerca diretta di pattern, ma piuttosto nella guida del processo di apprendimento dei metodi di data mining, ad esempio nelle reti neurali. Essenzialmente essi agiscono come un metodo di ricerca guidata di buoni modelli nello spazio delle soluzioni.

Gli algoritmi genetici sono spesso molto importanti in fase di pre-processing: consentono, per esempio, di gestire il processo di scelta del modello in un modo molto flessibile, in quanto non hanno ipotesi modellistiche sottostanti. Ad esempio, nella costruzione di una rete neurale, gli algoritmi genetici possono sostituirsi al back propagation nella regolazione dei pesi; i cromosomi in questo caso conterranno i pesi. Oppure gli algoritmi genetici possono essere usati per trovare la migliore architettura della rete, e i cromosomi conterranno il numero di strati nascosti e di nodi per ogni strato.

Se da un lato gli algoritmi genetici sono un interessante approccio per l'ottimizzazione dei modelli, dall'altro aggiungono un forte surplus computazionale, e spesso è necessaria una lunga codifica e programmazione del problema prima di poter applicare l'algoritmo.

CAPITOLO 3

UN SOFTWARE COMMERCIALE

Il notevole interesse che si è sviluppato e continua a svilupparsi nei riguardi del data mining ha favorito la nascita di numerosi software, che spesso si differenziano in molteplici aspetti: alcuni implementano procedure e algoritmi per realizzare l'intero processo di KDD, altri si occupano più specificamente del solo passo di data mining; alcuni sono generici e sono utilizzabili in qualsiasi campo di indagine, ma solitamente sono in grado di ricevere i dati solo in taluni formati, altri invece sono realizzati per uno specifico campo (ad esempio il marketing, o la genomica) e sono in grado, talvolta, di ricevere i dati da analizzare direttamente da strumenti scientifici (senza alcuna pre-elaborazione); di alcuni viene fornito anche il codice sorgente, consentendo così ad un utente esperto di effettuare modifiche per adattarlo meglio alle proprie esigenze. Infine l'ultima ma non meno importante distinzione che si può effettuare è tra software freeware e software commerciali.

Dato che il prezzo di un software non è una informazione sufficiente a caratterizzarlo, si è scelto di analizzare un esempio di software a pagamento e un esempio di software ad uso gratuito, al fine di evidenziarne potenzialità, funzionalità, vantaggi e svantaggi, ponendo l’accento soprattutto sulla loro flessibilità e semplicità d’uso piuttosto che sull’accuratezza e sulla precisione dell’analisi, che richiederebbero un ben più lungo e approfondito esame.

Come esempio di software commerciale si è scelto PolyAnalyst 4.5, realizzato dalla società Megaputer e disponibile in rete per una valutazione gratuita di 30 giorni (previa registrazione presso la società produttrice).

3.1) Introduzione a PolyAnalyst 4.5

Fondata nel 1993, Megaputer è una società leader nel campo della scoperta di conoscenza, e fornisce diversi prodotti per il data mining, il text mining e la web data analysis. Tra i suoi clienti annovera numerose grandi società ed organizzazioni, tra cui: Siemens, McKinsey, DuPont, Boeing, 3M, The US Navy, National Cancer Institute, France Telecom.

3.1.1) Caratteristiche tecniche

PolyAnalyst è un ambiente per data mining che implementa alcuni tra i più recenti metodi sviluppati nell'ambito del KDD. Implementa sedici algoritmi in grado di effettuare una analisi diretta dei dati e presentare l'informazione estratta dai database in forma simbolica:

- Classify;
- Cluster;
- Decision Tree;
- Decision Forest (effettua un'analisi multicategorica costruendo un insieme di alberi in competizione tra loro, uno per ciascuna classe selezionata);
- Discriminate;
- Find Dependencies;
- Find Laws;
- Linear Regression;

- Link Analysis;
- Link Terms;
- Market Basket Analysis;
- Memory Based Reasoning;
- PolyNet Predictor (basato su un algoritmo di rete neurale);
- Summary Statistics;
- Text Analysis;
- Text Categorization.

Alcuni dei sopraelencati algoritmi sono stati modificati dal team della Megaputer, rispetto alle versioni classiche riportate nella letteratura specialistica, per incrementarne l'efficacia e la funzionalità.

In particolare l'algoritmo *Find Laws*, che rappresenta uno dei più avanzati sistemi di esplorazione dei dati, utilizza una tecnologia proprietaria denominata SKAT™ (Symbolic Knowledge Acquisition Technology): trova automaticamente dipendenze e leggi nascoste nei dati, e le presenta esplicitamente in forma di regole ed algoritmi; *Find Laws* coniuga la potenza di analisi di una rete neurale con la semplicità di comprensione (delle leggi trovate) tipica del linguaggio naturale. Tutto ciò torna molto utile in tutti quei casi in cui non è sufficiente sapere che esiste una certa regola, ma occorre sapere perché esiste e come essa operi.

Per rappresentare le conoscenze acquisite PolyAnalyst usa un linguaggio algoritmico universale denominato SRL (Symbolic Rule Language), usato sia in input per esprimere nuovi attributi basati sulla trasformazione funzionale di attributi presenti nei dati, sia in output per rappresentare le leggi trovate. SRL è in grado di esprimere formule, funzioni matematiche, funzioni logiche, espressioni condizionali, generazione di numeri casuali e manipolazione di attributi temporali.

Tutti i sistemi implementati in PolyAnalyst sono contenuti in DLL (dynamic link libraries) e fanno uso di controlli ActiveX accessibili anche da altre applicazioni, come direct marketing tool, spreadsheet e data warehouse: ciò consente una agevole integrazione di PolyAnalyst nei sistemi aziendali preesistenti.

PolyAnalyst si presenta come un tipico ambiente a finestre, in cui una interfaccia grafica (GUI) consente di operare con semplicità sui dati mediante un meccanismo del tipo point-and-click.

I dati di input possono provenire da un file di testo, da un foglio elettronico Microsoft Excel, o direttamente da database relazionali e data warehouse (sono supportati ODBC, OLE DB, Oracle Express e IBM Visual Warehouse). Inoltre i modelli costruiti possono essere esportati in formato XML.

PolyAnalyst è disponibile per piattaforma Windows 9x/2000/NT/XP, ma per il funzionamento dell'algoritmo *Find Laws* è necessario un sistema operativo Windows 2000/NT/XP.

Per poter utilizzare PolyAnalyst è necessario un computer con le seguenti caratteristiche minime:

- processore Pentium o superiore (raccomandabile almeno 600 MHz);
- 128 MB di RAM (raccomandabili 256 MB o più);
- 30 megabyte di spazio su hard disk;
- scheda video SVGA;
- mouse;
- Microsoft Internet Explorer 5 o successive versioni.

Un processore più potente e una maggior quantità di RAM incrementano la velocità delle elaborazioni compiute dal software. PolyAnalyst supporta anche i sistemi multiprocessore.

La documentazione fornita con il software consiste in un *help on line* che illustra il funzionamento di ogni sua parte, e in un *tutorial* che illustra passo passo l'utilizzo di PolyAnalyst in una decina di differenti applicazioni.

3.1.2) Acquisto ed installazione

La versione di valutazione del software, scaricabile dal sito della società produttrice, consta di un unico file eseguibile, che installa in maniera automatica PolyAnalyst 4.5. Inizialmente il software ha una limitazione d'uso di 5 giorni, e quasi tutte le funzioni sono disabilitate. Per usufruire appieno delle sue funzionalità è necessario che il computer su cui viene installato possieda un collegamento ad Internet. All'avvio del software una finestra mostra la procedura guidata di registrazione, che consiste nella compilazione di una scheda con dati personali e un indirizzo valido di posta elettronica; successivamente il programma provvede ad inviare le informazioni della scheda e altre informazioni relative al computer in uso al server della casa madre (lo scopo è evitare che la versione di valutazione venga installata più di una volta sullo stesso computer); il server risponde con una e-mail contenente il codice per l'attivazione. Quando si inserisce il codice ricevuto, il collegamento ad Internet deve essere attivo: il software accetta la registrazione solo dopo essersi collegato al server della casa madre per le opportune verifiche. Da questo momento in poi si hanno a disposizione 30 giorni di valutazione, e il software è quasi pienamente funzionante: le uniche limitazioni riguardano la dimensione del dataset da analizzare (in generale fino a 15000 record, ma gli algoritmi di *Text Analysis*, *Link Analysis* e *Text Categorizer* possono essere usati al più con 500 record) e le funzionalità di esportazione dei dati e dei modelli costruiti.

L'acquisto di questo ed altri software prodotti dalla Megaputer avviene interamente via web, e comporta semplicemente una registrazione da postazione remota: dopo aver scaricato ed installato la versione di valutazione, è sufficiente mettersi in contatto via web con la società per acquistare il numero di licenze desiderato ed eventualmente il servizio di supporto tecnico avanzato, e procedere alla registrazione nello stesso modo in cui avviene la registrazione della versione di valutazione.

I prezzi vanno da 4000 $ fino ad oltre 12000 $ per licenza. Sono previsti sconti per usi in ambito accademico ed educativo (è anche previsto un apposito Megaputer Educational Program).

L'acquisto di una licenza comprende un servizio base di supporto tecnico (Standard Customer Support), che consta di:

- consultazione telefonica iniziale sul data mining con PolyAnalyst (3 ore o 1 progetto);
- supporto tecnico telefonico o tramite e-mail gratuito per un mese;
- 80% di sconto sull'upgrade del corrispondente prodotto Megaputer nei 3 mesi successivi alla data di rilascio dell'aggiornamento.

E' altresì possibile acquistare un servizio avanzato di supporto tecnico (Gold Customer Support), che comprende:

- Hot Line per il supporto tecnico gratuito tramite telefono o e-mail;
- aggiornamento gratuito del software nell'ambito della versione corrente;
- 80% di sconto sull'upgrade alla versione successiva del prodotto Megaputer acquistato;
- aggiornamento gratuito della corrispondente documentazione;
- partecipazione gratuita ai seminari per utenti.

Il costo di questo servizio avanzato di supporto tecnico per 1 anno è pari al 18% del prezzo della corrispondente licenza, per 2 anni è pari al 34% del prezzo della corrispondente licenza.

3.2) Casi d'uso

Come visto nel Cap. 1, il processo di KDD si compone di numerosi passi, e il data mining è solo uno di questi passi. Volendo porre l'enfasi su quest'ultimo, si trascureranno i passi relativi alla preparazione e alla pulizia dei dati, e pertanto si useranno dei set di dati (o *dataset*) già pronti all'uso, disponibili nel web (ad esempio sul sito dell'UCI [24]) per scopi didattici o per testare il funzionamento di software per data mining.

Il software è stato installato su di un computer con le seguenti caratteristiche:

- processore AMD Athlon 900 MHz;
- hard disk da 13 GB;
- 128 MB di memoria RAM;
- Microsoft Windows 98 Second Edition.

In fig. 3.1 si riporta la finestra di avvio del programma, e in fig. 3.2 la finestra che compare nella fase di caricamento dei dati.

Nel seguito si illustra l’utilizzo di PolyAnalyst 4.5 in tre tipici campi di applicazione del data mining.

Figura 3.1 Finestra che appare all'avvio di PolyAnalyst 4.5

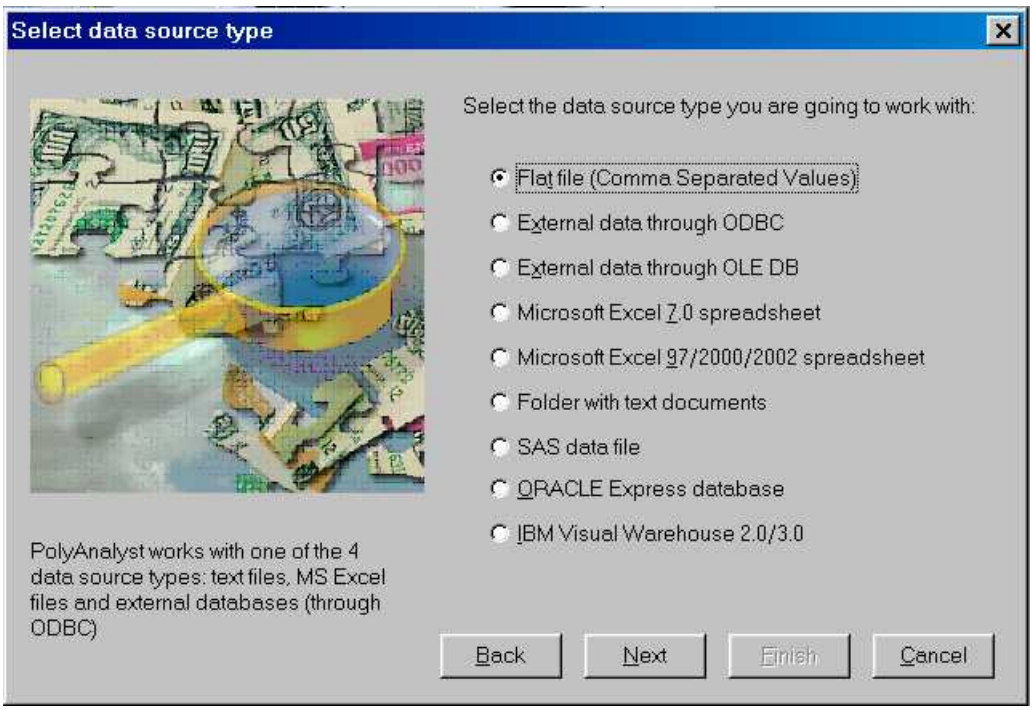


Figura 3.2 Finestra per la selezione del tipo di file sorgente

3.2.1) Data mining in campo economico

- Dataset utilizzato: credit-g
- Provenienza: UCI Repository [24]
- Autori: Professor Dr. Hans Hofmann
  Institut fur Statistik und Okonometrie
  Universitat Hamburg
  FB Wirtschaftswissenschaften
  Von-Melle-Park 5
  2000 Hamburg 13
- Descrizione dei dati: i dati rappresentano diversi attributi di un generico cliente bancario tedesco. Lo scopo è distinguere i buoni clienti (quelli che restituiscono i prestiti) dai cattivi (quelli che non restituiscono i prestiti). Nel dataset si consiglia di preferire che un cliente buono venga classificato come cattivo piuttosto che uno cattivo venga classificato come buono.
- Algoritmi utilizzati: Decision Tree, Decision Forest.
- Risultati e commenti:

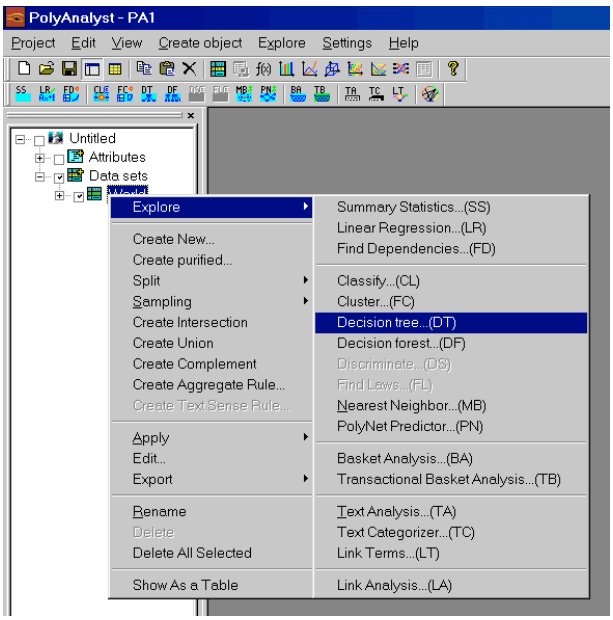


Figura 3.3 La scelta del tipo di algoritmo da applicare ai dati

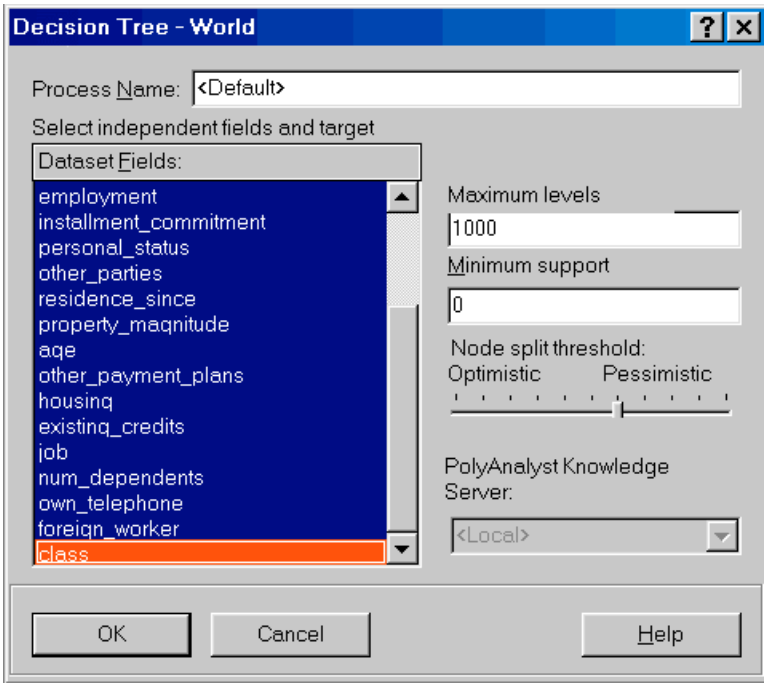


Figura 3.4 La scelta dell'attributo target

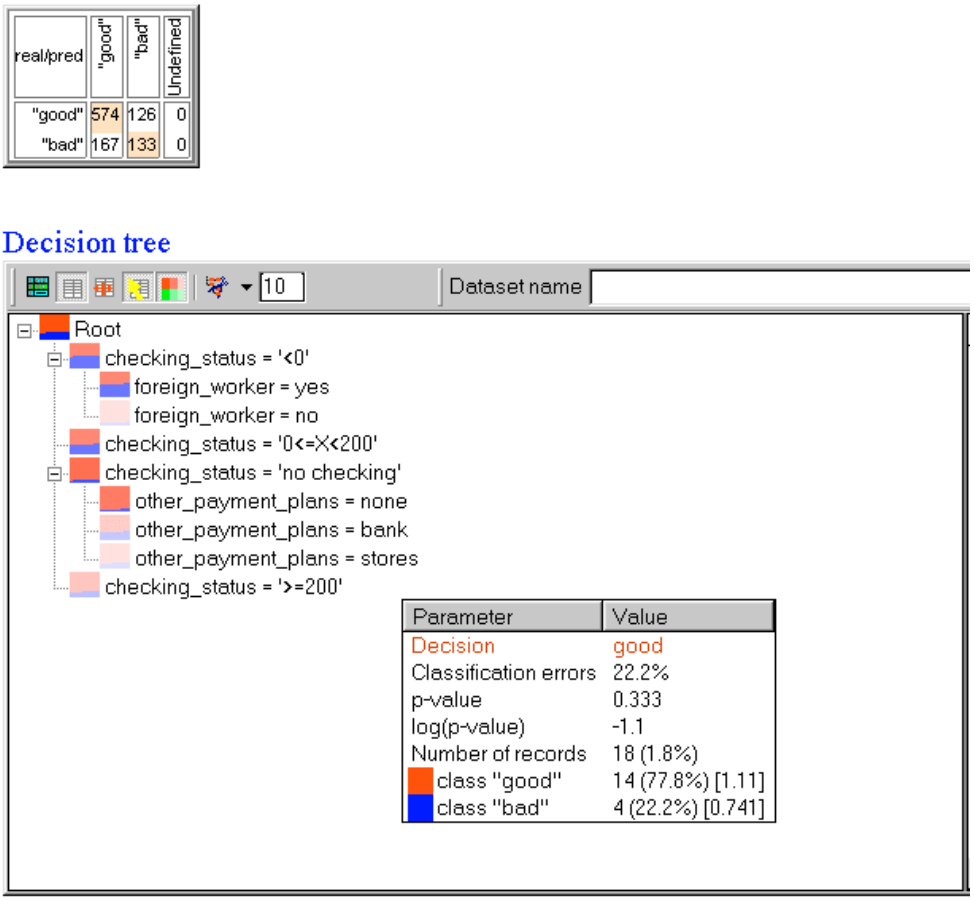


Figura 3.5 Risultati dell'elaborazione effettuata con l'algoritmo Decision Tree

real/pred	"good"	"bad"	Undefined
"good"	561	139	0
"bad"	131	169	0

Decision tree

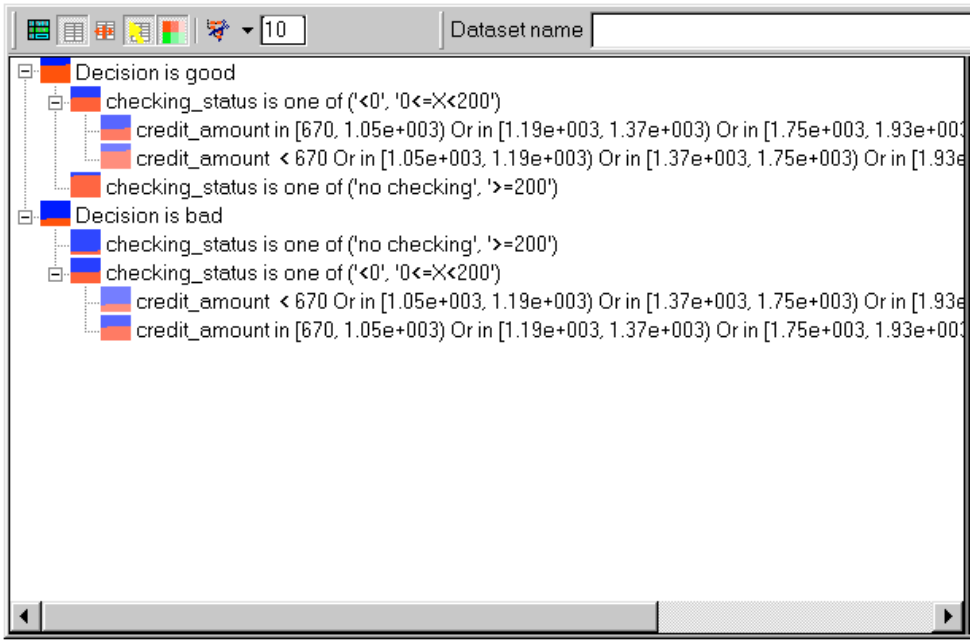


Figura 3.6 Risultati dell'elaborazione con l'algoritmo Decision Forest

Come appare evidente dalle figg. 3.5 e 3.6, in entrambi i casi PolyAnalyst ha costruito alberi semplici da interpretare, ma il tasso di errore è piuttosto elevato, e non è possibile migliorare tale situazione (la fig. 3.4 mostra che i parametri a disposizione dell'utente per influenzare la costruzione dell'albero sono davvero pochi).

Scendendo nel dettaglio, i risultati dell'algoritmo Decision Tree sono:

```
Tree depth  1000
Number of non-terminal nodes:  3
Number of leaves:  7
Depth of constructed tree:  2
Total classification error:  29.30%
%% of undefined prediction cases:  0.00%
Classification probability:  70.70%
Classification efficiency:  2.33%
Classification error for class "good":  18.00%
Classification error for class "bad":  55.67%
```

mentre i risultati dell'algoritmo Decision Forest sono:

```
Tree depth  1000
Number of non-terminal nodes:  4
Number of leaves:  6
Depth of constructed tree:  2
Total classification error:  27.00%
%% of undefined prediction cases:  0.00%
Classification probability:  73.00%
Classification efficiency:  10.00%
Classification error for class "good":  19.86%
Classification error for class "bad":  43.67%
```

3.2.2) Data mining in campo medico

- Dataset utilizzato: breast-cancer
- Provenienza: UCI Repository [24]
- Autori: Matjaz Zwitter & Milan Soklic (physicians)
  Institute of Oncology
  University Medical Center
  Ljubljana, Yugoslavia
  Date: 11 July 1988
- Descrizione dei dati: i dati rappresentano diversi attributi di pazienti donne operate per cancro al seno. La classe (attributo target) distingue i casi in cui si è avuta o non si è avuta ricomparsa del cancro. Di seguito è riportato un frammento del dataset:

```
age;menopause;tumor_size;inv_nodes;node_caps;deg_malig;breast;breast_quad;irradiat;Class
'40-49';'premeno';'15-19';'0-2';'yes';'3';'right';'left_up';'no';'recurrence-events'
'50-59';'ge40';'15-19';'0-2';'no';'1';'right';'central';'no';'no-recurrence-events'
'50-59';'ge40';'35-39';'0-2';'no';'2';'left';'left_low';'no';'recurrence-events'
'40-49';'premeno';'35-39';'0-2';'yes';'3';'right';'left_low';'yes';'no-recurrence-events'
'40-49';'premeno';'30-34';'3-5';'yes';'2';'left';'right_up';'no';'recurrence-events'
'50-59';'premeno';'25-29';'3-5';'no';'2';'right';'left_up';'yes';'no-recurrence-events'
'50-59';'ge40';'40-44';'0-2';'no';'3';'left';'left_up';'no';'no-recurrence-events'
'40-49';'premeno';'10-14';'0-2';'no';'2';'left';'left_up';'no';'no-recurrence-events'
```

- Algoritmi utilizzati: Classify, Link Analysis.
- Risultati e commenti:

L'algoritmo *Classify* non ha prodotto alcun risultato, in quanto richiede che l'attributo target sia di tipo booleano, quindi sarebbe stato necessario effettuare una trasformazione dell'ultimo attributo (class).

I risultati dell'algoritmo *Link Analysis* sono riportati nelle figg. 3.7, 3.8 e 3.9.

Si noti che in gradazioni di rosso sono rappresentati i link "positivi", mentre in gradazioni di blu sono rappresentati i link "negativi".

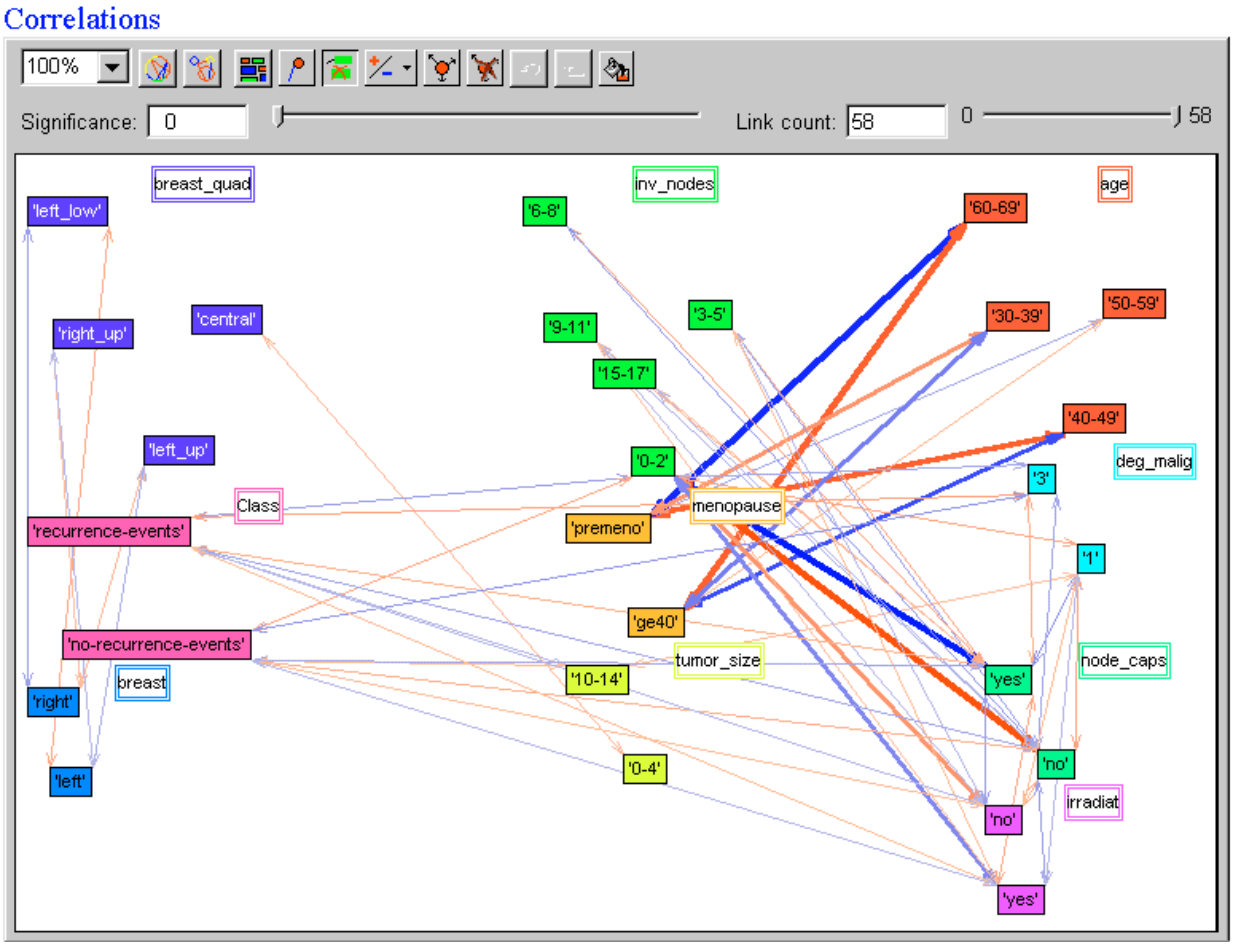


Figura 3.7 Rappresentazione grafica dei risultati

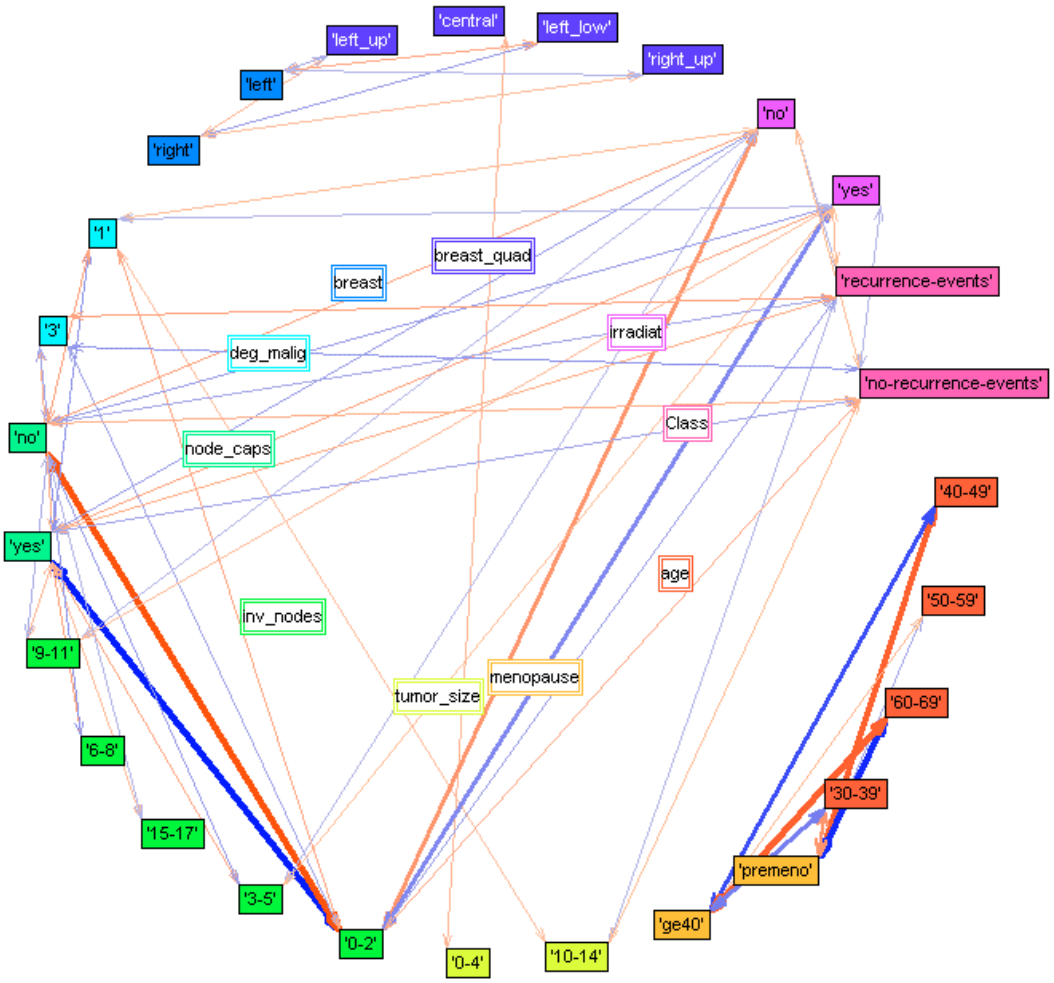


Figura 3.8 Tutti i gruppi sono mostrati come parte di un cerchio

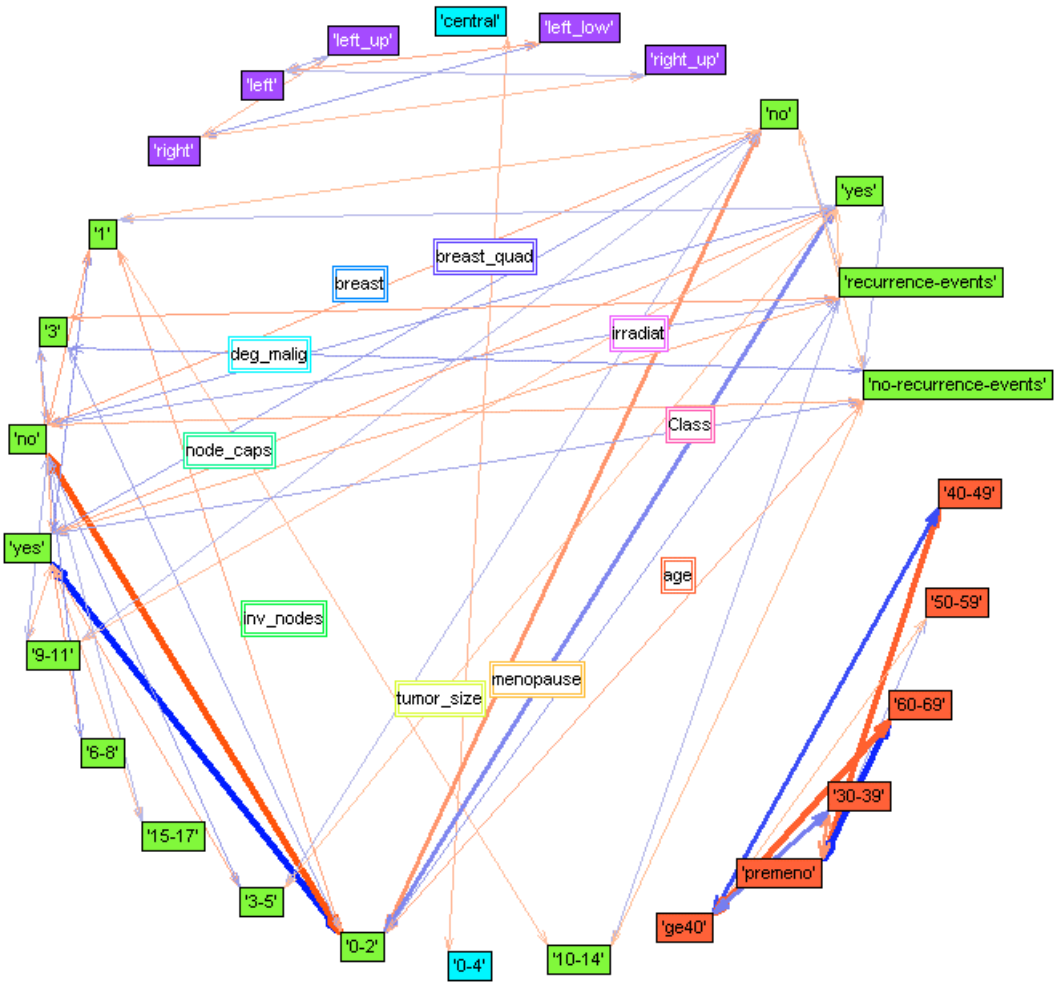


Figura 3.9 Ogni sottografo connesso viene visualizzato con un diverso colore

3.2.3) Data mining in campo tecnico

- Dataset utilizzato: cpu-performance
- Provenienza: UCI Repository [24]
- Autori: Phillip Ein-Dor and Jacob Feldmesser
  Faculty of Management; Tel Aviv University; Ramat-Aviv;
  Tel Aviv, 69978; Israel
  Date: October, 1987
- Descrizione dei dati: i dati rappresentano diversi attributi di cpu costruiti da differenti aziende. La classe è di tipo numerico ed è un indice della performance di ciascuna cpu. Di seguito è riportato un frammento del dataset:

```
vendor;MYCT;MMIN;MMAX;CACH;CHMIN;CHMAX;PRP(published_relative_performance)=class
adviser;125;256;6000;256;16;128;199
amdahl;29;8000;32000;32;8;32;253
apollo;400;512;3500;4;1;6;24
basf;60;2000;8000;65;1;8;70
amdahl;29;8000;32000;32;8;32;253
burroughs;110;3100;6200;0;6;64;45
c.r.d;320;128;6000;0;1;12;28
hp;75;3000;8000;8;3;48;54
hp;175;256;2000;0;3;24;20
harris;300;768;3000;0;6;24;23
ibm;1500;768;2000;0;0;0;20
ipl;50;8000;16000;48;1;10;128
honeywell;140;2000;4000;8;1;20;32
ibm;57;4000;16000;1;6;12;82
ibm;57;4000;24000;64;12;16;171
magnuson;100;1000;8000;0;2;6;37
ibm;800;768;2000;0;0;0;20
siemens;26;8000;32000;64;12;16;275
siemens;26;8000;32000;128;24;32;382
sperry;116;2000;8000;32;5;28;56
```

- Algoritmi utilizzati: PolyNet Predictor.
- Risultati e commenti:

I risultati dell'algoritmo *PolyNet Predictor* (l'algoritmo di rete neurale di PolyAnalyst) sono riportati in fig. 3.10 e nelle successive righe.

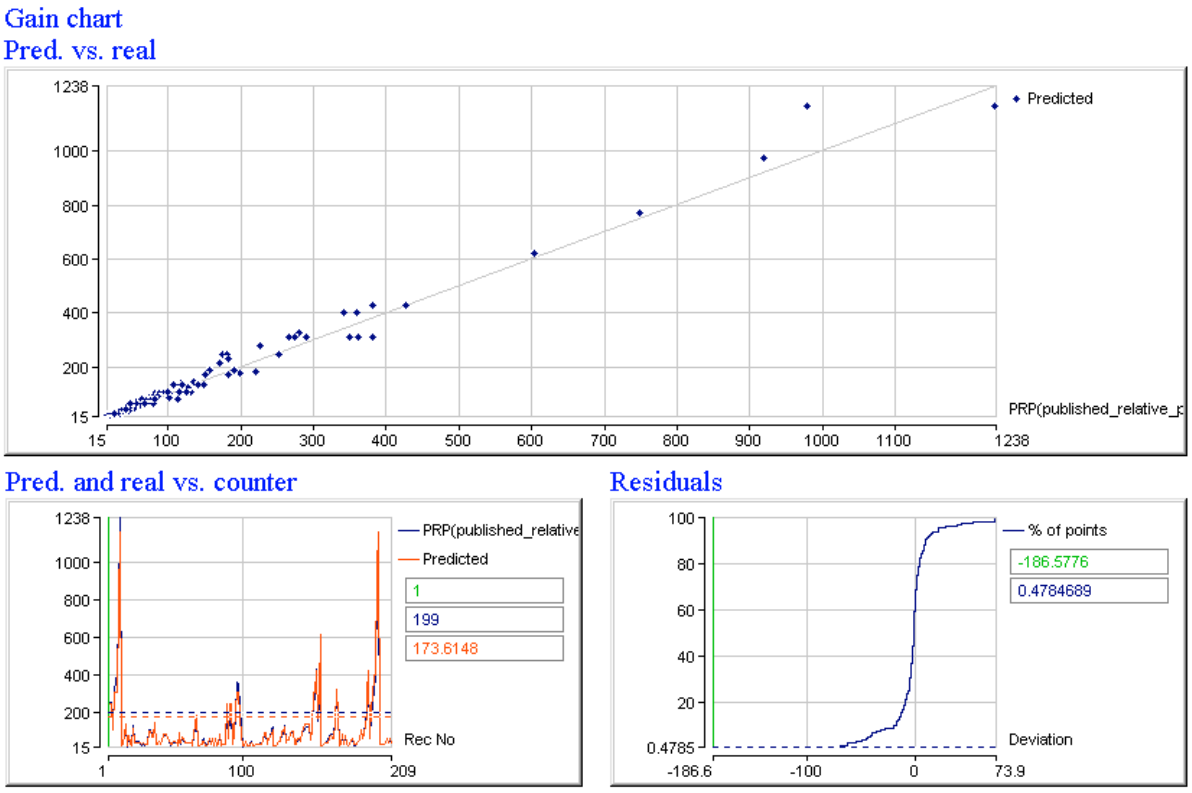


Figura 3.10 Risultati dell'algoritmo *PolyNet Predictor*

```
Exploration parameters:
Argument                                Value
Critical F-Ratio                        1
Minimum part of predicted values,       % 50
Order                                   3

Significance index:  24.18
Standard error:  0.1461
R-squared:  0.9786
Standard deviation:  22.61
Points processed:  209
Number of network layers:  1
Number of network nodes:  3

Result rule:
(17.6949 +MMAX*(0.00115805+MMAX*(1.76237e-007+-1.78255e-012*MMAX))
+CACH*(0.58374+CACH*(+-5.20136e-008*MMAX) +MMAX*(+1.52208e-009*MMAX)))
```

Come si può vedere, il risultato dell'analisi è riassunto in un'unica formula.

3.3) Considerazioni finali

PolyAnalyst 4.5 si è rivelato uno strumento molto potente e flessibile, anche se il suo costo, certamente non irrisorio, ne consiglia l'uso quasi esclusivamente in ambito professionale, piuttosto che per sporadiche applicazioni o per analisi piccole e semplici.

CAPITOLO 4

UN SOFTWARE FREEWARE

Così come è accaduto in molti altri campi di applicazione, anche nel campo del data mining la diffusione e il successo hanno favorito lo sviluppo di numerosi software freeware.

Alcuni sono stati realizzati per un ben determinato utilizzo, e successivamente sono stati messi a disposizione della comunità mondiale. Altri sono stati sviluppati proprio con l'intento di consentirne l'uso in tutti o quasi i campi del data mining, per cui in essi sono stati implementati il maggior numero possibile di algoritmi.

Si è scelto uno di questi software freeware per confrontarne prestazioni e funzionalità con il software a pagamento visto nel capitolo precedente. La scelta è caduta su WEKA.

4.1) Introduzione a WEKA

WEKA è un progetto software per data mining sviluppato nell'università di Waikato in Nuova Zelanda: WEKA è un acronimo che sta per "*Waikato Environment for Knowledge Analysis*". Tutti gli algoritmi implementati nel software sono liberamente disponibili nel web: lo scopo principale è consentire una piena comprensione di come lavorano e di che cosa fanno.

4.1.1) Caratteristiche tecniche

WEKA è scritto in Java, in quanto il software realizzato con questo linguaggio di programmazione orientato agli oggetti è automaticamente portabile in tutte le principali piattaforme per computer. Il funzionamento di WEKA è stato testato su Linux, Windows e Macintosh; grazie alle caratteristiche di portabilità del linguaggio Java, WEKA presenta una interfaccia uniforme e un identico funzionamento su tutti questi diversi sistemi operativi.

Inoltre WEKA è un software Open Source e viene rilasciato con licenza *GNU General Public License*: ciò consente l'utilizzo e la modifica del codice sorgente (disponibile insieme al software).

WEKA concentra il suo interesse principalmente sugli algoritmi di classificazione e sui filtri per la pre-elaborazione dei dataset (entrambi presenti in gran numero), ma include anche l'implementazione di un algoritmo per la scoperta di regole di associazione (l'algoritmo Apriori, il più noto) e di qualche algoritmo per il clustering e per la regressione.

Tra i principali schemi (*learner*) figurano:

- Apriori
- C4.5 decision tree
- Cobweb and Classit clusterer
- decision table majority classifier
- EM (estimation maximization) clusterer
- HyperPipe classifier
- K-nearest neighbour classifier
- K* classifier
- Id3 decision tree classifier

- linear regression
- Locally-weighted regression
- logistic regression
- Naive Bayes classifier
- Neural Network back propagation classifier
- PART decision list
- PRISM classifier
- 1R classifier
- voting feature interval classifier
- voted perceptron

Il package `weka.filters` offre un utile supporto alla pre-elaborazione dei dati (*data preprocessing*), che come detto è un importante passo del KDD. Questo package contiene numerose classi Java in grado di operare trasformazioni sull'insieme dei dati, come ad esempio la rimozione o l'aggiunta di attributi, il rimescolamento del dataset, la rimozione di determinate tuple, ed altro ancora.

I filtri possono suddividersi in *supervisionati* e *non supervisionati*: nel primo caso fanno uso dell'informazione aggiuntiva dovuta al conoscere l'attributo target, mentre nel secondo caso trattano tutti gli attributi nello stesso modo.

Un'ulteriore suddivisione è in *filtri d'attributo* e *filtri di tupla*: nel primo caso operano sugli attributi, mentre nel secondo operano sulle tuple.

WEKA inoltre contiene dei *Meta-learning schemes*, ossia metodi che non implementano direttamente un algoritmo per data mining, ma consentono di migliorare le performance dei *learner*.

I principali sono:

- Stacking;

- Bagging;
- AdaBoostM1;
- LogitBoost;
- MultiClassClassifier;
- CVParameterSelection.

Il metodo CVParameterSelection, ad esempio, consente di variare i parametri di un *learner* in un intervallo di valori, e ne prova tutte le combinazioni possibili, scegliendo tramite Cross Validation il miglior modello ottenuto.

4.1.2) Installazione e funzionamento

Sul sito del progetto WEKA [19] si trovano tre versioni del software:

- **WEKA 3.0.6**: è una versione stabile e testata con funzionamento da linea di comando, legata al libro sul data mining [06] scritto dagli ideatori e coordinatori del progetto WEKA.
- **WEKA 3.2.3**: è una versione GUI (Graphical User Interface), cioè ha una interfaccia grafica a finestre stile Windows; anche questa versione, come la precedente, è stabile e testata.
- **WEKA 3.3.6**: è l'ultima versione GUI disponibile, ma è meno stabile delle due precedenti in quanto ancora in fase di sviluppo e testing.

I coordinatori del progetto WEKA hanno seguito il modello di Linux per quanto riguarda il rilascio di nuove release del software: la seconda cifra pari sta ad indicare una versione stabile, mentre invece la seconda cifra dispari sta ad indicare una versione ancora in fase di sviluppo.

Per poter funzionare, tutte le precedenti versioni richiedono che nel computer in cui viene installato WEKA sia già installata almeno la versione 1.2 dell'ambiente

di sviluppo Java (*Java Runtime Environment*).

Dato che per scopi didattici è consigliato l'utilizzo di una release stabile, si è deciso di far riferimento alla seconda versione (WEKA 3.2.3), nel seguito indicata anche come WEKA GUI o semplicemente WEKA. Tale versione, all'avvio, consente la scelta di tre diversi ambienti (vedi fig. 4.1):

- **Simple CLI**: è una semplice interfaccia a linea di comando che consente una esecuzione diretta dei comandi di WEKA.
- **Explorer**: è un ambiente che consente di "esplorare" i dati utilizzando tutti i comandi di WEKA (filtri, *learner schemes*, *Meta-learning schemes*).
- **Experimenter**: è un ambiente che consente di compiere esperimenti e test statistici tra i diversi algoritmi per data mining (*learner schemes*).

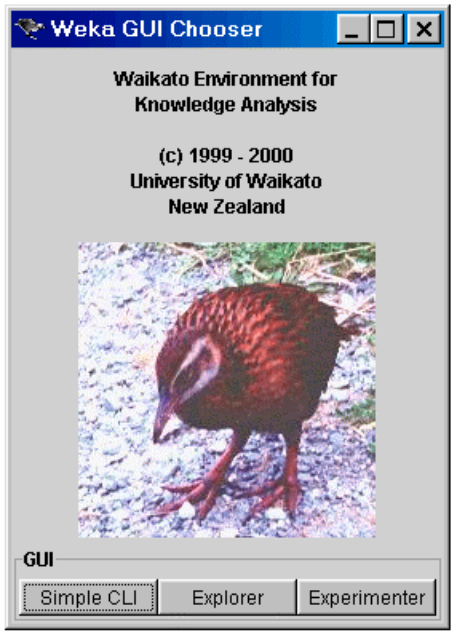


Figura 4.1 Finestra di avvio di WEKA 3.2.3

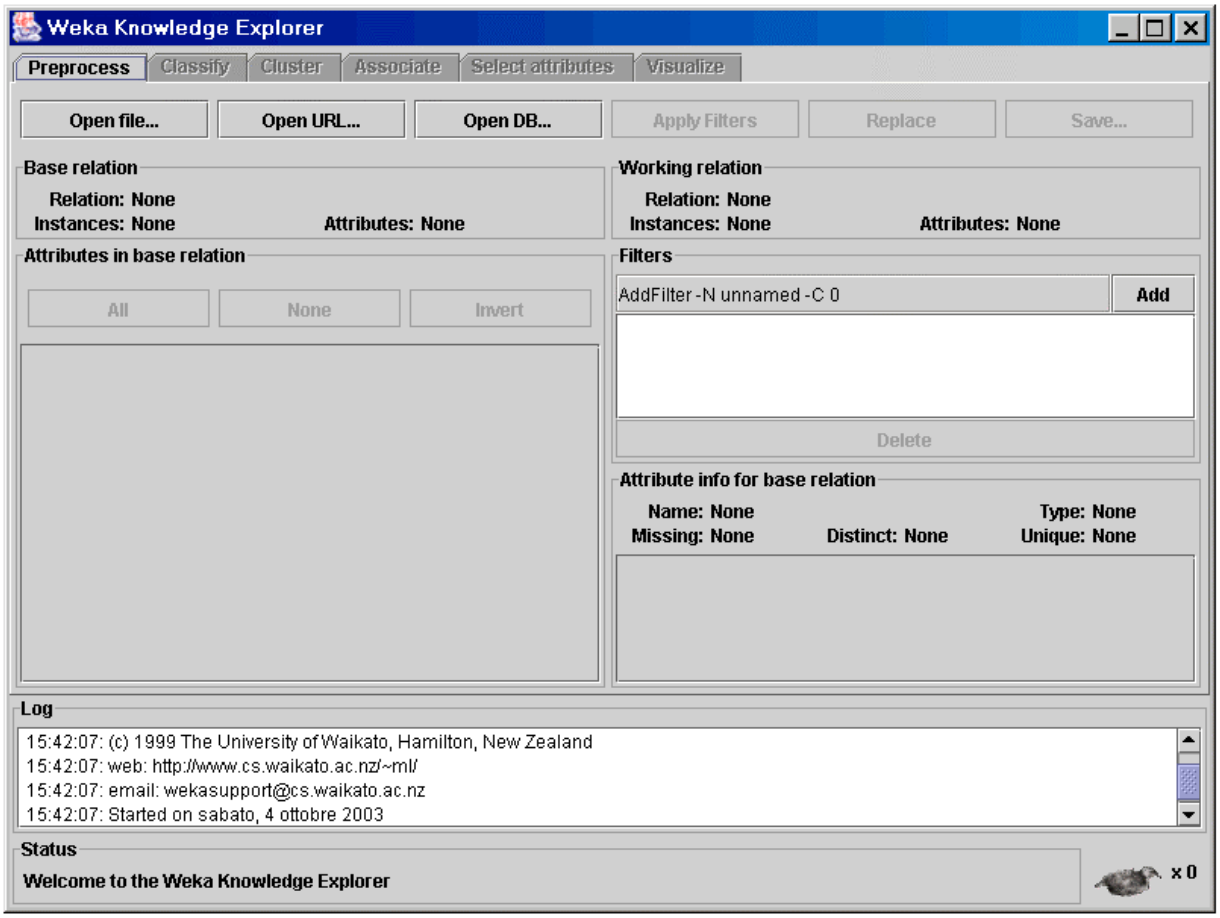


Figura 4.2 L'ambiente Explorer

WEKA è abbastanza complesso da utilizzare, e la documentazione fornita con il software è alquanto scarsa. Dal sito [19] è liberamente scaricabile il capitolo 8 (*tutorial.pdf*) del libro sul data mining [06] scritto dai coordinatori del progetto WEKA. Tale libro associa la teoria e la discussione degli algoritmi sul data mining ad esempi di applicazioni realizzate con WEKA. Il capitolo 8 è una introduzione alle funzionalità di WEKA, ma fa esclusivamente riferimento alla versione da linea di comando. In esso è anche riportato un esempio di riutilizzo delle classi Java di WEKA: viene mostrato come si può realizzare una applicazione Java per text mining che può fungere da filtro per bloccare e-mail pubblicitarie indesiderate (spam).

Sempre sullo stesso sito si trovano una guida all'ambiente Explorer (*Explorer-*

Guide.pdf) e una guida all'ambiente Experimenter (*Experiments.pdf*). E' presente inoltre una breve introduzione ai principali concetti e comandi di WEKA realizzata da Alex Seewald [20].

Infine, insieme al software si trova la documentazione generata usando Javadoc, utile a quanti intendono riutilizzare le classi di WEKA nel proprio software, ma per sapere con più esattezza cosa fa esattamente una certa classe è meglio leggerne il codice sorgente.

Per quanto riguarda i dataset da analizzare, anche se WEKA è in grado di importare i dati di input da un file di testo in formato CSV (*comma separated*) o direttamente da un database (tramite driver JDBC-ODBC), il formato standard per i suoi input è un file di testo in formato **.ARFF** (*Attribute-Relation File Format*) [19], che è un file di testo in formato ASCII contenente una lista di record caratterizzati dagli stessi attributi, e organizzati nella forma di "matrice dei dati". Esso consiste in una intestazione (*header*) in cui sono descritti il tipo degli attributi ed è riportato il nome del dataset, e in un corpo contenente i dati separati da virgole.

4.2) Casi d'uso

Come nel capitolo precedente, l'attenzione sarà focalizzata principalmente sul processo di data mining, per cui si useranno dataset già pronti e non si farà ricorso ai pur numerosi filtri presenti in WEKA, né ai *Meta-learning schemes*.

Il computer utilizzato presenta la seguente configurazione:

- processore AMD Athlon 900 MHz;
- hard disk da 13 GB;
- 128 MB di memoria RAM;
- Microsoft Windows 98 Second Edition;

- Java 2 Runtime Environment SE v1.4.1_02.

Nel seguito, quindi, viene illustrato l'utilizzo dell'ambiente *Explorer* di WEKA 3.2.3 in tre tipici campi di applicazione del data mining, facendo ricorso agli stessi dataset utilizzati nel capitolo precedente, al fine di mettere in maggiore evidenza affinità e differenze nelle analisi effettuate dai due software.

4.2.1) Data mining in campo economico

- Dataset utilizzato: credit-g
- Provenienza: UCI Repository [24]
- Autori: Professor Dr. Hans Hofmann
  Institut fur Statistik und Okonometrie
  Universitat Hamburg
  FB Wirtschaftswissenschaften
  Von-Melle-Park 5, 2000 Hamburg 13
- Descrizione dei dati: i dati rappresentano diversi attributi di un generico cliente bancario tedesco. Lo scopo è distinguere i buoni clienti (quelli che restituiscono i prestiti) dai cattivi (quelli che non restituiscono i prestiti). Nel dataset si consiglia di preferire che un cliente buono venga classificato come cattivo piuttosto che uno cattivo venga classificato come buono, ossia questo secondo errore ha un peso maggiore rispetto al primo.
- Algoritmi utilizzati: C4.5 decision tree (J48).
- Risultati e commenti:

Grazie a questo dataset si può dimostrare come la possibilità di variare numerosi parametri nell'algoritmo J48 (vedi fig. 4.3) consenta di costruire il modello più adatto alle proprie esigenze.

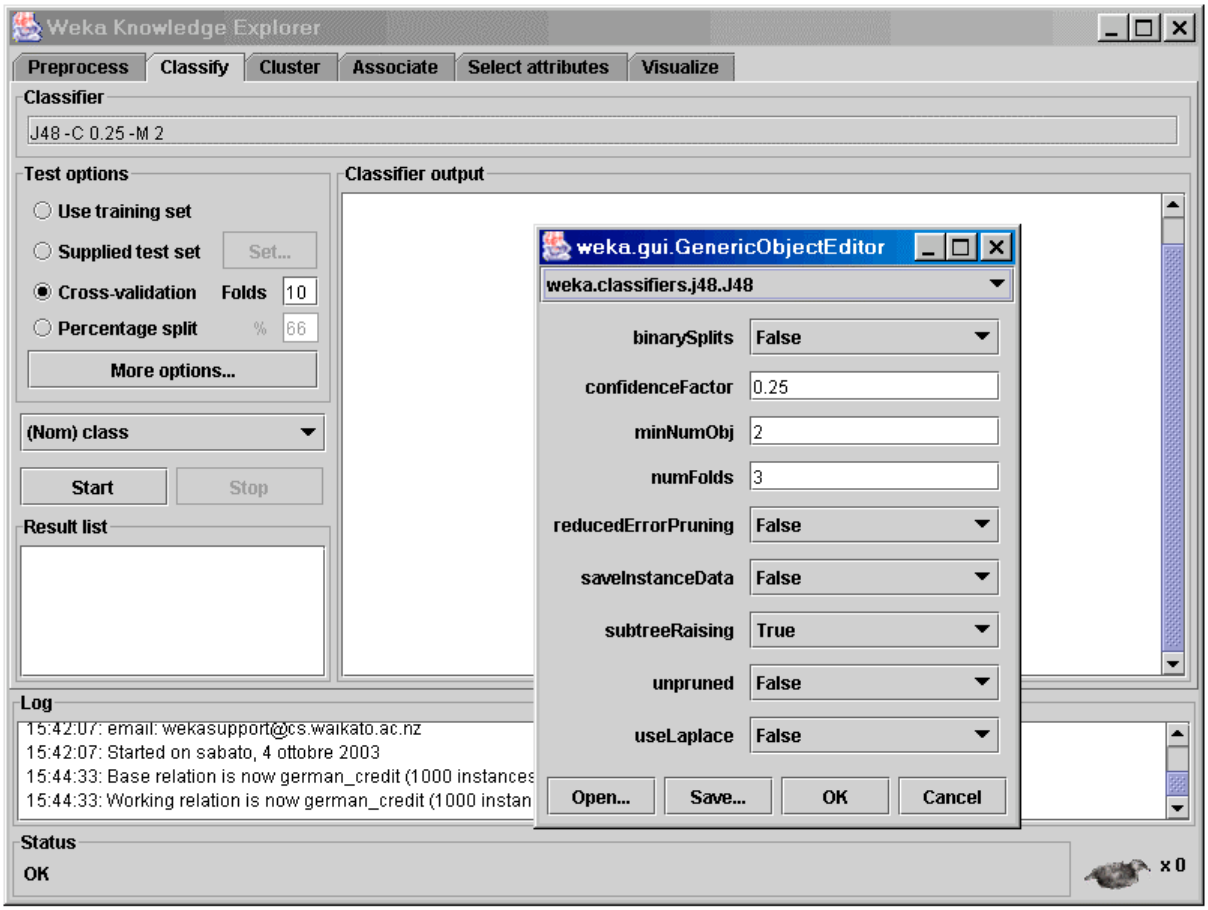


Figura 4.3 L'editor dei parametri dell'algoritmo J48

In particolare si è operato sui seguenti parametri:

- minNumObj: minimo numero di record che devono essere presenti in ogni foglia dell'albero;
- reducedErrorPruning (True,False): se questa variabile ha valore True, la classe J48 utilizza un algoritmo per la potatura dell'albero differente da quello standard ;
- binarySplits (True,False): se questa variabile ha valore True, la classe J48 costruisce un albero binario.

Nella prima combinazione dei parametri è stato posto: MinNumObj = 10; reducedErrorPruning = False; binarySplits = False. Il risultato è riportato in fig. 4.4 e nelle successive righe.

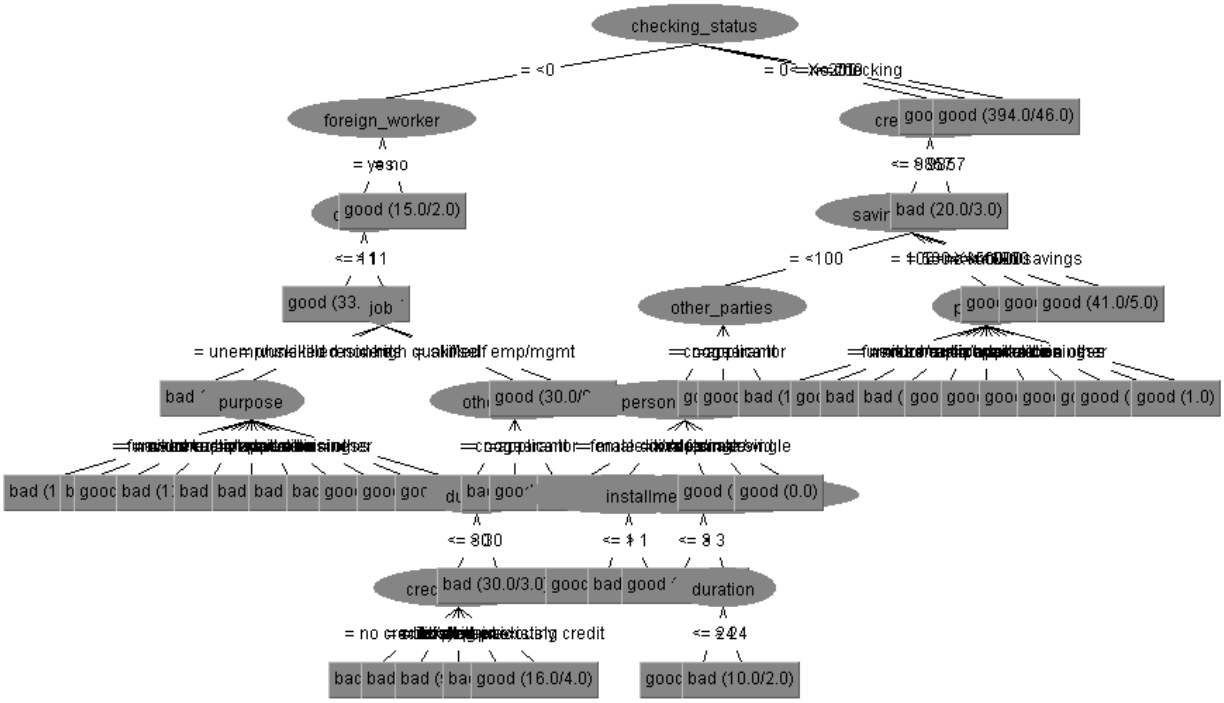


Figura 4.4 Il primo albero costruito con l'algoritmo J48

```
=== Run information ===

Scheme:       weka.classifiers.j48.J48 -C 0.25 -M 10
Relation:     german_credit
Instances:    1000
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

checking_status = <0
|   foreign_worker = yes
|   |   duration <= 11: good (33.0/7.0)
|   |   duration > 11
|   |   |   job = unemp/unskilled non res: bad (5.0/1.0)
|   |   |   job = unskilled resident
|   |   |   |   purpose = new car: bad (12.0/4.0)
|   |   |   |   purpose = used car: bad (1.0)
|   |   |   |   purpose = furniture/equipment: good (10.0/3.0)
|   |   |   |   purpose = radio/tv: bad (12.0/5.0)
|   |   |   |   purpose = domestic appliance: bad (1.0)
|   |   |   |   purpose = repairs: bad (1.0)
|   |   |   |   purpose = education: bad (1.0)
|   |   |   |   purpose = vacation: bad (0.0)
|   |   |   |   purpose = retraining: good (1.0)
|   |   |   |   purpose = business: good (3.0)
|   |   |   |   purpose = other: good (1.0)
|   |   |   job = skilled
|   |   |   |   other_parties = none
|   |   |   |   |   duration <= 30
|   |   |   |   |   |   credit_history = no credits/all paid: bad (9.0/2.0)
|   |   |   |   |   |   credit_history = all paid: bad (10.0/3.0)
|   |   |   |   |   |   credit_history = existing paid: bad (59.0/25.0)
|   |   |   |   |   |   credit_history = delayed previously: bad (5.0/1.0)
|   |   |   |   |   |   credit_history = critical/other existing credit: good (16.0/4.0)
|   |   |   |   |   duration > 30: bad (30.0/3.0)
|   |   |   |   other_parties = co applicant: bad (7.0/1.0)
|   |   |   |   other_parties = guarantor: good (12.0/3.0)
|   |   |   job = high qualif/self emp/mgmt: good (30.0/8.0)
```

```
|   foreign_worker = no: good (15.0/2.0)
checking_status = 0<=X<200
|   credit_amount <= 9857
|   |   savings_status = <100
|   |   |   other_parties = none
|   |   |   |   personal_status = male div/sep: bad (8.0/2.0)
|   |   |   |   personal_status = female div/dep/mar
|   |   |   |   |   residence_since <= 1: good (13.0/4.0)
|   |   |   |   |   residence_since > 1: bad (25.0/9.0)
|   |   |   |   personal_status = male single
|   |   |   |   |   installment_commitment <= 3: good (27.0/4.0)
|   |   |   |   |   installment_commitment > 3
|   |   |   |   |   |   duration <= 24: good (19.0/7.0)
|   |   |   |   |   |   duration > 24: bad (10.0/2.0)
|   |   |   |   personal_status = male mar/wid: good (16.0/7.0)
|   |   |   |   personal_status = female single: good (0.0)
|   |   |   other_parties = co applicant: good (2.0)
|   |   |   other_parties = guarantor: good (20.0/3.0)
|   |   savings_status = 100<=X<500
|   |   |   purpose = new car: bad (15.0/5.0)
|   |   |   purpose = used car: good (3.0)
|   |   |   purpose = furniture/equipment: bad (4.0/1.0)
|   |   |   purpose = radio/tv: bad (8.0/2.0)
|   |   |   purpose = domestic appliance: good (0.0)
|   |   |   purpose = repairs: good (2.0)
|   |   |   purpose = education: good (0.0)
|   |   |   purpose = vacation: good (0.0)
|   |   |   purpose = retraining: good (0.0)
|   |   |   purpose = business: good (11.0/3.0)
|   |   |   purpose = other: good (1.0)
|   |   savings_status = 500<=X<1000: good (11.0/3.0)
|   |   savings_status = >=1000: good (13.0/3.0)
|   |   savings_status = no known savings: good (41.0/5.0)
|   credit_amount > 9857: bad (20.0/3.0)
checking_status = >=200: good (63.0/14.0)
checking_status = no checking: good (394.0/46.0)

Number of Leaves  :   50
Size of the tree :    66

=== Summary ===

Correctly Classified Instances         725               72.5    %
Incorrectly Classified Instances       275               27.5    %
Kappa statistic                          0.2905
Mean absolute error                      0.344
Root mean squared error                  0.4372
Relative absolute error                 81.874  %
Root relative squared error             95.3993 %
Total Number of Instances             1000

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.861     0.593      0.772     0.861     0.814    good
  0.407     0.139      0.557     0.407     0.47     bad

=== Confusion Matrix ===

   a   b   <-- classified as
 603  97 |   a = good
 178 122 |   b = bad
```

Dal punto di vista della correttezza del modello il risultato è buono (errore di classificazione del 27.5 %, contro il 29.3 % del Decision Tree e il 27.0 % del De-

cision Forest di PolyAnalyst), ma la struttura dell'albero è talmente complessa, a causa dell'elevato numero di attributi presenti nel dataset, da risultare illeggibile sia nella forma grafica che nella forma testuale. E la leggibilità non migliora cambiando il valore dei parametri reducedErrorPruning e binarySplits.

Per aumentare la leggibilità è necessario diminuire il numero delle foglie, per cui bisogna aumentare il valore di minNumObj; la combinazione di parametri utilizzata è quindi: minNumObj = 45; reducedErrorPruning = False; binarySplits = False. Il risultato è riportato in fig. 4.5 e nelle successive righe.

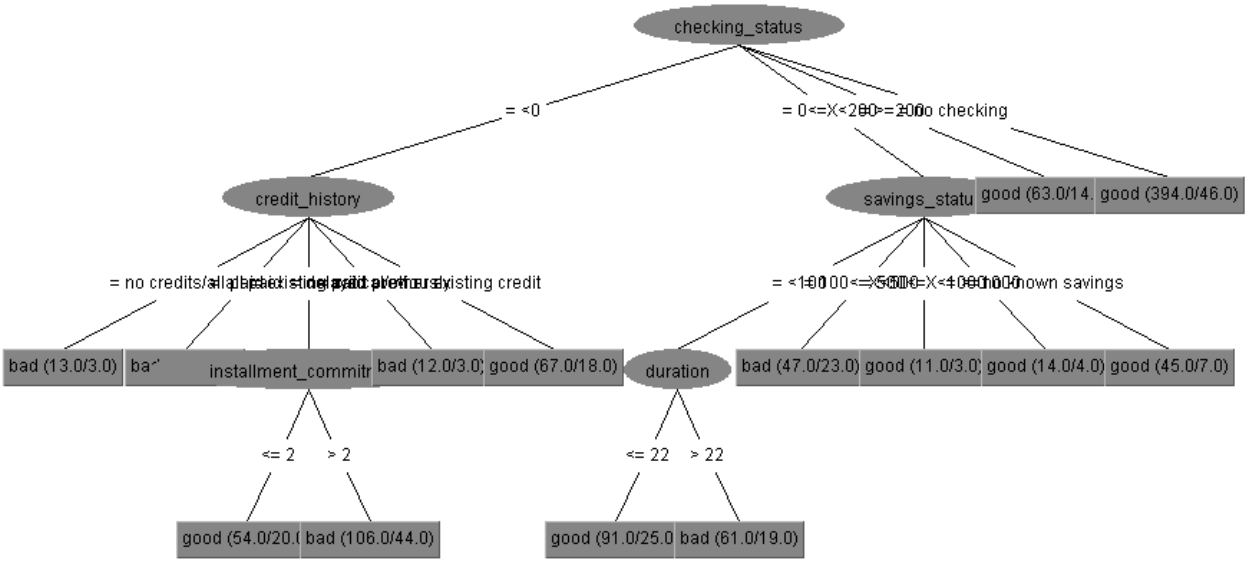


Figura 4.5 Albero le cui foglie possiedono almeno 45 elementi

```
=== Run information ===

Scheme:       weka.classifiers.j48.J48 -C 0.25 -M 45
Relation:     german_credit
Instances:    1000
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===
```

```
J48 pruned tree
------------------

checking_status = <0
|   credit_history = no credits/all paid: bad (13.0/3.0)
|   credit_history = all paid: bad (22.0/6.0)
|   credit_history = existing paid
|   |   installment_commitment <= 2: good (54.0/20.0)
|   |   installment_commitment > 2: bad (106.0/44.0)
|   credit_history = delayed previously: bad (12.0/3.0)
|   credit_history = critical/other existing credit: good (67.0/18.0)
checking_status = 0<=X<200
|   savings_status = <100
|   |   duration <= 22: good (91.0/25.0)
|   |   duration > 22: bad (61.0/19.0)
|   savings_status = 100<=X<500: bad (47.0/23.0)
|   savings_status = 500<=X<1000: good (11.0/3.0)
|   savings_status = >=1000: good (14.0/4.0)
|   savings_status = no known savings: good (45.0/7.0)
checking_status = >=200: good (63.0/14.0)
checking_status = no checking: good (394.0/46.0)

Number of Leaves  :   14
Size of the tree :    19

=== Summary ===

Correctly Classified Instances         722               72.2    %
Incorrectly Classified Instances       278               27.8    %
Kappa statistic                          0.2669
Mean absolute error                      0.3558
Root mean squared error                  0.4304
Relative absolute error                 84.6929 %
Root relative squared error             93.9191 %
Total Number of Instances             1000

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.874     0.633      0.763     0.874     0.815    good
  0.367     0.126      0.556     0.367     0.442    bad

=== Confusion Matrix ===

   a   b   <-- classified as
 612  88 |   a = good
 190 110 |   b = bad
```

Adesso la struttura dell'albero nella forma testuale è molto più leggibile, ma nella forma grafica risulta ancora troppo complessa. L'errore di classificazione si è mantenuto tollerabile.

Per aumentare ulteriormente la leggibilità senza peggiorare troppo la correttezza del modello si può operare sui parametri reducedErrorPruning e binarySplits: ponendoli entrambi pari a True si ottiene il risultato riportato in fig. 4.6 e nelle successive righe.

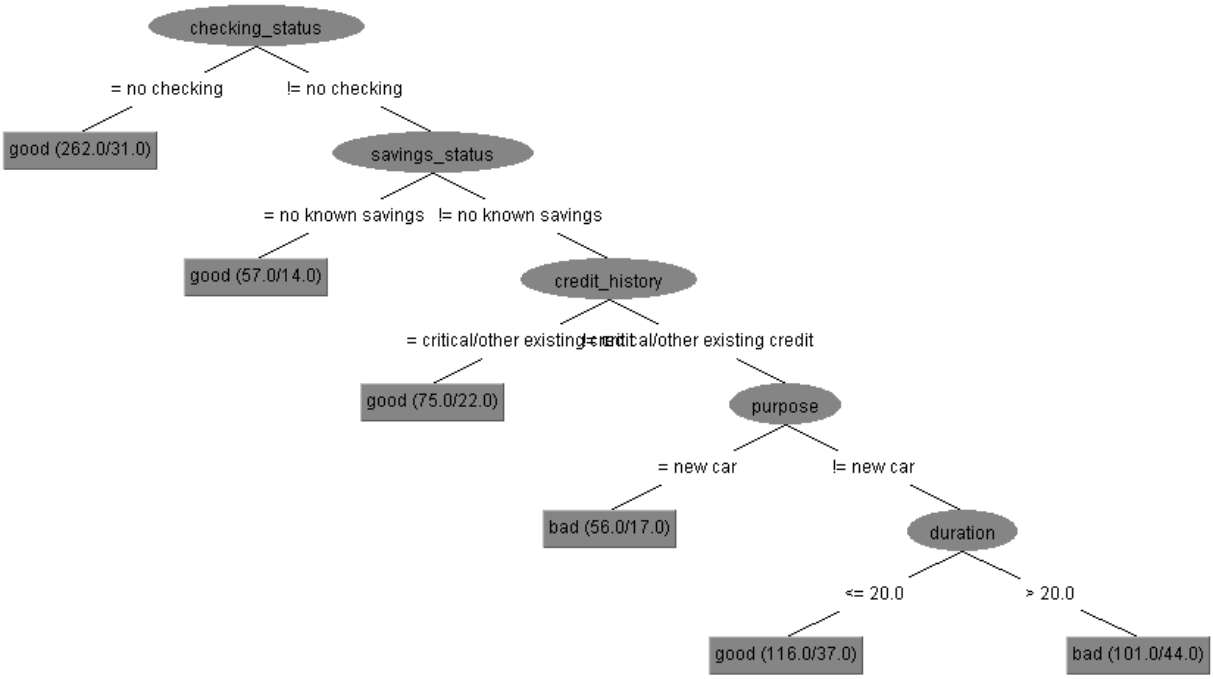


Figura 4.6 Albero binario potato con algoritmo non standard

```
=== Run information ===

Scheme:       weka.classifiers.j48.J48 -R -N 3 -B -M 45
Relation:     german_credit
Instances:    1000
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

checking_status = no checking: good (262.0/31.0)
checking_status != no checking
|   savings_status = no known savings: good (57.0/14.0)
|   savings_status != no known savings
|   |   credit_history = critical/other existing credit: good (75.0/22.0)
|   |   credit_history != critical/other existing credit
|   |   |   purpose = new car: bad (56.0/17.0)
|   |   |   purpose != new car
|   |   |   |   duration <= 20.0: good (116.0/37.0)
|   |   |   |   duration > 20.0: bad (101.0/44.0)

Number of Leaves  :   6
Size of the tree :    11

=== Summary ===

Correctly Classified Instances         702               70.2    %
Incorrectly Classified Instances       298               29.8    %
Kappa statistic                          0.1317
Mean absolute error                      0.3852
Root mean squared error                  0.4428
Relative absolute error                 91.685  %
Root relative squared error             96.6343 %

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.924     0.817      0.725     0.924      0.813    good
  0.183     0.076      0.509     0.183      0.27     bad
```

```
=== Confusion Matrix ===

   a   b   <-- classified as
 647  53 |   a = good
 245  55 |   b = bad
```

Appare evidente che adesso la struttura dell'albero è ben leggibile sia nella forma testuale che nella forma grafica, ma è sorto un altro problema: anche se l'errore di classificazione non è cresciuto molto (29.8 %), è cresciuto enormemente il numero di clienti cattivi classificati come buoni (da 190 a 245), e questo è indesiderabile.

Il miglior compromesso possibile tra leggibilità e correttezza della classificazione (soprattutto per quanto riguarda il numero di clienti cattivi classificati come buoni) si ottiene utilizzando la seguente combinazione dei parametri: MinNumObj = 45; reducedErrorPruning = False; binarySplits = True. Il risultato è riportato in fig. 4.7 e nelle successive righe.

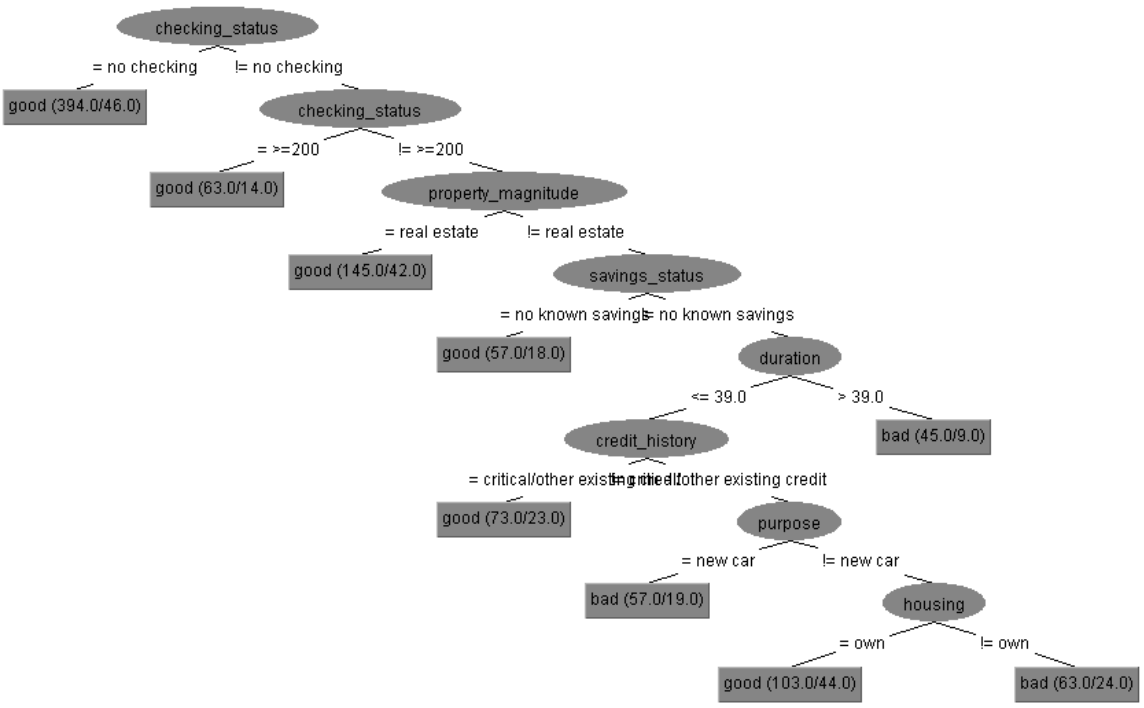


Figura 4.7 Albero binario costruito con l'algoritmo J48

```
=== Run information ===

Scheme:       weka.classifiers.j48.J48 -C 0.25 -B -M 45
Relation:     german_credit
Instances:    1000
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

checking_status = no checking: good (394.0/46.0)
checking_status != no checking
|   checking_status = >=200: good (63.0/14.0)
|   checking_status != >=200
|   |   property_magnitude = real estate: good (145.0/42.0)
|   |   property_magnitude != real estate
|   |   |   savings_status = no known savings: good (57.0/18.0)
|   |   |   savings_status != no known savings
|   |   |   |   duration <= 39.0
|   |   |   |   |   credit_history = critical/other existing credit: good (73.0/23.0)
|   |   |   |   |   credit_history != critical/other existing credit
|   |   |   |   |   |   purpose = new car: bad (57.0/19.0)
|   |   |   |   |   |   purpose != new car
|   |   |   |   |   |   |   housing = own: good (103.0/44.0)
|   |   |   |   |   |   |   housing != own: bad (63.0/24.0)
|   |   |   |   duration > 39.0: bad (45.0/9.0)

Number of Leaves  :   9
Size of the tree :    17

=== Summary ===

Correctly Classified Instances         694               69.4    %
Incorrectly Classified Instances       306               30.6    %
Kappa statistic                          0.2031
Mean absolute error                      0.3619
Root mean squared error                  0.4361
Relative absolute error                 86.1392 %
Root relative squared error             95.1615 %
Total Number of Instances             1000

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.846     0.66       0.749     0.846     0.795    good
  0.34      0.154      0.486     0.34      0.4      bad

=== Confusion Matrix ===

   a   b   <-- classified as
 592 108 |   a = good
 198 102 |   b = bad
```

4.2.2) Data mining in campo medico

- Dataset utilizzato: breast-cancer
- Provenienza: UCI Repository [24]
- Autori: Matjaz Zwitter & Milan Soklic (physicians)
  Institute of Oncology
  University Medical Center
  Ljubljana, Yugoslavia
  Date: 11 July 1988
- Descrizione dei dati: i dati rappresentano diversi attributi di pazienti donne operate per cancro al seno. La classe (attributo target) distingue i casi in cui si è avuta o non si è avuta ricomparsa del cancro.
- Algoritmi utilizzati: K*, C4.5 decision tree (J48).
- Risultati e commenti:

I risultati dell'analisi dei dati effettuata con l'algoritmo J48 sono riportati in fig. 4.8 e nelle successive righe.

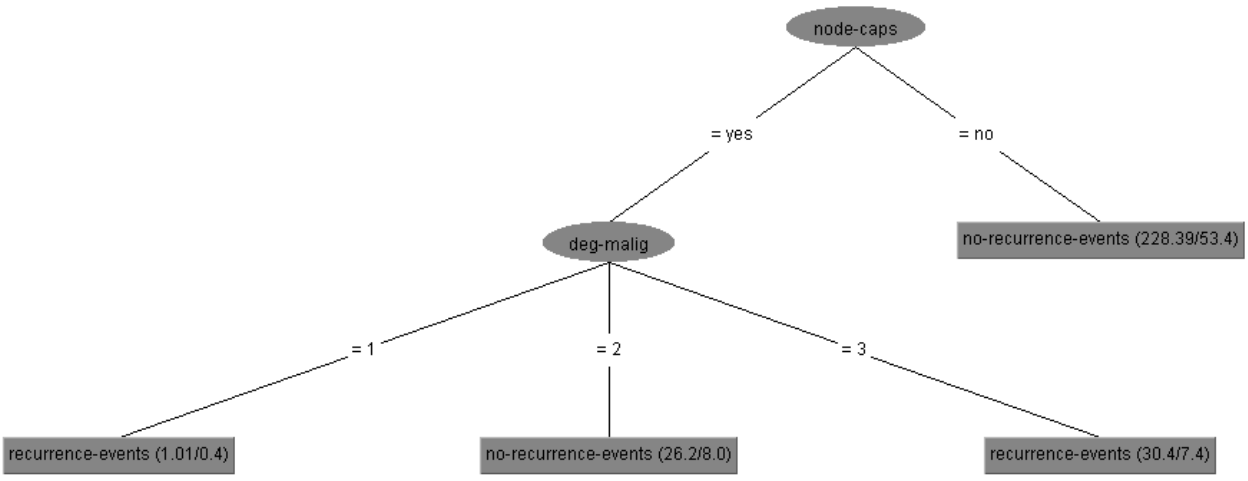


Figura 4.8 L'albero costruito con l'algoritmo J48

```
=== Run information ===

Scheme:       weka.classifiers.j48.J48 -C 0.25 -M 2
Relation:     breast-cancer
Instances:    286
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

node-caps = yes
|   deg-malig = 1: recurrence-events (1.01/0.4)
|   deg-malig = 2: no-recurrence-events (26.2/8.0)
|   deg-malig = 3: recurrence-events (30.4/7.4)
node-caps = no: no-recurrence-events (228.39/53.4)

Number of Leaves  :   4
Size of the tree :    6

=== Summary ===

Correctly Classified Instances         215               75.1748 %
Incorrectly Classified Instances        71               24.8252 %
Kappa statistic                          0.2632
Mean absolute error                      0.3702
Root mean squared error                  0.4354
Relative absolute error                 88.5088 %
Root relative squared error             95.2723 %

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.965     0.753      0.752     0.965     0.845    no-recurrence-events
  0.247     0.035      0.75      0.247     0.372    recurrence-events

=== Confusion Matrix ===

   a   b   <-- classified as
 194   7 |   a = no-recurrence-events
  64  21 |   b = recurrence-events
```

I risultati dell'analisi dei dati effettuata con l'algoritmo K* sono riportati nelle seguenti righe:

```
=== Run information ===

Scheme:       weka.classifiers.kstar.KStar -B 20 -M a
Relation:     breast-cancer
Instances:    286
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

KStar Beta Verion (0.1b).
Copyright (c) 1995-97 by Len Trigg (trigg@cs.waikato.ac.nz).
Java port to Weka by Abdelaziz Mahoui (am14@cs.waikato.ac.nz).

KStar options : -B 20 -M a
```

```
=== Summary ===

Correctly Classified Instances         213               74.4755 %
Incorrectly Classified Instances        73               25.5245 %
Kappa statistic                          0.3119
Mean absolute error                      0.3263
Root mean squared error                  0.4414
Relative absolute error                 78.0128 %
Root relative squared error             96.5768 %
Total Number of Instances              286

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.9       0.624      0.774     0.9        0.832    no-recurrence-events
  0.376     0.1        0.615     0.376      0.467    recurrence-events

=== Confusion Matrix ===

   a   b   <-- classified as
 181  20 |   a = no-recurrence-events
  53  32 |   b = recurrence-events
```

L'applicazione di questi due algoritmi evidenzia come sia molto importante, per una efficace analisi, specificare lo scopo per cui viene effettuato il data mining.

Se infatti lo scopo è costruire un modello da applicare a nuovi pazienti, in modo da monitorare con più attenzione le persone risultanti a rischio, allora l'algoritmo da preferire è il K*, perché, pur presentando un tasso di errore quasi uguale a quello del J48, ha rispetto a quello un minor numero di casi classificati come *no-recurrence-events* che in realtà sono *recurrence-events* (53 contro 64), e questo è un fattore di cruciale importanza.

Se invece lo scopo è trarre informazioni chiare e comprensibili dai dati a disposizione, in modo da indirizzare la ricerca medica, allora l'algoritmo da preferire è il J48, perché il suo schema ad albero evidenzia con chiarezza quelle che potrebbero essere considerate le cause principali di un ritorno del male.

4.2.3) Data mining in campo tecnico

- Dataset utilizzato: cpu-performance
- Provenienza: UCI Repository [24]
- Autori: Phillip Ein-Dor and Jacob Feldmesser
  Faculty of Management; Tel Aviv University; Ramat-Aviv;
  Tel Aviv, 69978; Israel
  Date: October, 1987
- Descrizione dei dati: i dati rappresentano diversi attributi di cpu costruiti da differenti aziende. La classe è di tipo numerico ed è un indice della performance di ciascuna cpu.
- Algoritmi utilizzati: Neural Network.
- Risultati e commenti:

I risultati dell'analisi dei dati sono riportati in fig. 4.9 e nelle successive righe (per semplicità si è omesso di riportare la composizione dei nodi della rete neurale).

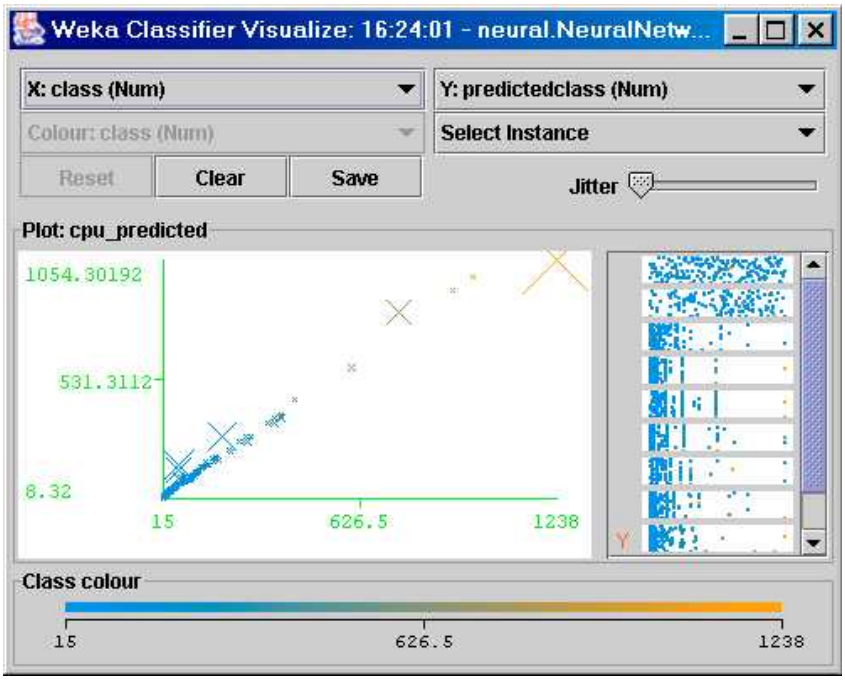


Figura 4.9 Visualizzazione dei risultati dell'algoritmo Neural Network

```
=== Run information ===

Scheme:       weka.classifiers.neural.NeuralNetwork -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20
-H a
Relation:     cpu
Instances:    209
Test mode:    10-fold cross-validation

=== Summary ===

Correlation coefficient                  0.9936
Mean absolute error                      7.1281
Root mean squared error                 17.9644
Relative absolute error                  8.1573 %
Root relative squared error             11.636  %
```

In questo caso non occorre un modello leggibile, ma un modello il più possibile esatto, cioè che possa effettuare previsioni accurate, per cui si è fatto ricorso all'algoritmo di rete neurale. La precisione ottenuta è elevata (nella fig. 4.9, infatti, quasi tutti i valori si trovano sulla retta che funge da bisettrice degli assi), anche se il risultato ottenuto con PolyAnalyst è di utilizzo più immediato, grazie alla formula riassuntiva del modello. Inoltre la presenza della formula in PolyAnalyst può aiutare a comprendere il peso dei diversi parametri nel modello costruito, mentre invece il modello costruito con WEKA ha un funzionamento a scatola chiusa.

4.3) Considerazioni finali

Come si è visto, tranne che nel caso del J48 e in pochi altri, WEKA, al contrario di PolyAnalyst, dispone di pochi strumenti di visualizzazione che possano consentire una immediata e facile lettura dei risultati ottenuti.

D'altra parte l'output testuale di WEKA è senza dubbio più completo rispetto a quello di PolyAnalyst, e un suo attento esame consente spesso di ottenere maggiori informazioni, e più precise, di quelle ottenibili con l'output grafico.

Per quanto riguarda la classificazione, WEKA è senza dubbio superiore a PolyA-

nalyst, grazie all'elevato numero di algoritmi disponibili e alla possibilità di intervenire su ciascuno di essi con diversi parametri. Ma per quanto riguarda tutti gli altri modelli di data mining, WEKA risulta molto carente. E non bisogna dimenticare che il punto di forza di PolyAnalyst risiede principalmente nella scoperta di regole e associazioni (soprattutto nell'algoritmo proprietario *Find Laws*).

Un punto debole di PolyAnalyst rispetto a WEKA risiede nella limitata possibilità di intervenire sul lavoro degli algoritmi mediante opportuni parametri, e ciò gli conferisce una certa rigidità di utilizzo. E inoltre si hanno meno informazioni su cosa sta facendo un algoritmo, e come lo sta facendo.

Nella seguente tabella si riassumono le principali caratteristiche e differenze dei software esaminati.

	WEKA 3.2.3	**POLYANALYST 4.5**
Prezzo	Gratis	A partire da 4000 $ (Sconti per usi educativi)
Ambienti supportati	Tutti quelli su cui è disponibili Java (Linux, Windows, Macintosh, ecc.)	Windows 9x/2000/NT/XP
Documentazione d'uso	Sufficiente per la versione a linea di comando, scarsa per la versione GUI	Buona (help on line, tutorial)
Codice sorgente	Disponibile e modificabile (Open Source)	Non disponibile
Modelli e algoritmi	Enfasi sulla classificazione e sui filtri. Scarsamente presenti tutti gli altri.	Buona varietà di algoritmi, con enfasi sulla scoperta di regole e associazioni.
Pre-elaborazione	Molteplici possibilità grazie ai numerosi filtri, facili da comprendere e da usare.	Potente grazie all'uso di formule, ma piuttosto difficile da apprendere.
Sorgenti dati	Preferito il formato testo in formato .ARFF. Possibile (ma macchinoso) collegamento a un database mediante driver JDBC-ODBC	Numerose possibilità offerte, tra cui file di testo in vari formati, foglio Excel, database e datawarehouse.
Interazione con l'elaborazione	Elevata grazie ai numerosi parametri di controllo degli algoritmi	Limitata
Usabilità	Talvolta macchinosa	Buona e intuitiva
Integrazione nei sistemi pre-esistenti	Possibilità di realizzarla mediante programmazione di apposite classi Java di interfaccia	Consentita tramite tecnologia Active X (è necessario acquistare la versione COM anziché quella Professional)

CAPITOLO 5

UN PSE VISUALE PER DATA MINING

Appare evidente, anche alla luce dell'esame dei software effettuato nei precedenti capitoli, che esiste ancora ampia possibilità di sviluppo ed evoluzione per gli applicativi per data mining. I forti e rapidi progressi che interessano questa disciplina, e nel contempo l'elevato numero di possibili applicazioni, fanno nascere l'esigenza di strumenti più potenti e più flessibili.

Con il presente lavoro si è progettato, quindi, di applicare la tecnologia Java Beans al fine di realizzare un Problem Solving Environment (PSE) visuale per data mining. Gli scopi che si intendono conseguire sono principalmente:

- avere uno strumento potente e flessibile, ma nel contempo facile da usare e da adattare alle proprie esigenze;
- avere uno strumento in grado di evolversi ed espandersi con facilità, seguendo le esigenze della ricerca e del mercato, riutilizzando però il più possibile il lavoro già svolto, in modo da accelerare la realizzazione di nuovi tool ed evitare lo spreco di tempo e risorse.

Nel presente capitolo, dopo una breve introduzione al linguaggio Java e alla tecnologia Java Beans, si illustrano i componenti del PSE progettati e realizzati, e il loro utilizzo in diversi ambienti di sviluppo.

5.1) Il linguaggio Java

Il linguaggio Java è stato presentato ufficialmente dalla Sun nel 1995.

E' un linguaggio ad oggetti con una spiccata capacità nella gestione della multimedialità e delle comunicazioni. Viene attualmente utilizzato per creare pagine Web dai contenuti dinamici e interattivi, per sviluppare applicazioni aziendali su larga scala, per migliorare le funzionalità dei server World Wide Web, per realizzare applicazioni destinate ai dispositivi di largo consumo (come telefoni cellulari e Personal Digital Assistent), e in tanti altri campi di applicazione.

Java è un linguaggio ad oggetti puro, dato che non è possibile programmare in modo non *object oriented* (in parte o del tutto), come ad esempio accade nel linguaggio C++.

Uno dei principali motivi del successo di Java è da individuarsi nella sua elevata *portabilità*: il codice compilato (*bytecode*) è eseguibile, senza necessità di ricompilazioni, su ogni tipo di piattaforma hardware o software che metta a disposizione una macchina virtuale Java (*Java Virtual Machine*, JVM). Dato che attualmente le JVM sono disponibili per ogni tipo di sistema operativo e per un numero molto elevato di hardware differenti, il bytecode è di fatto un esempio di portabilità reale, non solo teorica. Ed inoltre, il livello di astrazione dato dall'introduzione di uno strato di software aggiuntivo semplifica la programmazione, in quanto svincola il programmatore dal doversi preoccupare delle problematiche connesse alla piattaforma sottostante. Però, anche se con Java è più facile scrivere programmi portabili, il solo fatto di utilizzare Java non sempre garantisce la portabilità di un programma, e talvolta il programmatore deve gestire direttamente il problema delle differenze tra sistemi; è bene quindi testare i programmi Java su tutti i sistemi su cui si intende eseguirli.

Ha contribuito al successo di Java anche il fatto che esso sia stato progettato con il preciso obiettivo di offrire un elevato livello di sicurezza, mantenendo nel contempo una elevata semplicità di implementazione. Ad esempio, la memoria viene gestita direttamente dalla JVM mediante il *Garbage Collector*: quando un oggetto non è più utilizzato all'interno di un programma (ossia quando non ci sono più riferimenti a questo oggetto), l'oggetto viene contrassegnato per la *garbage collection*, e questo significa che la JVM esegue un task detto *Garbage Collector* che automaticamente rilascia la memoria che era utilizzata dall'oggetto. Ciò semplifica la programmazione, in quanto il programmatore non deve occuparsi esplicitamente del rilascio della memoria occupata (come invece avviene nel linguaggio C++), e rende i programmi più efficienti, in quanto si hanno minori possibilità che si verifichino gli sprechi di memoria comuni in altri linguaggi.

Un'altra delle caratteristiche fondamentali di Java è che esso mette a disposizione un meccanismo di *multithreading*, col quale è possibile realizzare programmi con attività parallele, ossia è possibile, all'interno della stessa applicazione, eseguire contemporaneamente più *task*: il *Garbage Collector* ne è un esempio.

I programmi Java si compongono di classi, che a loro volta si compongono di metodi, che eseguono le varie attività. Le classi poi sono raggruppate in *package*, ossia in una struttura di directory utilizzata per organizzare le varie classi in una sorta di libreria. L'istanza di una classe è un *oggetto*.

Con Java è semplice far uso del principio del *riutilizzo del software*, che è alla base della programmazione orientata agli oggetti: uno degli obiettivi dei programmatori è quello di creare componenti software riutilizzabili, in modo che non sia necessario ridefinire continuamente tutto il codice all'interno di ogni nuovo programma. Il software viene quindi costruito partendo da componenti già esistenti

ben definiti, attentamente testati, ben documentati e portabili: esistono numerose librerie di classi Java, alcune a pagamento altre freeware. Il riutilizzo del software rende più semplice e più veloce lo sviluppo di software potente e di alta qualità.

5.2) La tecnologia Java Beans

Uno degli obiettivi più ambiziosi dell'ingegneria del software è organizzare lo sviluppo di sistemi in maniera simile a quanto è stato fatto in altre branche dell'ingegneria, dove la presenza di un mercato di parti standard altamente riutilizzabili permette di aumentare la produttività riducendo nel contempo i costi. Nella meccanica, ad esempio, esiste da tempo un importante mercato di componenti riutilizzabili, come viti, dadi, bulloni e ruote dentate; ciascuno di questi componenti trova facilmente posto in centinaia di prodotti diversi.

L'industria del software, sempre più orientata alla filosofia dei componenti, sta dando vita a due nuove figure di programmatore: il progettista di componenti e l'assemblatore di applicazioni. Il primo ha il compito di scoprire e progettare oggetti software di uso comune, che possano essere riutilizzati con successo in contesti differenti. Il secondo è un professionista specializzato in un particolare dominio applicativo, capace di creare programmi complessi utilizzando componenti standard e componendoli con strumenti grafici o linguaggi di scripting.

Java Beans è una specifica, ossia un insieme di regole seguendo le quali è possibile realizzare in Java componenti software riutilizzabili, che abbiano la capacità di interagire con altri componenti attraverso un protocollo di comunicazione comune. Tali componenti vengono detti *JavaBean* o più semplicemente *bean*. Un bean rappresenta un mattone di un programma: ogni componente è una unità di utilizzo abbastanza grossa da incorporare una funzionalità evoluta, ma piccola ri-

spetto ad un programma completo.

Le specifiche della Sun danno la seguente definizione per un bean:

> *Un JavaBean è un componente software riutilizzabile che può essere manipolato visualmente in un Builder Tool.*

Un bean può essere visuale (*visible*) o non visuale (*invisible*): nel primo caso, il bean possiede un aspetto grafico (*GUI representation*) deciso dal programmatore e visualizzato in tutti i contesti grafici in cui si fa uso del bean. Nel secondo caso, invece, il bean non possiede un aspetto grafico deciso a priori dal programmatore, per cui in taluni ambienti di sviluppo non verrà visualizzato (sarà trasparente), mentre in altri gli verrà associato un aspetto di default. Un *invisible bean* è un bean a tutti gli effetti, e può essere utilizzato in tutti i Builder Tool, anche in quelli grafici; l'unica sua differenza è che non possiede un aspetto grafico suo proprio. E' da sottolineare che, mentre in generale non è necessario che un bean erediti da una qualche specifica classe o interfaccia, per essere visuale un bean deve necessariamente ereditare dalla classe `java.awt.Component`.

Ciascun bean è caratterizzato dai servizi che è in grado di offrire e può essere utilizzato in un ambiente di sviluppo differente rispetto a quello in cui è stato realizzato. Quest'ultimo punto è cruciale nella filosofia dei componenti: sebbene i bean siano a tutti gli effetti classi Java, e possano essere manipolati completamente per via programmatica, essi vengono solitamente utilizzati in ambienti di sviluppo diversi, come tool grafici o linguaggi di scripting.

I tool grafici, come ad esempio NetBeans [21], permettono di manipolare i componenti in maniera visuale: un assemblatore di componenti può selezionare i bean da una *palette*, inserirli in un apposito contenitore software, impostarne le pro-

prietà e collegare gli eventi di un bean ai metodi di un altro, generando in tal modo applicazioni, applet e persino nuovi componenti, il tutto senza scrivere una sola riga di codice.

I linguaggi di scripting, di contro, offrono una maggiore flessibilità rispetto ai tool grafici, senza presentare le complicazioni di un linguaggio di programmazione generico. Ad esempio, la programmazione di pagine web dinamiche, uno dei domini applicativi di maggior attualità, deve il suo rapido sviluppo a un'intelligente politica di stratificazione, che vede le funzionalità di più basso livello (come la gestione dei database o l'interfacciamento con le risorse di sistema) incapsulate all'interno di bean, mentre tutto l'aspetto della presentazione viene sviluppata con un semplice linguaggio di scripting, tipo Java Server Pages o PHP.

Ciascun bean, quindi, può essere eseguito in un ampio numero di differenti ambienti. Ma è necessario sottolineare che può farlo in due modalità differenti. La prima, detta *design time*, rappresenta la capacità del bean di funzionare in un ambiente di sviluppo (al quale ci si riferisce, in questo caso, come *design environment*); in tale contesto è fondamentale che il bean fornisca correttamente al Buider Tool le informazioni che lo riguardano, e consenta all'utilizzatore di manipolarne proprietà e comportamento. La seconda modalità, detta *run time*, rappresenta la capacità del bean di funzionare in un applicativo finito; in tale contesto non è fondamentale che il bean consenta all'utente di modificarne il comportamento.

Le specifiche *Java Beans* sono state progettate, tra l'altro, con lo scopo di consentire un agevole utilizzo dei bean negli ambienti distribuiti, in particolar modo nel World Wide Web. Esse, infatti, presuppongono che i bean vengano eseguiti in un ambiente *multi-threaded*, e che quindi diversi *thread* possano simultaneamente

lanciare eventi, chiamare metodi, impostare proprietà. Inoltre i bean sono soggetti al modello standard di sicurezza di Java, senza alcun inasprimento né rilassamento.

Ci sono diversi fattori che caratterizzano un componente JavaBean: proprietà, metodi, eventi, introspezione, persistenza, deployment. Nel seguito si illustrano in dettaglio tali fattori.

❖ **Proprietà**

Ciascun bean può possedere delle proprietà. Esse sono attributi privati che descrivono l'aspetto e il comportamento del bean, e possono essere modificate durante tutto il ciclo di vita del componente, ma sono accessibili soltanto attraverso una coppia di appositi metodi pubblici *get* e *set*, che devono essere realizzati nella forma:

```
public  <PropertyType>  getPropertyName ()
public  void  setPropertyName (<PropertyType>)
```

Seguendo tale convenzione di *naming*, i precedenti metodi consentiranno agli ambienti di sviluppo di rilevare automaticamente le proprietà del bean, di dedurne il tipo e di determinarne le regole di accesso: Read Only in presenza del solo metodo *get*, Write Only in presenza del solo metodo s*et* o Read/Write in presenza di entrambi i metodi. Tale meccanismo prende il nome di *introspezione*.

I metodi *get* e *set* costituiscono quindi l'unica via di accesso pubblica alle proprietà: ciò permette al progettista di componenti di stabilire per ogni parametro precise regole di accesso.

Quelle finora descritte sono le proprietà semplici o *standard*, che seguono una convenzione radicata da tempo nella normale programmazione ad oggetti. Ma i bean possono implementare anche proprietà *bound*, che sono caratteristiche dell'universo dei componenti, dove si verifica molto spesso la necessità di collegare il valore delle proprietà di un componente a quelle di un altro, in modo che si mantengano aggiornati entrambi. I metodi *set* delle proprietà bound inviano una notifica a tutti gli ascoltatori registrati ogni qualvolta viene alterato il valore della proprietà. Tutto ciò avviene secondo un meccanismo codificato nelle specifiche *Java Beans*, che prevede il lancio di un evento `PropertyChange`, contenente il nome della proprietà, il vecchio valore ed il nuovo.

Se si utilizzano i bean all'interno di un programma di sviluppo visuale (Builder Tool grafico), le proprietà di un componente vengono visualizzate in un apposito pannello, detto *Property Sheet*, che mostra lo stato delle proprietà e permette di modificarne il valore con un opportuno strumento grafico, ad esempio con un campo di testo per valori `String` o una palette per proprietà `Color`: tali strumenti grafici vengono detti *editor di proprietà*. I tool grafici dispongono di editor di proprietà in grado di supportare i tipi Java più comuni, come i tipi numerici, le stringhe e i colori; nel caso si desideri rendere editabile una proprietà di un tipo diverso, è necessario realizzare una opportuna classe di supporto conforme all'interfaccia `PropertyEditor`.

❖ **Metodi**

Come detto, un bean è una normale classe Java che rispetta le specifiche imposte dalla tecnologia Java Beans. E i metodi di un bean sono metodi pubblici Java, con l'unica differenza che essi risultano accessibili anche attraverso lin-

guaggi di scripting e Builder Tool: i metodi sono la prima e più importante via di accesso ai servizi di un bean.

❖ **Eventi**

Nella programmazione ad oggetti tradizionale non esiste nessuna convenzione su come modellare lo scambio di messaggi tra oggetti: ogni programmatore adotta un proprio sistema, creando una fitta rete di dipendenze che rende molto difficile il riutilizzo di oggetti in contesti differenti da quello di partenza.

Invece le classi Java progettate secondo la specifica Java Beans adottano un meccanismo di comunicazione basato sugli eventi. L'esistenza di un unico protocollo di comunicazione standard garantisce l'intercomunicabilità tra componenti, indipendentemente da chi li abbia prodotti.

Per implementare un meccanismo di comunicazione basato su eventi occorre innanzitutto definire un'opportuna sottoclasse di `EventObject`, che racchiuda tutte le informazioni relative all'evento da propagare. Il numero e il tipo dei parametri della sottoclasse dipendono dall'evento da descrivere. L'unico parametro che è obbligatorio fornire è un riferimento all'oggetto che ha generato l'evento: tale riferimento, richiamabile con il metodo `getsource()` della classe `EventObject`, permetterà all'ascoltatore di interrogare la sorgente degli eventi qualora ce ne fosse bisogno.

In secondo luogo è necessario definire l'interfaccia di programmazione degli ascoltatori di eventi. Tale interfaccia deve essere definita come sottoclasse di `EventListener`, per essere riconoscibile come ascoltatore durante l'introspezione.

Se poi si desidera aggiungere a un bean la capacità di generare eventi, occorre

implementare la seguente coppia di metodi nel bean:

```
add<EventListenerType> (<EventListenerType>  ev)
remove<EventListenerType> (<EventListenerType>  ev)
```

Il primo metodo serve a registrare un ascoltatore di eventi presso il bean sorgente di eventi, il secondo serve a rimuovere un ascoltatore di eventi precedentemente registratosi.

Se si vuole che un evento generato da un bean scateni un'azione su un altro bean, è necessario creare una classe che realizzi un collegamento tra i due. Tale classe, detta *Adapter*, viene registrata come ascoltatore presso la sorgente dell'evento, e formula una chiamata al metodo di destinazione del secondo bean ogni volta che riceve una notifica del bean sorgente. Gli strumenti grafici (tipo BeanBox o NetBeans) generano questo tipo di classi in maniera automatica: tutto quello che l'utente deve fare è collegare, con pochi click di mouse, l'evento di un bean sorgente a un metodo di un bean target.

❖ **Introspezione**

I Builder Tool scoprono i servizi di un bean (proprietà, metodi ed eventi) attraverso un processo noto come introspezione, che consiste principalmente nell'interrogare il componente per conoscerne i metodi e dedurre da questi le caratteristiche del bean.

Il progettista di componenti può attivare l'introspezione in due maniere: seguendo precise convenzioni nella formulazione delle firme dei metodi, o creando una speciale classe *BeanInfo*, che fornisce esplicitamente un elenco dei servizi del bean cui si riferisce. Il primo metodo è senza dubbio più sempli-

ce: definire i metodi di accesso a un determinato servizio seguendo le regole di naming descritte dalla specifica Java Beans consente ai Builder Tool di individuare i servizi di un bean semplicemente osservandone l'interfaccia di programmazione. Il ricorso alla classe *BeanInfo*, d'altra parte, torna utile in tutti quei casi in cui è necessario mascherare alcuni metodi (in modo da esporre solamente un sottoinsieme dei servizi effettivi del bean, mascherando ad esempio alcuni dei metodi ereditati da una superclasse) o in cui si vuole dotare il bean di funzionalità avanzate, come ad esempio associargli delle icone che lo rappresentino nella palette dei componenti di un tool grafico.

Per creare una classe BeanInfo bisogna definire una classe il cui nome è quello del bean con l'aggiunta del suffisso BeanInfo. Tale classe deve implementare l'interfaccia `BeanInfo`.

Alcuni ambienti di sviluppo, come ad esempio NetBeans, sono in grado di costruire automaticamente la classe BeanInfo associata ad un certo bean.

La classe BeanInfo deve essere messa nello stesso package che contiene il bean. In assenza di una classe BeanInfo viene usata l'introspezione standard per determinare le caratteristiche esposte del bean.

❖ **Persistenza**

La persistenza permette ad un bean di salvare il proprio stato e di ripristinarlo in un secondo tempo. Le specifiche Java Beans supportano la persistenza grazie alla s*erializzazione* (*Object Serialization*), che permette di risolvere questo problema in modo molto rapido: per rendere serializzabile una classe bean è di norma sufficiente implementare l'interfaccia `Serializable`, sfuttando così l'*Object Serialization* di Java.

Solitamente in Java una interfaccia possiede dei metodi che devono necessariamente essere implementati da una classe che implementa l'interfaccia. L'interfaccia `Serializable` invece è priva di metodi: essa viene usata come "marcatore" per segnalare al compilatore quelle classi di cui si intende consentirne il salvataggio dello stato per un successivo ripristino. E' compito dell'ambiente di sviluppo garantire la persistenza delle classi che implementano l'interfaccia `Serializable`, cioè è l'ambiente di sviluppo che deve occuparsi del salvataggio e del ripristino delle classi serializzabili, e non il programmatore.

Esistono solo poche regole per implementare classi serializzabili: anzitutto è necessario dichiarare un costruttore privo di argomenti; in secondo luogo una classe serializzabile deve definire al suo interno solamente attributi serializzabili. Se si desidera fare in modo che un particolare attributo non venga salvato al momento della serializzazione, si può ricorrere al modificatore `transient`. La serializzazione standard, inoltre, non salva lo stato delle variabili `static`.

Per tutti i casi in cui la serializzazione standard non risultasse applicabile, occorre procedere all'implementazione dell'interfaccia `Externalizable`, fornendo, attraverso i metodi `readExternal (ObjectInput in)` e `writeExternal (ObjectOutput out)`, delle istruzioni esplicite su come salvare lo stato di un oggetto (istanza di un bean) su uno *stream* e su come ripristinarlo in un secondo tempo.

❖ **Deployment**

I bean possono essere consegnati, in gruppo o singolarmente, attraverso *file JAR*, che sono speciali archivi complessi in grado di trasportare tutto quello di cui un bean ha bisogno, come classi, immagini o altri file di supporto. Grazie ai

file **.JAR** è possibile consegnare i bean con una modalità del tipo "chiavi in mano": l'utente deve solo caricare il file JAR nel proprio ambiente di sviluppo, e i bean in esso contenuti verranno subito messi a sua disposizione.

Ad ogni archivio JAR è associato un file di testo `manifest.MF`, detto *file manifesto*, che si trova nella directory `META-INF` del file JAR ed è usato per descriverne i contenuti. Tutti gli ambienti di sviluppo che supportano i JavaBean sanno come cercare il file manifesto nel file JAR.

Quando un file JAR contenente uno o più bean viene caricato in un ambiente di sviluppo, quest'ultimo analizza il file manifesto per determinare quali delle classi del file JAR rappresentano dei bean. Inoltre l'interprete Java può eseguire direttamente un'applicazione da un file JAR se il file manifesto indica quale classe del file JAR contiene il metodo `main` da lanciare.

5.3) I componenti del PSE visuale per data mining

Per realizzare il PSE visuale per data mining si sono progettati e implementati cinque bean e un evento che potesse collegarli opportunamente. Per le funzionalità inerenti al data mining si è fatto ricorso alle classi messe a disposizione da WEKA 3.2.3, mettendo in pratica il principio del riutilizzo del software. Ciò non è stato semplice a causa della scarsa documentazione fornita con questo software, che per quanto riguarda la descrizione delle classi e dei corrispondenti metodi consiste unicamente nella documentazione creata con Javadoc (utility fornita da Sun Microsystems all'interno del *Java 2 Software Development Kit* per la realizzazione automatica della documentazione di un programma, in formato HTML, utilizzando parte dei commenti presenti nel codice sorgente del programma stesso). Più volte si è quindi dovuto studiare direttamente il codice sorgente delle

classi WEKA per comprenderne il funzionamento e utilizzarle correttamente.

I bean realizzati non hanno ereditato dalle classi WEKA, ma le contengono. Essi sono stati implementati come sottoclassi indirette di `java.awt.Component`.

Si è fatta questa scelta per i seguenti motivi:

- avere dei bean visuali (*visible bean*), ossia componenti con un aspetto grafico definito a priori, in modo da mantenere un aspetto uniforme in tutti gli ambienti in cui si andranno ad utilizzare;
- mantenere una elevata padronanza e flessibilità nella realizzazione dei componenti, in modo da poter includere o escludere con semplicità funzionalità di WEKA ed eventualmente di altre librerie di classi;
- porre in enfasi l'aspetto grafico e visuale dei bean, affinché siano utilizzabili nel modo più semplice possibile.

La differenza principale tra il PSE visuale progettato e WEKA non sta tanto in ciò che fa, ma in come lo fa: con il PSE visuale l'applicativo per data mining viene discretizzato in tanti minuscoli componenti, che sfruttano appieno tutte le potenzialità e la flessibilità conferita loro dalla tecnologia *Java Beans*. Ne consegue che i componenti possono essere utilizzati tutti insieme o solo in parte; possono essere connessi in serie o in parallelo, utilizzando componenti diversi o lo stesso componente con impostazioni differenti; possono funzionare su computer distinti in modalità client-server; possono essere agevolmente inseriti in un altro applicativo Java, o in una pagina Web sotto forma di applet. Il tutto con i vantaggi di avere un piccolo e compatto componente software indipendente e fortemente specializzato.

Nel seguito si illustrano nel dettaglio i componenti e l'evento realizzati. Il codice Java sviluppato è riportato in appendice A.

5.3.1) Il componente InputDataset

Lo scopo di questo componente è consentire il caricamento dei dati nel PSE.

Esso estende la classe `java.awt.Panel`, in modo da poter presentare dei pulsanti nella sua parte visibile. Tali pulsanti evidenziano all'utente le azioni che possono essere compiute dal bean e lo guidano nel suo funzionamento, variando dinamicamente in numero e funzionalità.

Questo bean incapsula ed espande le funzionalità della classe `weka.core.Instances`, che implementa una tabella bidimensionale per la rappresentazione interna a WEKA del dataset. Quindi WEKA, e di conseguenza il software che fa uso delle sue classi, è in grado di elaborare solamente dati organizzati nella forma di *matrice dei dati*. Il formato **.ARFF** corrisponde alla rappresentazione esterna della classe `Instances`.

Il bean consente di caricare un solo dataset (fig.5.2) o due dataset distinti (fig. 5.3), uno per il training e l'altro per il testing: una apposita finestra di dialogo chiede all'utente di effettuare la scelta (fig. 5.1).

Quando si carica un unico dataset, le scelte possibili sono 3: solo dataset di training, solo dataset di testing, dataset che verrà utilizzato in parte per il training ed in parte per il testing.

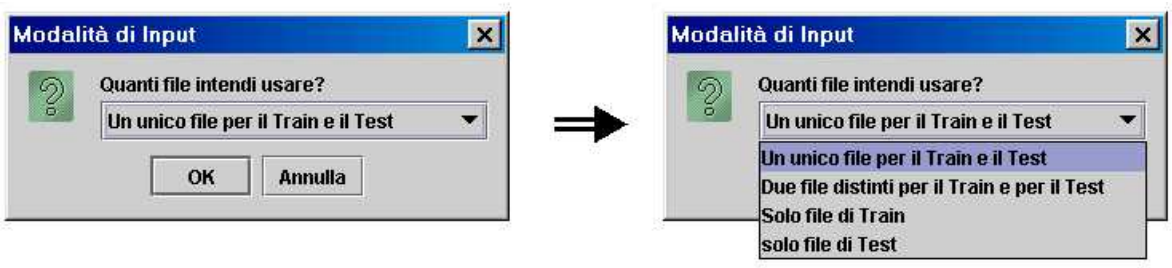


Figura 5.1 Finestra di dialogo per la scelta del numero di dataset da caricare

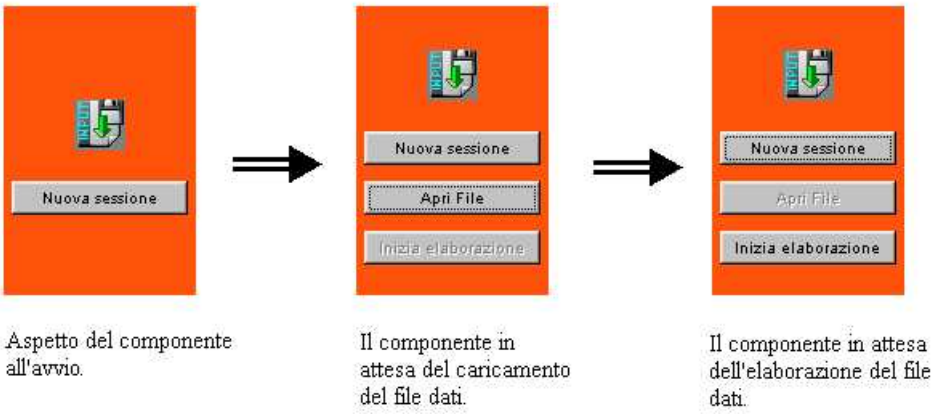


Figura 5.2 Fasi di caricamento di un unico dataset

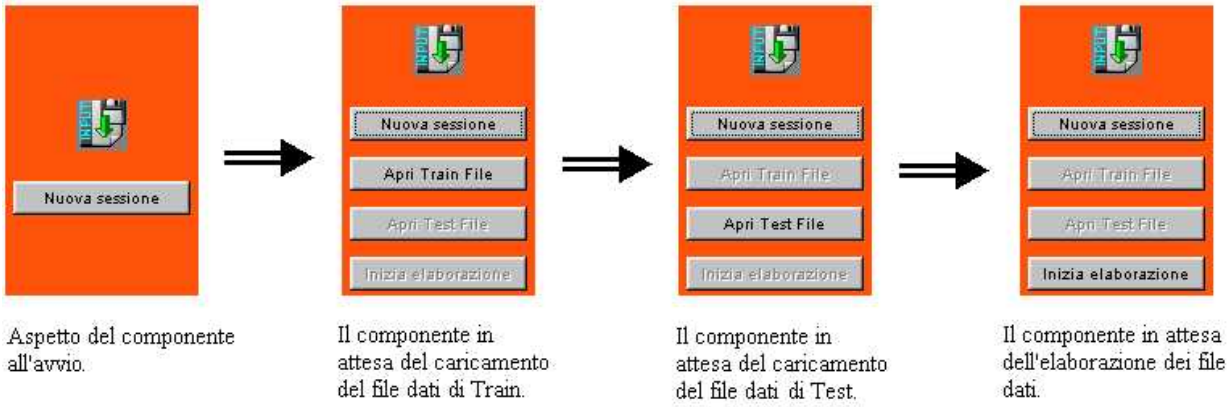


Figura 5.3 Fasi di caricamento di due distinti dataset

Come mostrato in fig. 5.4, il componente è in grado di caricare un dataset non solo a partire da un file **.ARFF**, ma anche da un file di testo in formato CSV (la prima riga deve contenere i nomi degli attributi, e i dati devono essere separati tramite virgole o caratteri TAB) o da un database registrato come origine dati **ODBC** (si fa ricorso al driver JDBC-ODBC); in quest'ultimo caso viene visualizzata una finestra di dialogo che consente all'utente di inserire l'indirizzo del database e la query SQL da sottoporre al database (fig. 5.5).

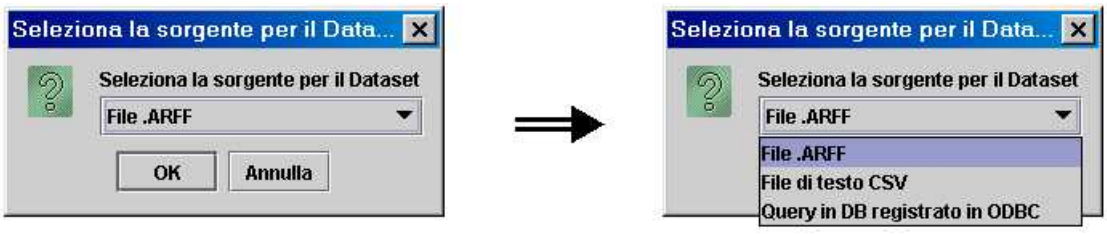


Figura 5.4 Finestra di dialogo per la selezione del formato della sorgente dati

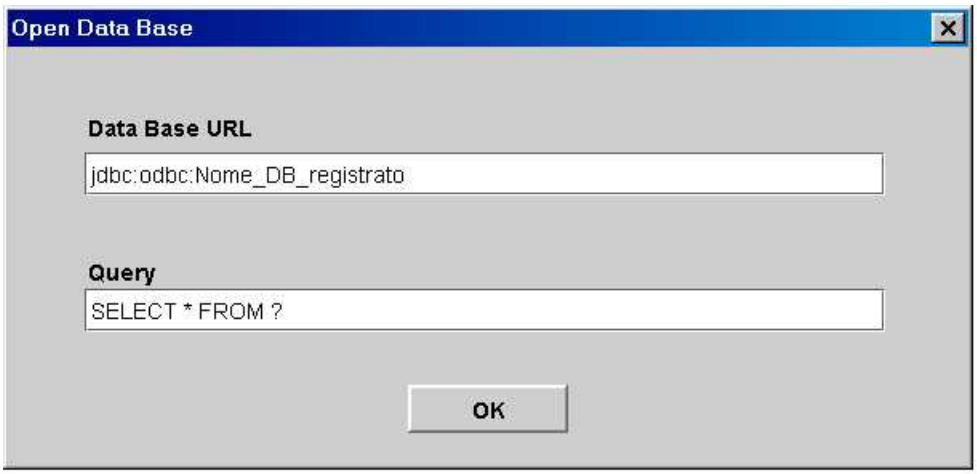


Figura 5.5 Finestra di dialogo per l'interazione con un database

I metodi esposti dal bean *InputDataset* sono:

- `startDatasetStream()`, che invia ai bean registrati presso di esso i dataset di Train e di Test utilizzando un evento `DatasetEventObject`;
- `obtainDatasetStream()`, che ritorna l'evento `DatasetEventObject` lanciato da questo bean; tale metodo è necessario per il funzionamento del bean in BeanBuilder, e in tale ambiente sostituisce il metodo `startDatasetStream()` (si veda il paragrafo 5.4.2).

Le proprietà esposte da questo bean, e illustrate in figura 5.6, sono:

- trainFileInUse: è una proprietà bound che ritorna il nome del file di training attualmente in uso, in modo che i componenti registrati possano essere notifi-

cati in caso di cambiamento; è read-only, cioè se ne può visualizzare il contenuto, ma non se ne può modificare direttamente il valore: per farlo occorre far ricorso alla procedura guidata dai pulsanti del bean;

- testFileInUse: è una proprietà bound che ritorna il nome del file di testing attualmente in uso, in modo che i componenti registrati possano essere notificati in caso di cambiamento; come la precedente è read-only;
- trainClassIndex: rappresenta l'indice dell'attributo del dataset di training che fungerà da classe durante la classificazione del dataset (la classe è l'attributo discriminante del dataset, ossia quello in base al quale avviene la classificazione delle tuple). La numerazione degli attributi parte da zero, e per default viene utilizzato l'ultimo attributo come classe;
- testClassIndex: rappresenta l'indice dell'attributo del dataset che fungerà da classe durante la classificazione del dataset di testing;
- trainNumClasses: proprietà read-only che rappresenta il numero di possibili "etichette" (ossia valori) per la classe del dataset di training;
- testNumClasses: proprietà read-only che rappresenta il numero di possibili "etichette" per la classe del dataset di testing;
- trainNumInstances: proprietà read-only che rappresenta il numero di tuple contenute nel dataset di training;
- testNumInstances: proprietà read-only che rappresenta il numero di tuple contenute nel dataset di testing;
- trainNumAttributes: proprietà read-only che rappresenta il numero di attributi del dataset di training;
- testNumAttributes: proprietà read-only che rappresenta il numero di attributi del dataset di testing.

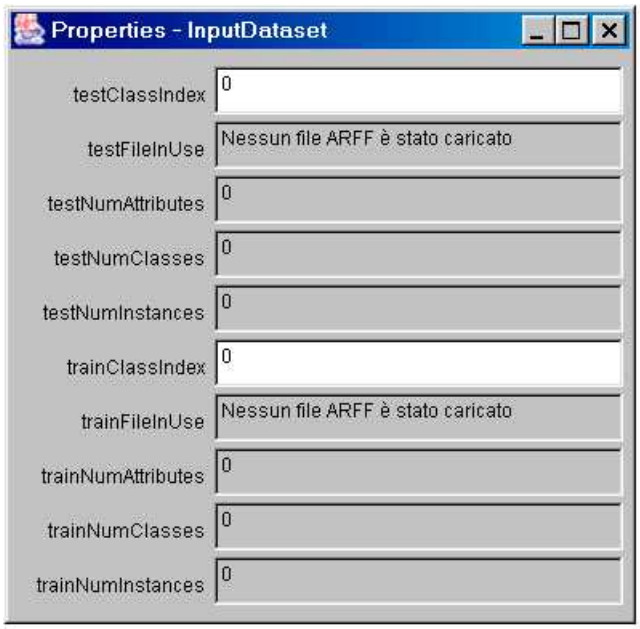


Figura 5.6 Le proprietà del componente *InputDataset*

5.3.2) Il componente J48Bean

Lo scopo di questo componente è quello di elaborare i dati facendo ricorso ad un algoritmo che produca un albero di decisione.

Esso estende la classe `java.awt.Canvas`, e incapsula ed espande le funzionalità della classe `weka.classifiers.j48.J48`, che implementa l'algoritmo di classificazione C4.5 e produce un albero di decisione, anche in formato visuale.

Questo componente può operare in quattro modalità differenti:

- può elaborare un solo dataset per effettuare unicamente il training del classificatore (solo dataset di Train);
- può elaborare un solo dataset per effettuare unicamente il testing di un classificatore precedentemente costruito (solo dataset di Test);
- può elaborare due dataset distinti (uno per il training e uno per il testing);

- può elaborare un unico dataset da utilizzare sia per il training che per il testing. E' in grado di operare in due modi differenti per ottenere i dataset di Train e Test a partire dall'unico dataset: mediante Cross Validation o mediante divisione percentuale del dataset originario (fig. 5.7).

Il componente *J48Bean* espone un solo metodo, `dataset_EventPerformed`, che ha come argomento un evento del tipo `DatasetEventObject` contenente i dataset da elaborare. Il metodo risponde all'evento dando inizio all'elaborazione dei dataset ricevuti.

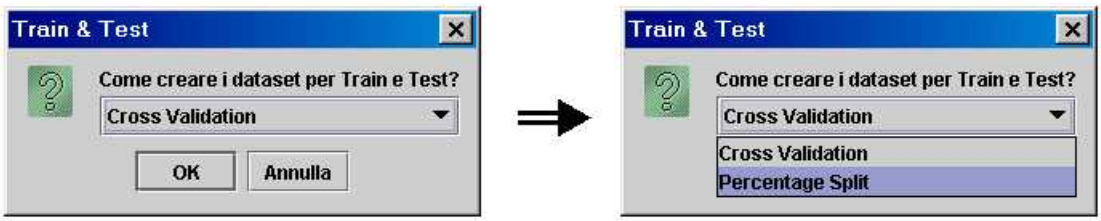


Figura 5.7 Finestra di dialogo per la selezione della modalità di creazione dei dataset di training e testing a partire da un unico dataset

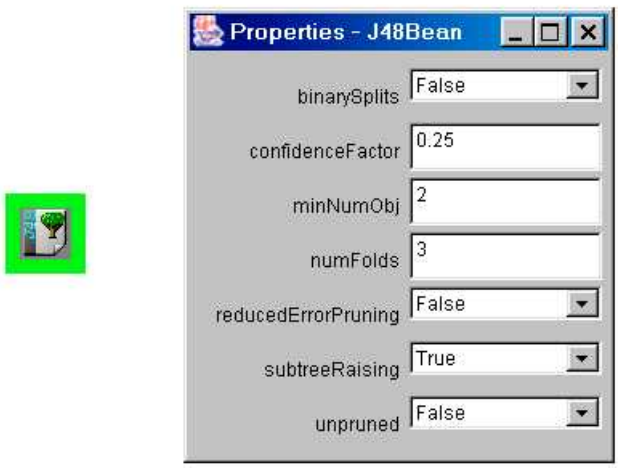


Figura 5.8 Il componente *J48Bean* e le sue proprietà

Le proprietà esposte da questo bean, e illustrate in figura 5.8, sono:

- binarySplits: proprietà booleana che stabilisce se l'albero costruito dal bean debba essere binario;
- confidenceFactor: rappresenta la soglia di confidenza per la potatura (*pruning*) dell'albero costruito dal componente;
- minNumObj: rappresenta il minimo numero di tuple che devono essere contenute in ciascuna foglia dell'albero;
- reducedErrorPruning: proprietà booleana che stabilisce se il bean debba usare l'algoritmo di potatura standard o l'algoritmo di *reduced error pruning*;
- numFolds: stabilisce il numero di *folds* nel caso venga utilizzato l'algoritmo di *reduced error pruning*;
- subtreeRaising: proprietà booleana che regola la crescita dei sottoalberi;
- unpruned: proprietà booleana che stabilisce se il bean debba costruire un albero senza effettuare alcuna potatura.

5.3.3) Il componente NeuralN

Lo scopo di questo componente è classificare i dati facendo ricorso ad una rete neurale back propagation.

Esso estende la classe `java.awt.Canvas`, e incapsula ed espande le funzionalità della classe `weka.classifiers.neural.NeuralNetwork`.

Questo componente, come il precedente, può operare in quattro modalità differenti:

- può elaborare un unico dataset per effettuare il training del classificatore;
- può elaborare un unico dataset per effettuare il testing di un classificatore pre-

cedentemente costruito;

- può elaborare due dataset distinti (uno per il training e uno per il testing);
- può elaborare un unico dataset da utilizzare sia per il training che per il testing, utilizzando la Cross Validation o mediante divisione percentuale del dataset originario.

Il componente *NeuralN* espone un solo metodo, `dataset_EventPerformed`, che ha come argomento un evento del tipo `DatasetEventObject` contenente i dataset da elaborare. Il metodo risponde all'evento dando inizio all'elaborazione dei dataset ricevuti.

Inoltre il componente espone un'unica proprietà (fig.5.9): *trainingTime*. Tale proprietà definisce il numero di epoche attraverso cui avviene l'addestramento della rete neurale.

Figura 5.9 Il componente *NeuralN* e la sua proprietà

5.3.4) Il componente EMCluster

Lo scopo di questo componente è elaborare i dati facendo ricorso ad un algoritmo di clustering.

Esso estende la classe `java.awt.Canvas`, e incapsula ed espande le funzionalità della classe `weka.clusterers.EM`, che implementa l'algoritmo di clustering EM (*Estimation Maximization*).

Questo componente, come il precedente, può operare in quattro modalità differenti:

- può elaborare un unico dataset per effettuare il training del classificatore;
- può elaborare un unico dataset per effettuare il testing di un classificatore precedentemente costruito;
- può elaborare due dataset distinti (uno per il training e uno per il testing);
- può elaborare un unico dataset da utilizzare sia per il training che per il testing, utilizzando la Cross Validation o mediante divisione percentuale del dataset originario.

Il componente *EMCluster* espone un solo metodo, `dataset_EventPerformed`, che ha come argomento un evento del tipo `DatasetEventObject` contenente i dataset da elaborare. Il metodo risponde all'evento dando inizio all'elaborazione dei dataset ricevuti.

Le proprietà esposte da questo bean, e illustrate in figura 5.10, sono:

- maxIterations: rappresenta il massimo numero di iterazioni da effettuare;
- minStdDev: rappresenta la minima deviazione standard ammissibile;
- numClusters: stabilisce il numero di cluster. Se il valore è posto a -1 il numero di cluster viene selezionato automaticamente dall'algoritmo mediante Cross Validation;
- seed: rappresenta il "seme" per la generazione di numeri random.

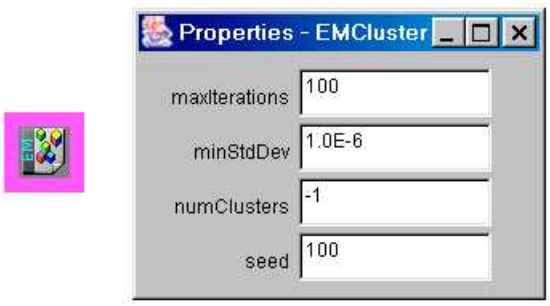


Figura 5.10 Il componente *EMCluster* e le sue proprietà

5.3.5) Il componente DataStreamMonitor

Lo scopo di questo componente è visualizzare il flusso dei dati tra gli altri componenti ed eventualmente salvare tali dati in un file formato testo. In altre parole questo bean rappresenta una sonda (*probe*).

Esso estende la classe `java.awt.Canvas`, e incapsula ed espande le funzionalità della classe `weka.filters.AllFilter`.

I metodi esposti dal componente *DataStreamMonitor* sono:

- `startDatasetStreamOutput()`, che utilizzando un evento `DatasetEventObject` invia ai bean registrati presso di esso i dataset di Train e di Test ricevuti;
- `obtainDatasetStream()`: tale metodo è necessario per il funzionamento del bean in BeanBuilder, e in tale ambiente sostituisce il metodo `startDatasetStreamOutput()` (si veda il paragrafo 5.4.2);
- `dataset_EventPerformed (DatasetEventObject dso)`, il cui argomento è un evento contenente i dataset ricevuti. Il metodo risponde all'evento visualizzando i dataset tramite il metodo `openInfoWindow()` e rilanciandolo tramite il metodo `startDatasetStreamOutput()`;
- `openInfoWindow()`, che apre una finestra per visualizzare i dataset captati dal bean (fig. 5.11);
- `changeVisualization()`, che cambia in modalità alternativa (on/off) lo stato della proprietà *visualization*.

Inoltre il componente espone un'unica proprietà (fig.5.12): *visualization*. Tale proprietà stabilisce se debba essere aperta o meno una finestra per visualizzare i dataset captati dal bean.

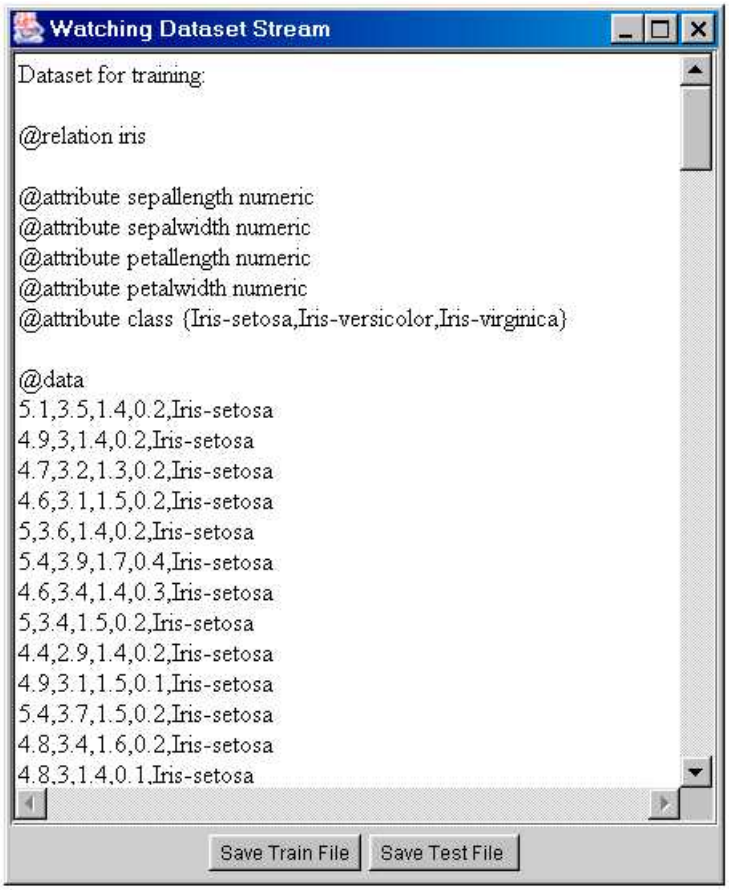


Figura 5.11 La finestra visualizzata dal componente *DataStreamMonitor*

Figura 5.12 Il componente *DataStreamMonitor* e la sua proprietà

5.3.6) L'evento DatasetEventObject

Per consentire il collegamento dei precedenti bean si è progettato ed implementato l'evento *DatasetEventObject*, che propaga una informazione racchiusa in tre parametri: il parametro obbligatorio contenente un riferimento all'oggetto che ha generato l'evento e due parametri contenenti rispettivamente il dataset di training (*TrainDataset*) e il dataset di testing (*TestDataset*). Questi ultimi due parametri sono oggetti del tipo `weka.core.Instances`, e uno dei due può assumere valore *null* se il dataset da comunicare agli altri bean è unico.

In base a quanto previsto dalle specifiche Java Beans, la classe `DatasetEventObject` estende la classe `EventObject`, e fornisce due metodi pubblici per l'accesso ai parametri contenenti i dataset.

Inoltre è stata definita l'interfaccia di programmazione degli ascoltatori di eventi `DatasetListener`, che estende `EventListener`, in base ai dettami delle specifiche Java Beans. Tale interfaccia prevede un unico metodo, `dataset_EventPerformed (DatasetEventObject e)`, che gli ascoltatori dell'evento `DatasetEventObject` devono necessariamente implementare.

5.4) L'ambiente di esecuzione del PSE visuale

I componenti illustrati nel precedente paragrafo, essendo stati realizzati nel pieno rispetto delle specifiche *Java Beans*, funzionano correttamente in qualsiasi ambiente che rispetti tali specifiche.

Si è quindi deciso di testarne il funzionamento in tre diversi ambienti. Per un maggior dettaglio sull'utilizzo dei componenti in ciascuno di questi ambienti si veda l'appendice C.

Per un corretto funzionamento del PSE, è necessario che sul computer su cui si

intende utilizzarlo sia presente il file *WEKA.JAR*, contenente le classi di WEKA.

5.4.1) L'ambiente BDK 1.1 (BeanBox)

Il JavaBeans Development Kit (BDK) [22] è stato realizzato dalla Sun Microsystems per supportare lo sviluppo dei componenti JavaBean, e per fungere da riferimento sia per gli sviluppatori di bean che per gli sviluppatori di Builder Tool. Esso è stato realizzato per scopi dimostrativi ed educativi, non per essere un ambiente di sviluppo completo.

Attualmente il BDK è alla versione 1.1, e non sono previsti ulteriori aggiornamenti. La Sun ormai lo considera un progetto abbandonato, in quanto sta dedicando le proprie energie allo sviluppo del suo successore: il *Bean Builder*.

Per il funzionamento del BDK 1.1 è necessario che sia installato il Java 2 Standard Edition SDK 1.2 o successive versioni.

Il BDK 1.1 è un pacchetto contenente un set di 16 esempi funzionanti di bean e un tool scritto in java, il *BeanBox*. Quest'ultimo è un contenitore che consente di testare i bean e di connetterli insieme, e funge anche da esempio su come realizzare un contenitore per i bean: infatti il BeanBox stesso è un bean.

A scopo educativo ed esemplificativo, per guidare i programmatori alla realizzazione e all'uso dei bean, il BDK è fornito con il codice sorgente di tutto il software in esso contenuto, rilasciato in modalità free con licenza SPL (*Sun Public License*).

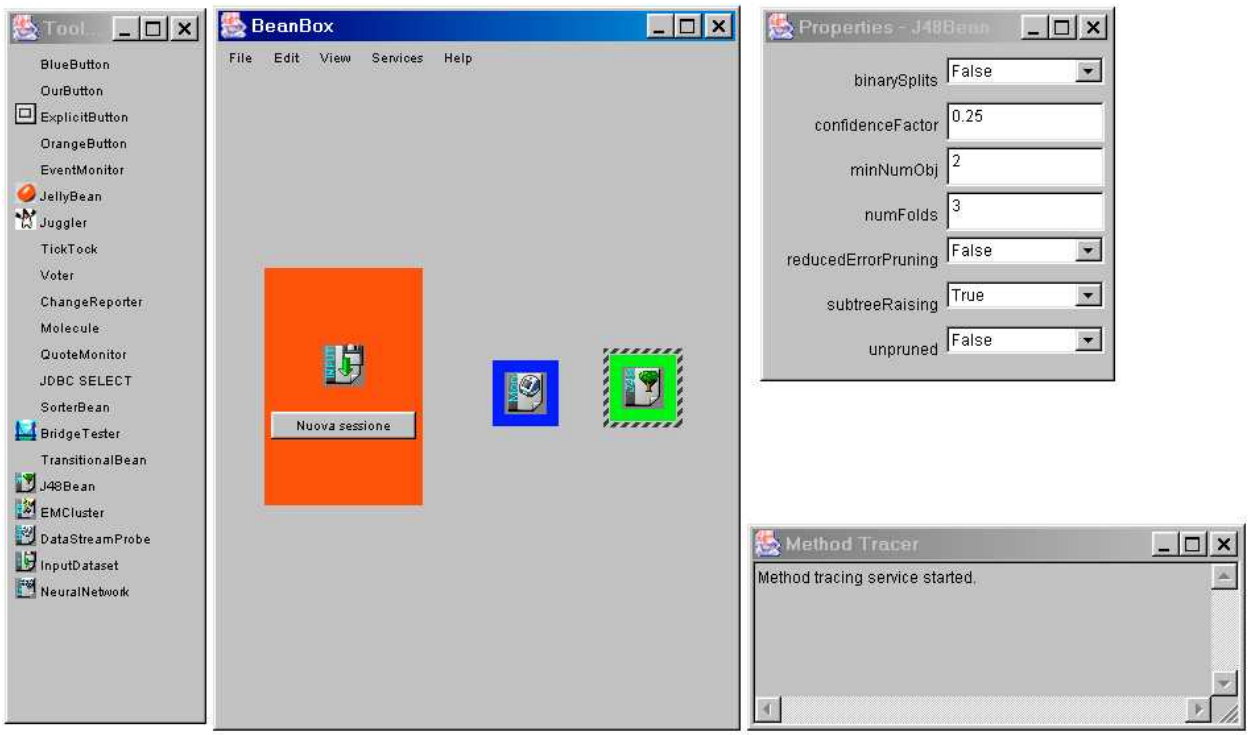


Figura 5.13 Il BeanBox

In figura 5.13 è riportata la schermata principale di questo applicativo. La finestra più grande, al centro, è il contenitore BeanBox, che rappresenta l'ambiente dove accadono le azioni; in esso sono stati inseriti alcuni componenti del PSE. A sinistra si trova la palette, contenente le icone dei componenti del PSE (in basso) e le icone dei 16 bean di esempio del BDK. A destra si trova l'editor di proprietà dei bean, che permette di vedere e cambiare il valore delle proprietà associate al bean selezionato. A tal proposito si è reso necessario apportare alcune modifiche a due classi del BDK: normalmente l'editor di proprietà del BeanBox visualizza solamente le proprietà read/write del bean, per cui si è deciso di modificarlo per consentire anche la visualizzazione delle proprietà read only di un bean (le modifiche sono riportate nell'appendice B). Nell'esempio sono visualizzate le proprietà del *J48Bean* in quanto nel BeanBox è stato selezionato tale componente (la selezione

è evidenziata dalla cornice tratteggiata che circonda il bean). Quando viene istanziato un nuovo oggetto, le sue proprietà assumono il valore di default, in quanto viene invocato il costruttore di default (costruttore senza argomenti) della corrispondente classe.

La finestra del BeanBox è fornita di menu, di cui i principali sono:

- File: permette di gestire i file contenenti i bean. In particolare, la voce *Loadjar...* consente di caricare un file jar contenente uno o più bean. La voce *Save...* consente di salvare tutto il contenuto del BeanBox, incluso lo stato attuale dei bean ivi presenti, secondo i dettami sulla serializzazione presenti nelle specifiche Java Beans, e la voce *Load...* consente di ripristinare ciò che si è salvato in precedenza.
- Edit: permette di gestire i bean. Oltre a consentire le normali operazioni di taglia, copia e incolla, mediante la voce *Bind property...* permette di legare le proprietà bound dei bean. Inoltre la voce *Events* consente la connessione dell'evento di un bean ai metodi di altri bean: in figura 5.14 è illustrata la scelta dell'evento; in figura 5.15 è illustrata la successiva fase della connessione in cui, mediante un segmento rosso, si collega il bean sorgente al bean target; poi (fig. 5.16) si sceglie il metodo del bean target che deve essere attivato quando il bean sorgente lancia l'evento; al termine della procedura il BeanBox genera automaticamente la classe *Adapter* (fig. 5.17), che ha il compito di gestire il collegamento tra i due bean che sono stati connessi.
- View: permette di visualizzare o meno gli *invisible bean* (*Hide/Show Invisible Beans*), e permette di passare dalla visualizzazione del progetto a quella che si avrebbe durante l'esecuzione (*Disable/Enable Design Mode*). Nel BeanBox un *invisible bean* è invisibile agli occhi dell'utente (cioè non ha GUI) solo durante

la fase di *run time*, mentre durante la fase di *design time* ha un aspetto standard fornitogli dal BeanBox stesso (appare come un pannello grigio con su scritto il nome del componente).

Uno dei vantaggi di un ambiente di sviluppo di tipo Java Beans è che i bean vengono eseguiti immediatamente, e ciò consente di vedere e testare subito, nell'ambiente di sviluppo stesso, le funzionalità del programma che si sta realizzando, anziché dover utilizzare il tradizionale ciclo di modifica, compilazione ed esecuzione. Nel BeanBox si ritrova pienamente tale vantaggio.

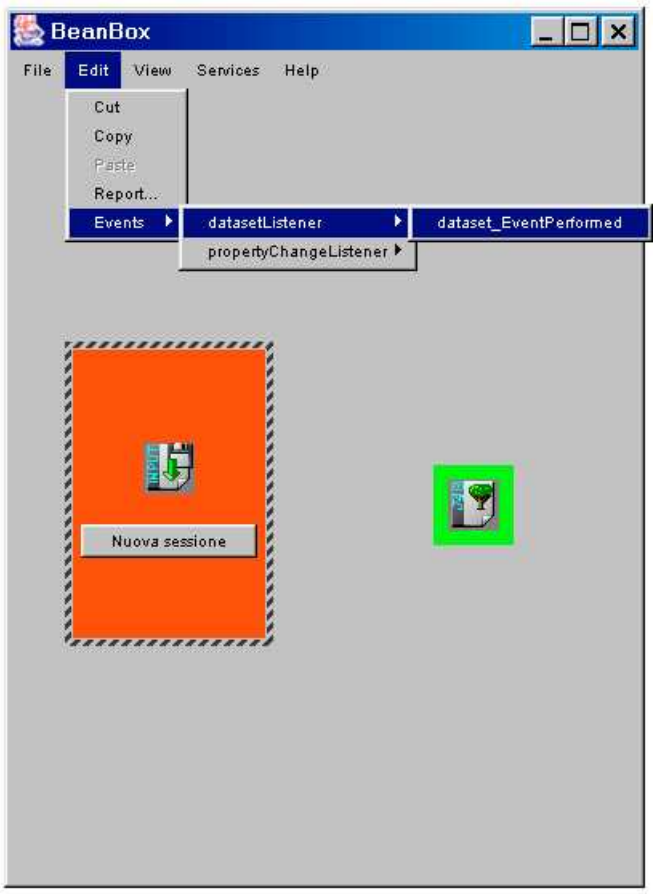


Figura 5.14 Scelta dell'evento del bean sorgente

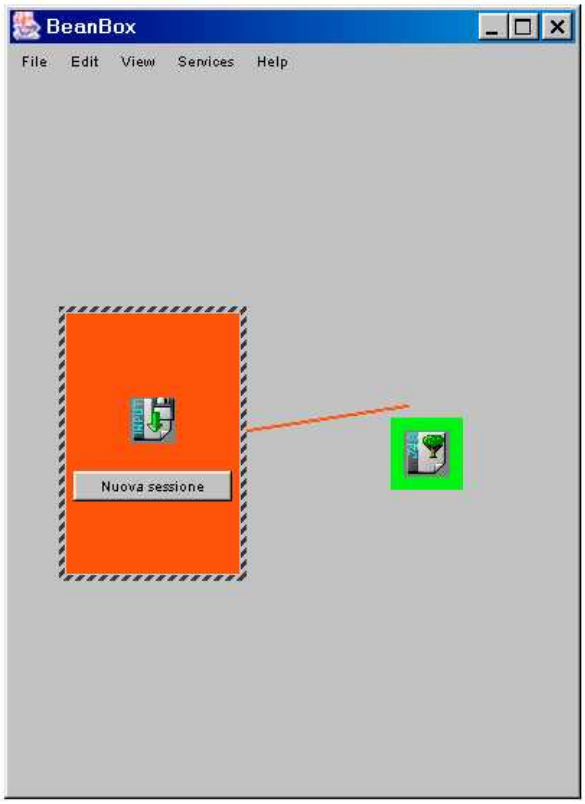


Figura 5.15 Connessione del componente sorgente al componente target

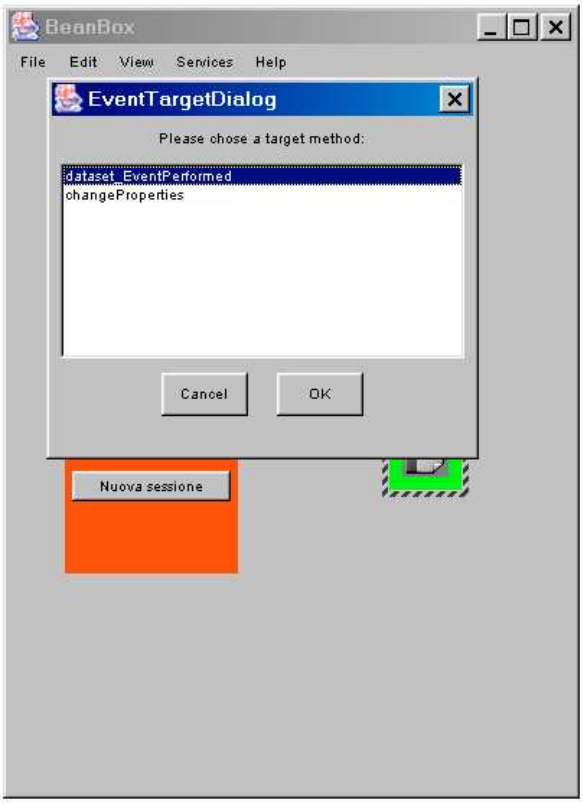


Figura 5.16 Scelta del metodo del componente target

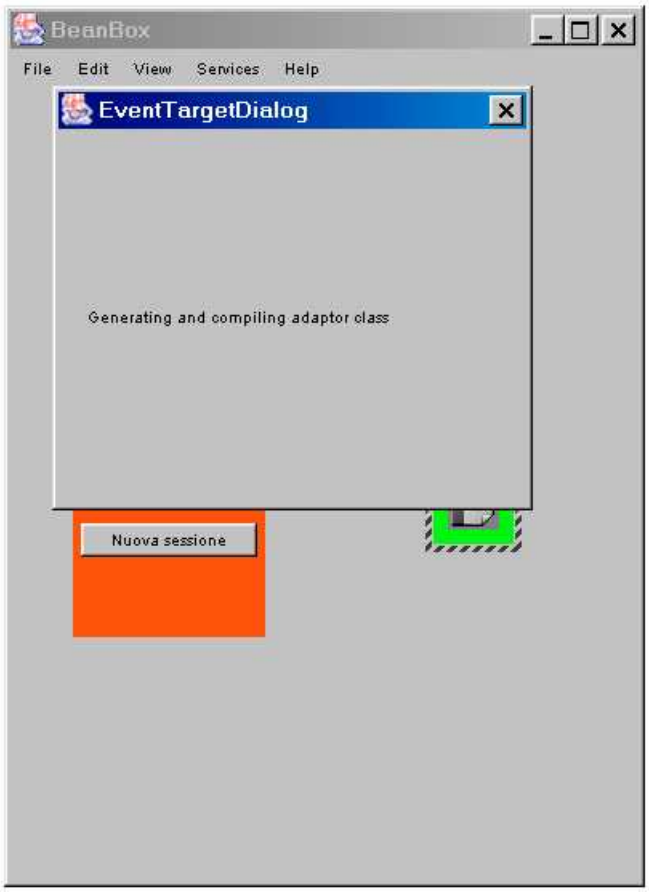


Figura 5.17 Creazione automatica della classe *Adapter*

5.4.2) L'ambiente Bean Builder

Il Bean Builder v 1.0 Beta [23] è un tool che consente l'assemblamento visuale di una applicazione tramite l'istanziazione e il settaggio delle proprietà di componenti basati sull'architettura *Java Beans*. Il suo comportamento dinamico è insito nella relazione di collegamento tra i bean, che rappresenta la gestione degli eventi e le chiamate ai metodi tra gli oggetti che compongono una applicazione.

Per il funzionamento del Bean Builder è necessario che sia installato il Java 2 Standard Edition SDK 1.4 o successive versioni.

Come per il BeanBox, anche per il Bean Builder lo scopo principale per cui è stato realizzato dalla Sun Microsystems è educativo ed esemplificativo, e a tal

motivo insieme ad esso è fornito anche il suo codice sorgente, rilasciato con licenza SPL (*Sun Public License*).

In figura 5.18 è riportata la schermata principale di questo applicativo.

La finestra in basso a destra è il *Designer*, in cui i bean vengono istanziati, manipolati e connessi. Esso è il contenitore radice di tutti gli oggetti grafici che vengono assemblati (riportati nella struttura ad albero sulla sinistra), ed "avvolge" ogni applicazione che viene costruita nel Bean Builder; nell'esempio della figura vi sono stati inseriti alcuni componenti del PSE.

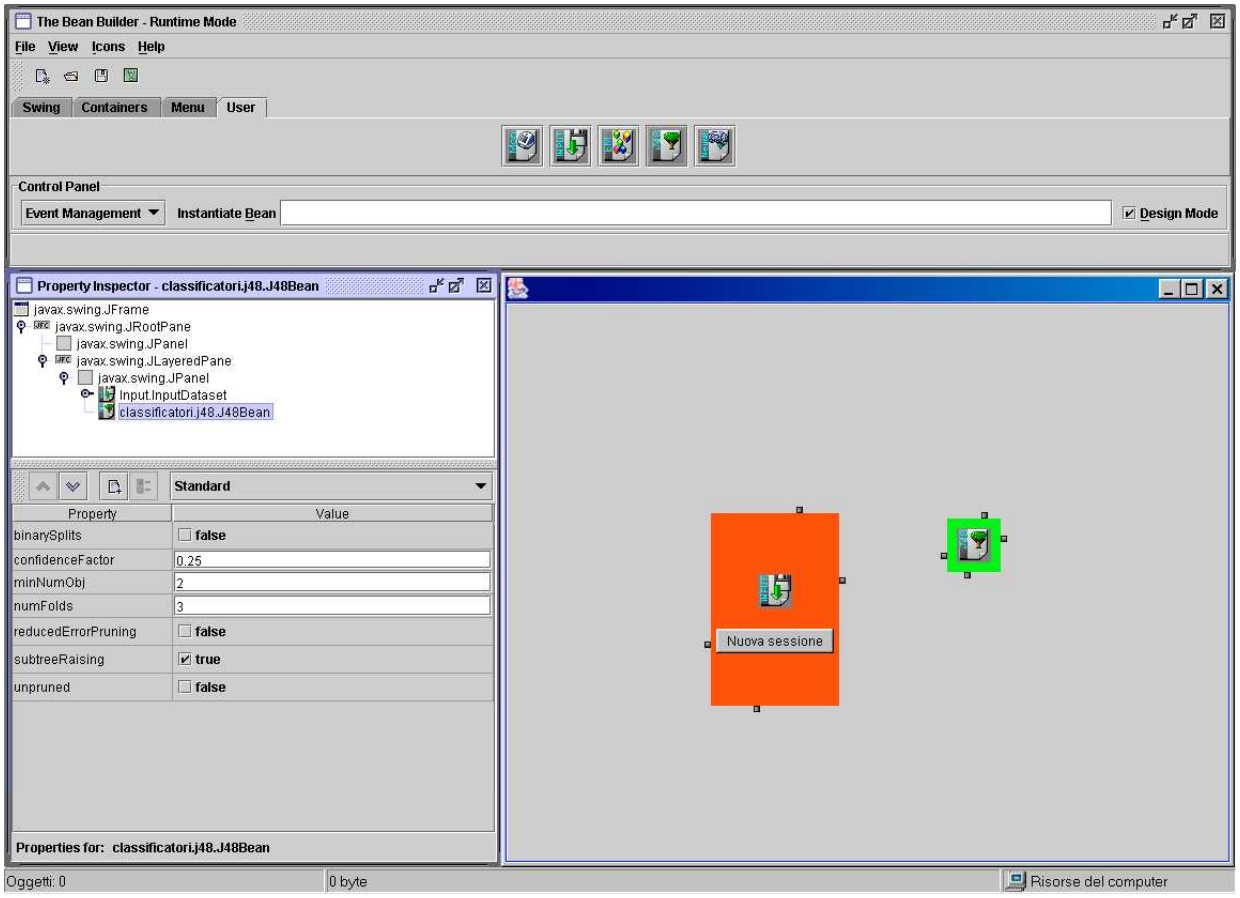


Figura 5.18 Il Bean Builder

A sinistra si trova l'editor di proprietà (*Property Inspector*), che permette di vedere e cambiare il valore delle proprietà dell'oggetto corrente; quando viene istanziato un nuovo oggetto, le sue proprietà assumono il valore di default, in quanto viene invocato il costruttore di default (costruttore senza argomenti) della corrispondente classe, secondo i dettami delle specifiche *Java Beans*. Ogni riga rappresenta una istanza della classe *PropertyDescriptor*. Il combo box presente al di sopra di tali righe consente di selezionare il tipo di proprietà che deve essere visualizzato nella sottostante tabella (Standard, Bound, Read Only, eccetera).

In alto, sotto i menu e la *toolbar*, si trova la *palette*, di cui è stato selezionato il tab *User*, contenente le icone dei componenti del PSE. Essa è specificata tramite un documento esterno editabile in formato XML.

Al di sotto della *palette* si trova il *pannello di controllo*. Il check box a destra consente di passare dalla fase di *design time* alla fase di *run time*, e viceversa. Il combo box a sinistra consente di alternare il funzionamento del *Designer* tra due differenti modalità:

- Event Management, che consente di editare le interazioni dinamiche (basate sugli eventi) tra gli oggetti;
- Layout Editing, che consente di editare le caratteristiche di mutuo ridimensionamento (*resizing*) dinamico degli oggetti.

Ogni componente inserito nel *Designer* possiede nove "maniglie" chiare e quattro scure (i quattro piccoli quadratini visibili intorno ai componenti in figura 5.18 e in figura 5.19). Le prime consentono di trascinarlo o di ridimensionarlo dinamicamente: basta tenere premuto il mouse sopra di esse e trascinarlo. Le seconde consentono di legare le proprietà bound dei bean e di connettere l'evento di un bean ai metodi di altri bean, il tutto sempre premendo il mouse sopra di esse e trascinan-

dolo: in fig. 5.19 è illustrata la fase della connessione in cui, mediante una freccia rossa, si collega il bean sorgente al bean target; in fig. 5.20 è visualizzata la finestra che compare nella fase successiva, in cui si sceglie l'evento del bean sorgente che si vuole legare al bean target, e il metodo di quest'ultimo che deve essere attivato quando viene lanciato l'evento. Al termine della procedura viene automaticamente generata la classe *Adapter*, che ha il compito di gestire il collegamento tra i due bean che sono stati connessi; a tale scopo, per creare un oggetto *EventHandler* che funga da ascoltatore degli eventi, il Bean Builder utilizza le *Dynamic Proxy API*, introdotte con la release 1.3 di Java 2 per sintetizzare un *listener* di tipo arbitrario.

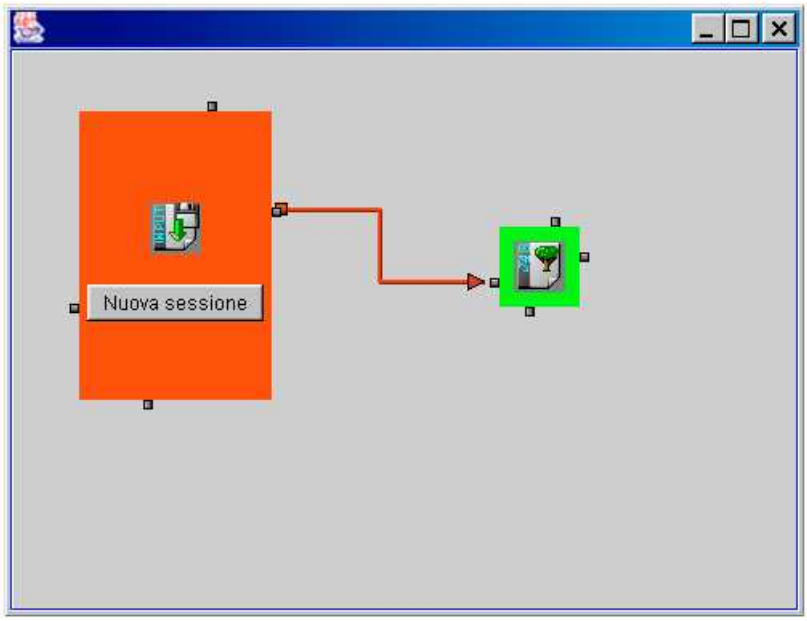


Figura 5.19 Connessione del bean sorgente al bean target nel Bean Builder

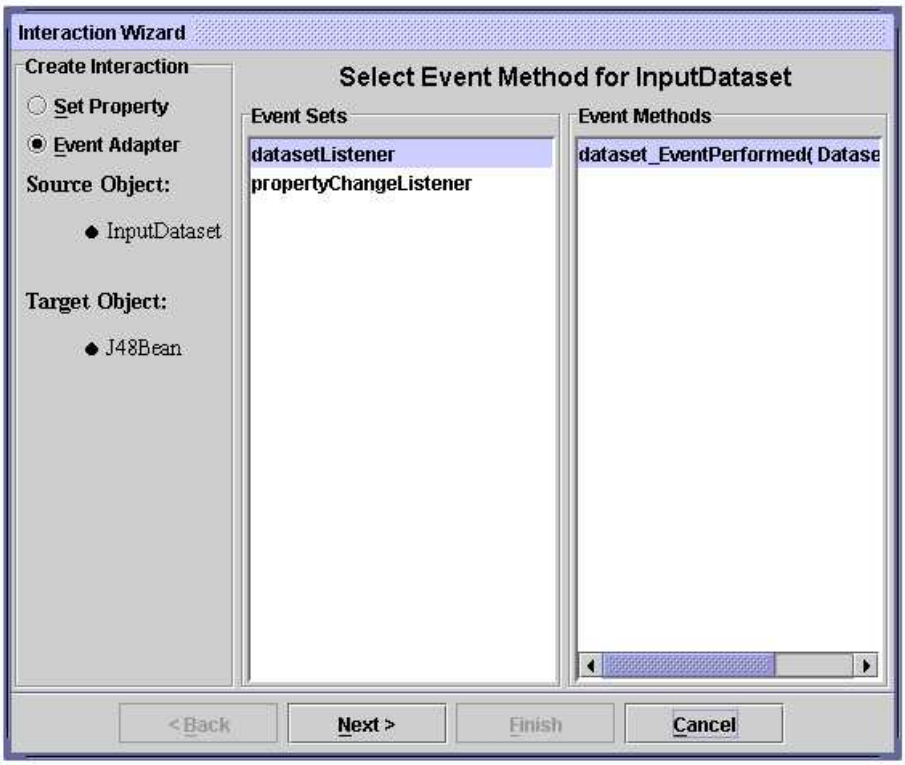


Figura 5.20 Scelta dell'evento del bean sorgente e del metodo del bean target

Come nel BeanBox, un *invisible bean* è invisibile agli occhi dell'utente (cioè non ha GUI) solo durante la fase di *run time*, mentre durante la fase di *design time* ha un aspetto standard fornitogli dal Bean Builder, ed appare come un pannello grigio con su scritto il nome del componente. Inoltre anche nel Bean Builder, come nel BeanBox, i bean vengono eseguiti immediatamente, e ciò consente di far funzionare subito i componenti del PSE, sia durante la fase di *run time* che durante la fase di *design time*.

Il Bean Builder consente il salvataggio ed il ripristino in formato XML dello stato di una applicazione che si sta realizzando con esso. Infatti il Bean Builder usa il concetto di *Long Term Persistence* per memorizzare lo stato dei bean: questo meccanismo di persistenza, che consiste nello scrivere lo stato di tutti i bean in un file editabile formato XML, è una nuova caratteristica introdotta con la piattafor-

ma Java 2 Standard Edition release 1.4, e rappresenta una evoluzione del vecchio concetto di serializzazione dei bean implementato in BeanBox.

Il tutorial realizzato da Mark Davidson, presente nel pacchetto contenente il Bean Builder, illustra un esempio di utilizzo del Bean Builder per la costruzione di una semplice applicazione senza che si debba scrivere neanche una linea di codice.

Nel complesso il Bean Builder si è rivelato un prodotto potenzialmente utile, ma ancora non pienamente funzionante. Infatti durante il suo utilizzo si sono riscontrati numerosi *bug*: tra gli altri, c'è da segnalare che il Bean Builder non è in grado di utilizzare il metodo `startDatasetStream()` di *InputDataset* e il metodo `startDatasetStreamOutput()`di *DataStreamMonitor* come sorgente dell'evento *DatasetEventObject*, contravvenendo alle specifiche *Java Beans*, per cui si è dovuto aggiungere il metodo `obtainDatasetStream()` al bean *InputDataset* e al bean *DataStreamMonitor* per consentire il funzionamento di tali componenti anche nel Bean Builder.

5.4.3) L'ambiente NetBeans

Il NetBeans IDE release 3.4.1 per Windows [21] non è un ambiente specifico per l'utilizzo dei bean, come i due precedenti, ma è un più generale ambiente di sviluppo integrato (*Integrated Development Environment*, IDE) che fornisce una interfaccia GUI per realizzare applicazioni, completa di editor di testi, GUI Editor per la creazione visuale delle applicazioni, debugger ed help on line. Questo ambiente, rilasciato gratuitamente con licenza SPL (*Sun Public License*), è stato utilizzato per lo sviluppo dei componenti e dell'evento del PSE visuale oggetto della tesi.

Il NetBeans, originariamente chiamato Xelfi, è nato nel 1996 come progetto di

uno studente della Repubblica Ceca con lo scopo di realizzare un ambiente integrato per Java scritto interamente in Java. Una compagnia, chiamata NetBeans, fu formata intorno a tale progetto per commercializzare le versioni 2.0 e 2.1 di questo ambiente. Nell'ottobre 1999, mentre era in fase di beta testing la versione 3.0, tale compagnia fu acquisita dalla Sun Microsystems, che successivamente utilizzò lo stesso ambiente IDE della versione 3.0 per rilasciare *Forte for Java Community Edition* IDE. Infine, nel giugno 2000 la Sun ha trasformato NetBeans IDE in un progetto Open Source.

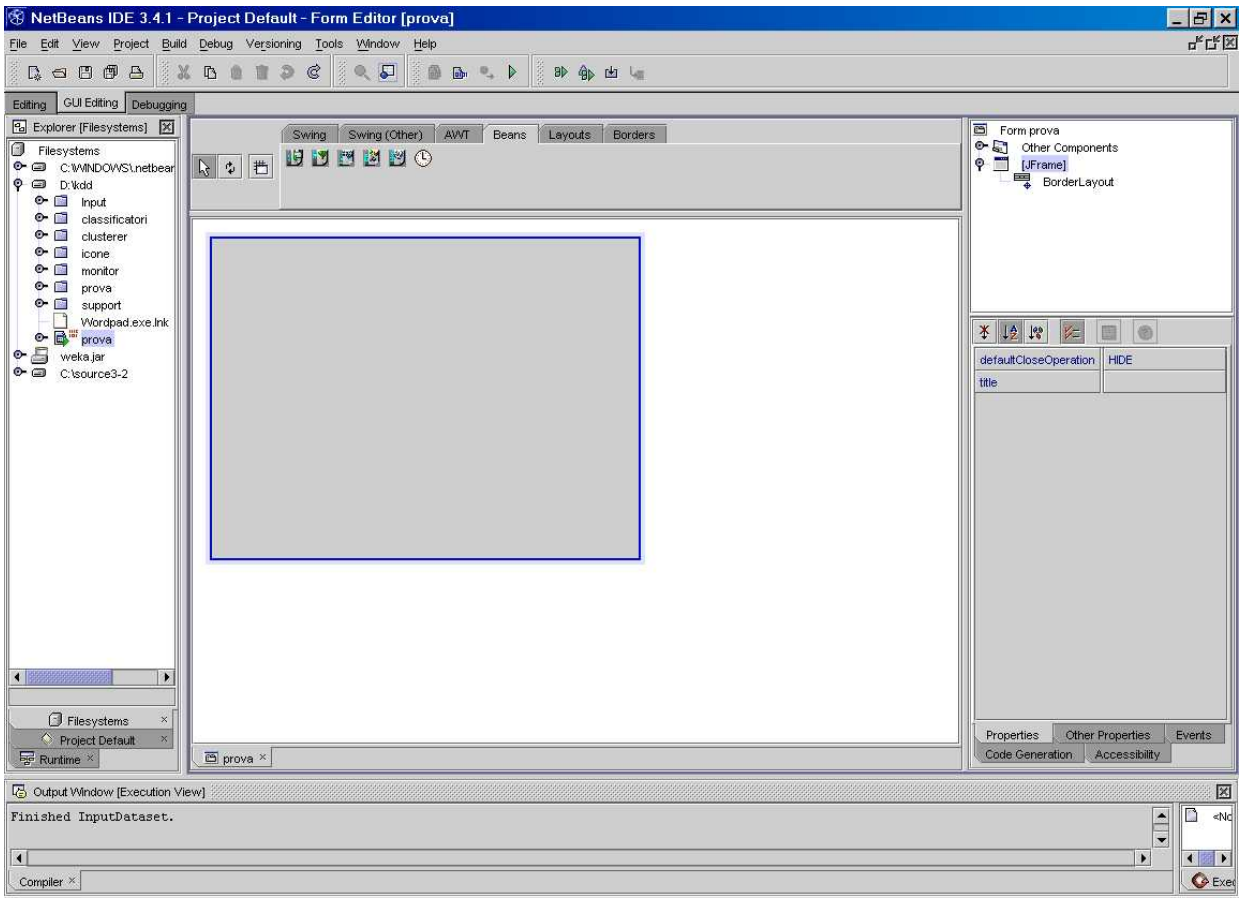


Figura 5.21 Il NetBeans IDE release 3.4.1

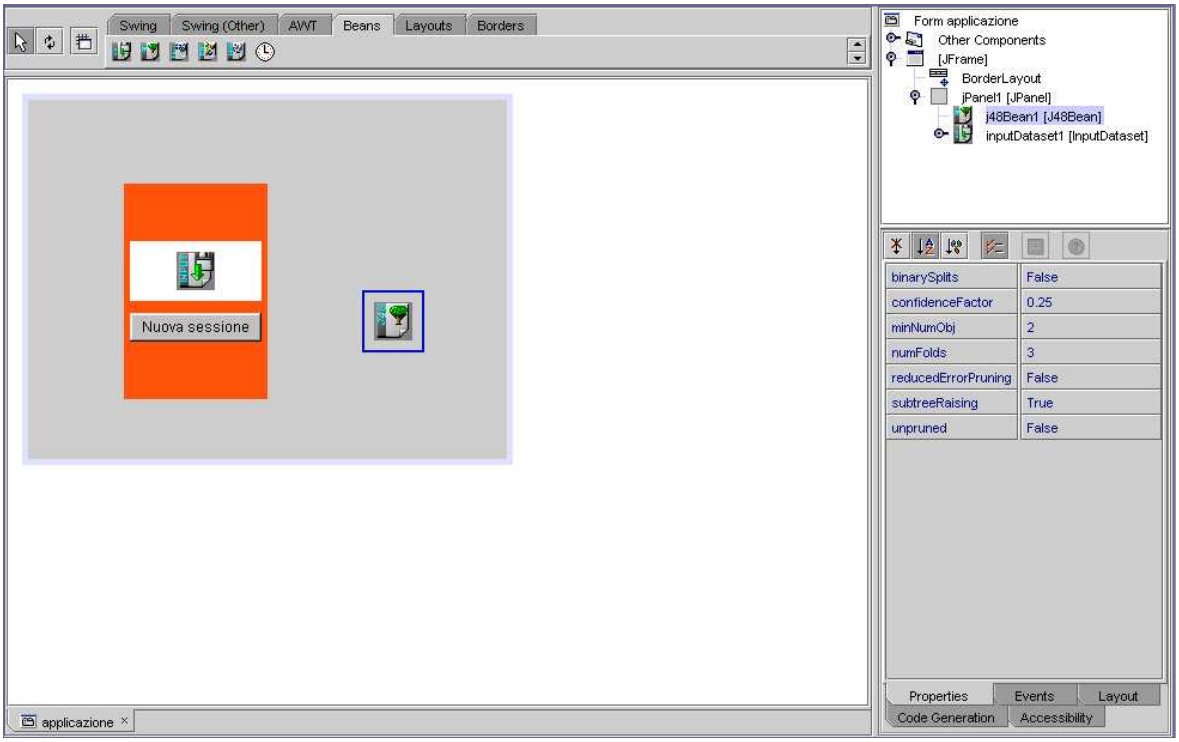


Figura 5.22 La realizzazione di un applicativo con NetBeans IDE

Il NetBeans IDE release 3.4.1, essendo scritto in Java, per il suo funzionamento richiede che il computer su cui viene utilizzato abbia installato il *Java 2 Software Development Kit* versione 1.3 o successiva.

In figura 5.21 è riportata la schermata del GUI Editor di questo applicativo.

A sinistra si trova il pannello Explorer, che consente di gestire file e progetti, e permette di caricare librerie di classi e bean.

Al centro si trova lo spazio per disegnare una nuova applicazione, mentre in alto, sotto i menu e la *toolbar*, si trova la *palette*, in cui è stato selezionato il tab *Beans*, contenente le icone dei componenti del PSE.

A destra si trova l'editor di proprietà, che permette di vedere e cambiare il valore delle proprietà dell'oggetto corrente. Quando viene istanziato un nuovo oggetto, le sue proprietà assumono il valore di default, in quanto viene invocato il costruttore

di default (costruttore senza argomenti) della corrispondente classe, secondo i dettami delle specifiche *Java Beans* (si veda l'esempio di fig. 5.22, in cui è selezionato il componente *J48Bean*). La struttura ad albero al di sopra di esso rappresenta la gerarchia degli oggetti che compongono l'applicazione in costruzione.

Anche in questo ambiente, come nei due precedenti, la connessione tra i bean avviene per via grafica: dopo aver fatto click sull'icona *Connection Mode*, si selezionano prima il bean sorgente e poi il bean target; successivamente compare una finestra che guida nella scelta dell'evento del bean sorgente che si intende legare al bean target, e nella scelta del metodo di quest'ultimo che deve essere attivato quando viene lanciato l'evento. Al termine della procedura viene automaticamente generata la classe *Adapter*, che ha il compito di gestire il collegamento tra i due bean che sono stati connessi.

Il NetBeans consente la realizzazione di applicazioni e applet con i bean ma, a differenza dei due precedenti ambienti, non è in grado di eseguire i bean immediatamente, e per poterli utilizzare è necessario prima compilare il codice del progetto che si sta realizzando. Per questo motivo, è compito del programmatore preoccuparsi del salvataggio dello stato dei bean durante l'esecuzione dell'applicativo (o dell'applet) realizzato, a differenza di quanto avviene negli altri due ambienti esaminati, in cui è direttamente l'ambiente a preoccuparsi del meccanismo della persistenza dei bean. Tutto ciò implica che, una volta compilato e distribuito, lo schema del progetto resterà fisso e immutabile, e all'utente non sarà consentito variarlo dinamicamente come negli altri due ambienti; ed inoltre, per consentire variazioni delle proprietà dei bean durante l'esecuzione dell'applicativo, sarà necessario prevedere appositi metodi e strumenti. Di contro, per far funzionare tale progetto non occorrerà il *Java 2 Software Development*

Kit, ma sarà sufficiente una JVM.

La tabella che segue riassume le principali caratteristiche dei tre ambienti esaminati.

	BDK 1.1 (BeanBox)	**Bean Builder v 1.0 Beta**	**NetBeans IDE release 3.4.1**
Prezzo	Gratis	Gratis	Gratis
Documentazione d'uso	Buona	Sufficiente	Ottima
Open Source	Sì	Sì	Sì
Usabilità	Intuitiva	Talvolta macchinosa	Semplice ed intuitiva
Stabilità	Qualche bug	Molti bug	Ottima
Persistenza dei bean	Automatica	Automatica (ma non sempre funziona)	Demandata al programmatore
Creazione automatica della classe *Adapter*	Sì	Sì	Sì
Uso immediato dei bean	Sì	Sì	No

Nel capitolo seguente si illustrerà l'applicazione del PSE visuale progettato in alcuni tipici campi di utilizzo del data mining.

C A P I T O L O 6

UTILIZZO DEL PSE VISUALE PER DATA MINING

In questo capitolo si illustra l'utilizzo del PSE visuale per data mining in cinque tipici campi di indagine. L'ambiente utilizzato è il BeanBox. Per i dettagli sull'utilizzo dei componenti in questo ambiente e negli altri si veda l'appendice C.

Il computer su cui si è operato presenta la seguente configurazione:

- processore AMD Athlon 900 MHz;
- hard disk da 13 GB;
- 128 MB di memoria RAM;
- Microsoft Windows 98 Second Edition;
- Java 2 Runtime Environment SE v1.4.1_02.

I dati utilizzati, come nei capitoli 3 e 4, provengono da *dataset* già pronti all'uso, disponibili nel web per scopi didattici o per testare il funzionamento di software per data mining.

6.1) Data mining in campo tecnico

- Dataset utilizzato: sheesle2.dat
- Provenienza: SED-NIST [27]
- Autori: John Sheesley (Gen. Electric)

  article in Experiments in Industry (edited by Snee, Hare, & Trout)

- Descrizione dei dati: i dati rappresentano un confronto tra differenti processi produttivi per la realizzazione di filamenti delle lampade ad incandescenza. Di seguito è riportato un frammento del dataset:

```
Weld,Plant,Speed,Shift, average of defect. lead wires / hour
1,    1,     1,      1,      28.4
2,    1,     1,      1,      21.9
1,    1,     1,      2,      36.8
2,    1,     1,      2,      19.2
1,    1,     1,      3,      28.2
2,    1,     1,      3,      26.6
1,    1,     2,      1,      30.4
2,    1,     2,      1,      25.1
```

- Componenti utilizzati: InputDataset, EMCluster, J48Bean.
- Risultati e commenti:

Lo scopo di questa analisi può essere duplice. Se viene effettuata con continuità sui dati provenienti dai diversi impianti di produzione, può costituire un efficace monitoraggio del processo produttivo, e segnalare l'insorgere di eventuali anomalie: si saprebbe non solo che qualcosa non va, perché è aumentato il numero medio di difetti per ogni ora, ma si avrebbero anche indicazioni sulle più probabili cause, e di conseguenza si potrebbe provvedere con rapidità ed efficienza alla rimozione del problema.

Ma questa analisi può anche essere utilizzata per cercare di capire quali sono i punti cruciali della produzione, su cui cercare di intervenire per migliorarla, e come si possono combinare i diversi parametri al fine di ottenere un più efficiente processo produttivo.

Per quanto riguarda lo svolgimento dell'analisi, c'è da rilevare che il dataset utilizzato contiene solo attributi numerici. La classe J48, inclusa nel componente J48Bean, richiede che l'attributo target sia categorico, per cui si è deciso di trasformare l'attributo "difetti medi per ora" in una serie di intervalli categorici. La scelta del numero e dell'ampiezza degli intervalli è molto importante

per poter assicurare una più precisa elaborazione da parte del componente J48Bean, in quanto una scelta inopportuna porta ad una percentuale di errore di classificazione molto elevata, anche superiore al 70 %. Si è provato ad utilizzare genericamente tre intervalli (def<20, 20 def 30, def>30) e quattro intervalli (def<18, 18 def<24, 24 def<30, def 30), ma il modo migliore per effettuare tale scelta è risultato quello di elaborare il dataset con il componente EMCluster, e utilizzare i cluster ottenuti come riferimento per la creazione degli intervalli (def 18, 18<def 22.7, 22.7<def 27.4, def>27.4).

Un ulteriore miglioramento della precisione del modello costruito si potrebbe ottenere con opportune pre-elaborazioni dei dati, ad esempio eliminando attributi che rappresentano aspetti non modificabili del processo produttivo.

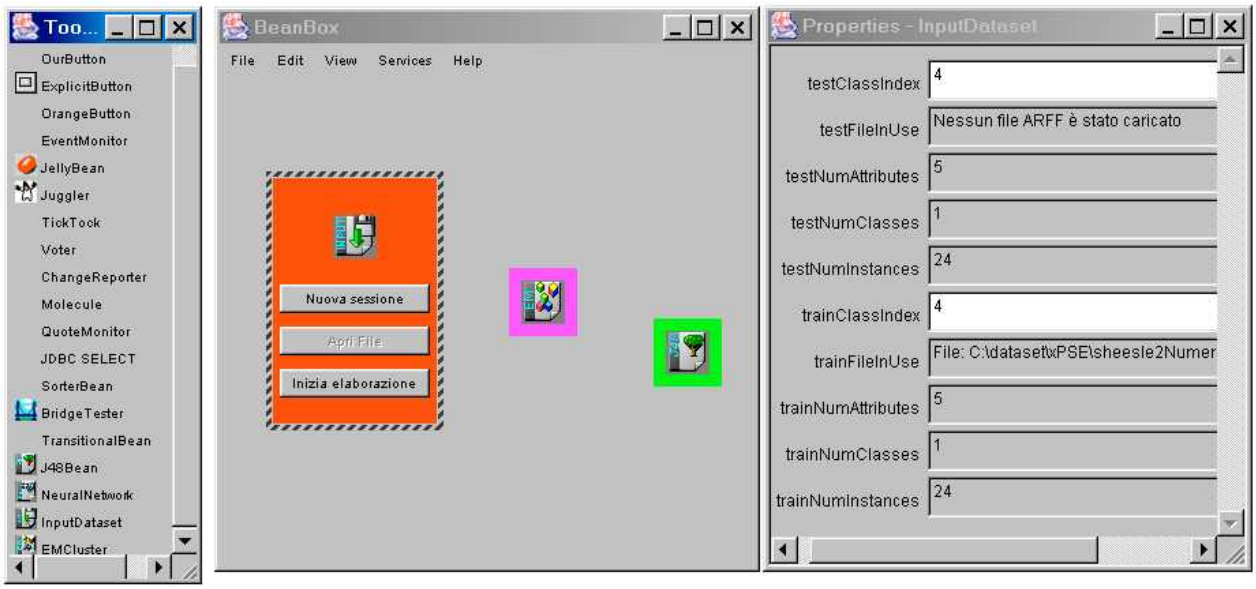


Figura 6.1 L'utilizzo dei componenti nel BeanBox

I risultati dell'analisi dei dati effettuata con il componente EMCluster sono ri-

portati nelle seguenti righe:

```
Informazioni statistiche sull'elaborazione:

EM
==
Number of clusters selected by cross validation: 3

Cluster: 0 Prior probability: 0.25
Attribute: Weld
Normal Distribution. Mean = 2 StdDev = 0.05
Attribute: Plant
Normal Distribution. Mean = 2 StdDev = 0.05
Attribute: Speed
Normal Distribution. Mean = 1.5 StdDev = 0.5
Attribute: Shift
Normal Distribution. Mean = 2 StdDev = 0.8165
Attribute: Defects
Normal Distribution. Mean = 18.05 StdDev = 3.6868

Cluster: 1 Prior probability: 0.5
Attribute: Weld
Normal Distribution. Mean = 1 StdDev = 0.05
Attribute: Plant
Normal Distribution. Mean = 1.5 StdDev = 0.5
Attribute: Speed
Normal Distribution. Mean = 1.5 StdDev = 0.5
Attribute: Shift
Normal Distribution. Mean = 2 StdDev = 0.8165
Attribute: Defects
Normal Distribution. Mean = 27.4417 StdDev = 5.6484

Cluster: 2 Prior probability: 0.25
Attribute: Weld
Normal Distribution. Mean = 2 StdDev = 0.05
Attribute: Plant
Normal Distribution. Mean = 1 StdDev = 0.05
Attribute: Speed
Normal Distribution. Mean = 1.5 StdDev = 0.5
Attribute: Shift
Normal Distribution. Mean = 2 StdDev = 0.8165
Attribute: Defects
Normal Distribution. Mean = 22.7 StdDev = 2.4474
```

I risultati dell'analisi dei dati effettuata con il componente J48Bean sono riportati nelle seguenti righe e in figura 6.2:

```
Informazioni statistiche sull'elaborazione:

J48 pruned tree
------------------
Weld <= 1
|   Shift <= 2
|   |   Plant <= 1: def>27,4 (3.0)
|   |   Plant > 1: 18<def<=22,7 (4.0/1.0)
|   Shift > 2: def>27,4 (4.0/2.0)
Weld > 1
|   Plant <= 1: 18<def<=22,7 (6.0/2.0)
|   Plant > 1
|   |   Shift <= 1: def<=18 (2.0)
|   |   Shift > 1: 18<def<=22,7 (3.0/1.0)
```

```
Number of Leaves  :       6
Size of the tree : 11

=== Summary ===
Correctly Classified Instances          12               50      %
Incorrectly Classified Instances        12               50      %
Kappa statistic                          0.2258
K&B Relative Info Score                638.5381 %
K&B Information Score                   12.2135 bits      0.5089 bits/instance
Class complexity | order 0              47.038  bits      1.9599 bits/instance
Class complexity | scheme             8606.0659 bits    358.5861 bits/instance
Complexity improvement     (Sf)      -8559.028  bits   -356.6262 bits/instance
Mean absolute error                      0.2887
Root mean squared error                  0.4684
Relative absolute error                 79.886  %
Root relative squared error            109.6321 %
Total Number of Instances               24


=== Confusion Matrix ===
 a b c d   <-- classified as
 3 2 1 0 | a = def>27,4
 1 9 0 0 | b = 18<def<=22,7
 2 2 0 1 | c = 22,7<def<=27,4
 0 3 0 0 | d = def<=18

=== Detailed Accuracy By Class ===
TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.5       0.167      0.5       0.5       0.5      def>27,4
  0.9       0.5        0.563     0.9       0.692    18<def<=22,7
  0         0.053      0         0         0        22,7<def<=27,4
  0         0.048      0         0         0        def<=18
```

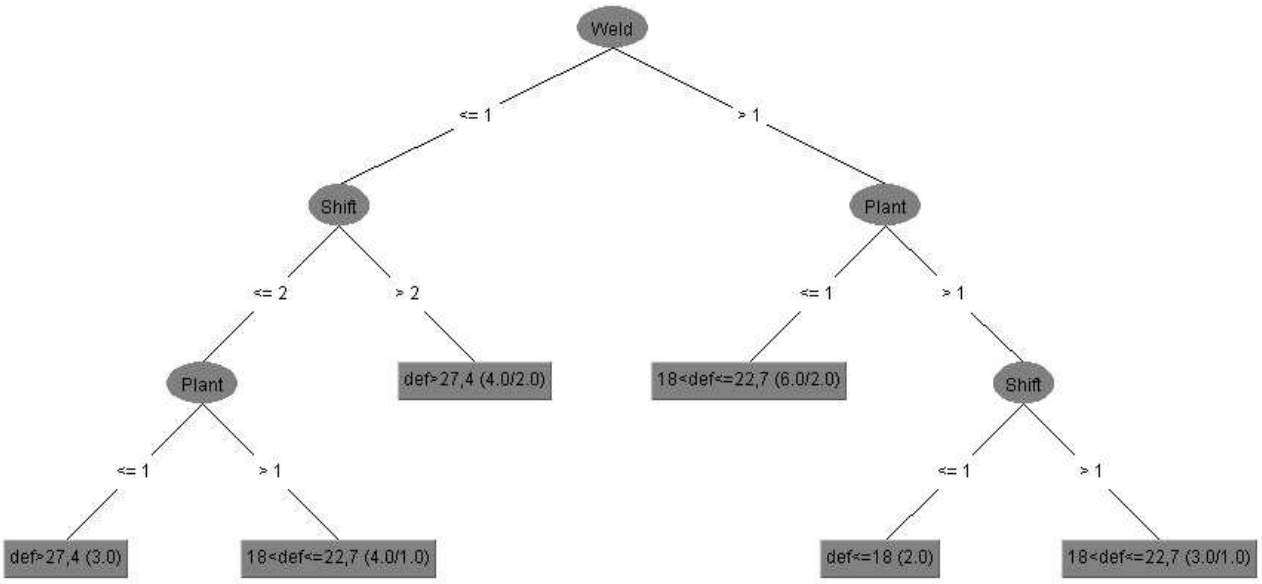


Figura 6.2 L'albero di decisione ottenuto dall'elaborazione con il J48Bean

6.2) Data mining in campo medico

- Dataset utilizzato: heart-statlog
- Provenienza: CMU StatLib Datasets Archive [25]
- Descrizione dei dati: i dati rappresentano diversi attributi di pazienti, raccolti con lo scopo di determinare la presenza o l'assenza di patologie cardiache. Di seguito è riportato un frammento del dataset:

```
age,sex,chest,resting_blood_pressure,serum_cholestoral,fasting_blood_sugar,resting_ele
ctrocardiograph-
ic_results,maximum_heart_rate_achieved,exercise_induced_angina,oldpeak,slope,number_of
_major_vessels,thal,class {absent,present}
70,1,4,130,322,0,2,109,0,2.4,2,3,3,present
67,0,3,115,564,0,2,160,0,1.6,2,0,7,absent
57,1,2,124,261,0,0,141,0,0.3,1,0,7,present
64,1,4,128,263,0,0,105,1,0.2,2,1,7,absent
65,1,4,120,177,0,0,140,0,0.4,1,0,7,absent
56,1,3,130,256,1,2,142,1,0.6,2,1,6,present
60,1,4,140,293,0,2,170,0,1.2,2,2,7,present
```

- Componenti utilizzati: InputDataset, NeuralN, J48Bean.
- Risultati e commenti:

I risultati dell'analisi dei dati effettuata con il componente NeuralN sono riportati nelle seguenti righe (per semplicità si è omesso di riportare la composizione dei nodi della rete neurale):

```
Informazioni statistiche sull'elaborazione:

=== Summary ===
Correctly Classified Instances         219               81.1111 %
Incorrectly Classified Instances        51               18.8889 %
Kappa statistic                          0.6185
K&B Relative Info Score              15527.2136 %
K&B Information Score                  153.9091 bits      0.57   bits/instance
Class complexity | order 0             267.5907 bits      0.9911 bits/instance
Class complexity | scheme              393.1506 bits      1.4561 bits/instance
Complexity improvement     (Sf)       -125.5599 bits     -0.465  bits/instance
Mean absolute error                      0.2118
Root mean squared error                  0.416
Relative absolute error                 42.8848 %
Root relative squared error             83.7123 %
Total Number of Instances              270

=== Confusion Matrix ===
   a   b   <-- classified as
 123  27 |   a = absent
  24  96 |   b = present

=== Detailed Accuracy By Class ===
TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.82      0.2        0.837     0.82      0.828    absent
  0.8       0.18       0.78      0.8       0.79     present
```

I risultati dell'analisi dei dati effettuata con il componente J48Bean sono riportati in figura 6.4 e nelle successive righe:

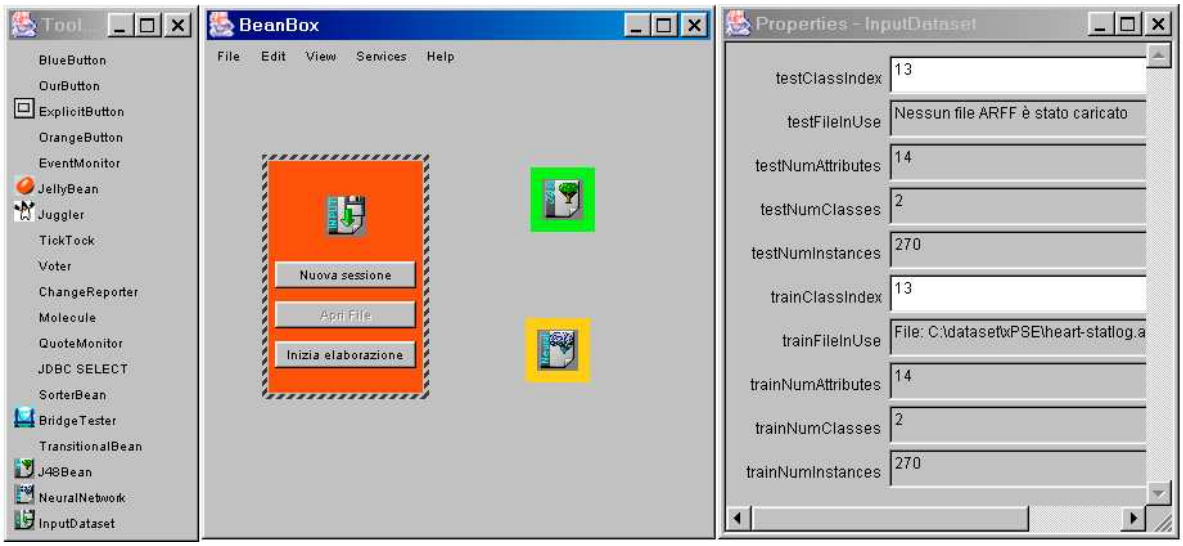


Figura 6.3 L'utilizzo dei componenti nel BeanBox

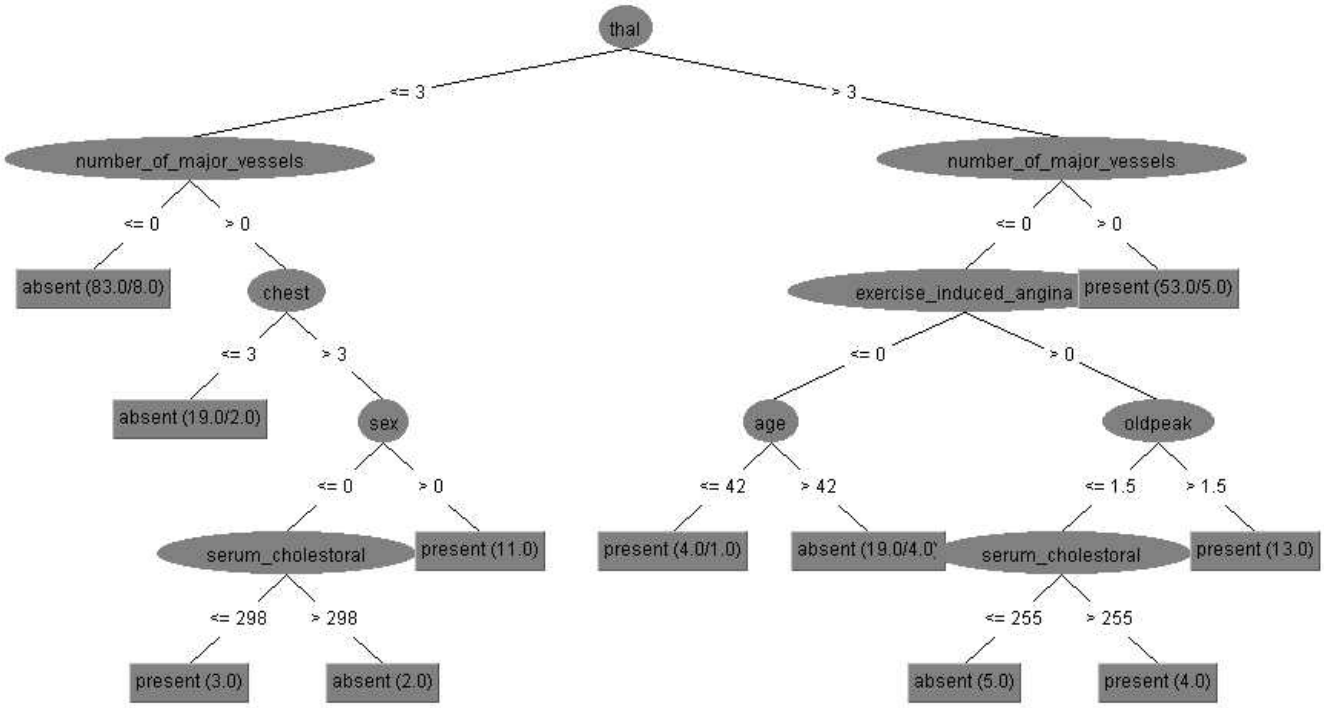


Figura 6.4 L'albero di decisione ottenuto dall'elaborazione con il J48Bean

```
Informazioni statistiche sull'elaborazione:

J48 pruned tree
------------------

thal <= 3
|   number_of_major_vessels <= 0: absent (83.0/8.0)
|   number_of_major_vessels > 0
|   |   chest <= 3: absent (19.0/2.0)
|   |   chest > 3
|   |   |   sex <= 0
|   |   |   |   serum_cholestoral <= 298: present (3.0)
|   |   |   |   serum_cholestoral > 298: absent (2.0)
|   |   |   sex > 0: present (11.0)
thal > 3
|   number_of_major_vessels <= 0
|   |   exercise_induced_angina <= 0
|   |   |   age <= 42: present (4.0/1.0)
|   |   |   age > 42: absent (19.0/4.0)
|   |   exercise_induced_angina > 0
|   |   |   oldpeak <= 1.5
|   |   |   |   serum_cholestoral <= 255: absent (5.0)
|   |   |   |   serum_cholestoral > 255: present (4.0)
|   |   |   oldpeak > 1.5: present (13.0)
|   number_of_major_vessels > 0: present (53.0/5.0)

Number of Leaves  :       11
Size of the tree : 21

=== Summary ===
Correctly Classified Instances         214               79.2593 %
Incorrectly Classified Instances        56               20.7407 %
Kappa statistic                          0.5786
K&B Relative Info Score              13833.2813 %
K&B Information Score                  137.1209 bits      0.5079 bits/instance
Class complexity | order 0            267.5907 bits      0.9911 bits/instance
Class complexity | scheme           19480.9646 bits     72.1517 bits/instance
Complexity improvement     (Sf)     -19213.3738 bits    -71.1606 bits/instance
Mean absolute error                      0.2494
Root mean squared error                  0.4284
Relative absolute error                 50.502  %
Root relative squared error             86.2047 %
Total Number of Instances              270

=== Confusion Matrix ===
   a   b   <-- classified as
 124  26 |   a = absent
  30  90 |   b = present

=== Detailed Accuracy By Class ===
TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.827     0.25       0.805     0.827     0.816    absent
  0.75      0.173      0.776     0.75      0.763    present
```

L'applicazione di questi due algoritmi evidenzia come sia molto importante, per una efficace analisi, specificare lo scopo per cui viene effettuato il data mining. Se infatti lo scopo è costruire un modello da applicare a nuovi pazienti, in modo da monitorare con più attenzione le persone risultanti a rischio, allora l'algoritmo da preferire è quello di rete neurale incluso in NeuralN, non sol-

tanto perché presenta un tasso di errore inferiore a quello del J48 (18.9 % contro 20.7 %), ma soprattutto perché il minor numero di errori si ha relativamente ai casi classificati come "absent" che in realtà sono "present" (24 contro 30), e questo è un fattore di cruciale importanza, in quanto è preferibile diagnosticare la malattia a chi non è malato (successivi controlli medici confuteranno facilmente tale errore) piuttosto che non diagnosticarla a chi invece è malato. Lo svantaggio dell'algoritmo di rete neurale è che non produce uno schema visivo del modello, quindi è impossibile dare una spiegazione dei risultati ottenuti, e inoltre per classificare nuovi casi si deve necessariamente sottoporli all'applicativo che ha generato il modello.

Se invece lo scopo è trarre informazioni chiare e comprensibili dai dati a disposizione, in modo da indirizzare la ricerca medica, allora l'algoritmo da preferire è il J48, perché il suo schema ad albero evidenzia con chiarezza quelli che potrebbero essere considerati i principali fattori di rischio. E tale conoscenza può anche tradursi in informazioni per il pubblico per stimolare la prevenzione del male.

6.3) Data mining in campo scientifico

- Dataset utilizzato: zoo
- Provenienza: CMU StatLib Datasets Archive [25]
- Autori: Richard Forsyth
- Descrizione dei dati: i dati sono tratti da un database con attributi relativi ad animali; lo scopo è predirne il tipo tra le seguenti possibili classi: mammifero, uccello, pesce, anfibio, rettile, insetto, invertebrato. Di seguito è riportato un frammento del dataset:

```
ani-
mal,hair,feathers,eggs,milk,airborne,aquatic,predator,toothed,backbone,breathes,venomo
us,fins,legs,tail,domestic,catsize,type
bear,true,false,false,true,false,false,true,true,true,true,false,false,4,false,false,t
rue,mammal
boar,true,false,false,true,false,false,true,true,true,true,false,false,4,true,false,tr
ue,mammal
buffa-
lo,true,false,false,true,false,false,false,true,true,true,false,false,4,true,false,tru
e,mammal
fruitbat,true,false,false,true,true,false,false,true,true,true,false,false,2,true,fals
e,false,mammal
giraf-
fe,true,false,false,true,false,false,false,true,true,true,false,false,4,true,false,tru
e,mammal
mo-
le,true,false,false,true,false,false,true,true,true,true,false,false,4,true,false,fals
e,mammal
mongoo-
se,true,false,false,true,false,false,true,true,true,true,false,false,4,true,false,true
,mammal
```

- Componenti utilizzati: InputDataset, J48Bean.
- Risultati e commenti:

Anche in questo caso, come nel precedente, l'algoritmo J48 deve considerarsi applicato a proposito se lo scopo è trarre informazioni chiare e comprensibili, magari per motivi didattici o educativi. Ma se lo scopo è procedere ad una classificazione automatica di nuove specie animali, allora conviene scegliere un altro algoritmo per costruire un modello più preciso, se possibile.

I risultati dell'analisi dei dati effettuata con il componente J48Bean sono riportati nelle seguenti righe e in figura 6.5:

```
Informazioni statistiche sull'elaborazione:

J48 pruned tree
------------------

feathers = false
|   milk = false
|   |   backbone = false
|   |   |   airborne = false: invertebrate (9.0/1.0)
|   |   |   airborne = true: insect (5.0)
|   |   backbone = true
|   |   |   fins = false
|   |   |   |   tail = false: amphibian (2.0)
|   |   |   |   tail = true: reptile (5.0/1.0)
|   |   |   fins = true: fish (11.0)
|   milk = true: mammal (33.0)
feathers = true: bird (16.0)

Number of Leaves  :       7
Size of the tree : 13
```

```
=== Summary ===
Correctly Classified Instances          96               95.0495 %
Incorrectly Classified Instances         5                4.9505 %
Kappa statistic                          0.9347
K&B Relative Info Score               9112.1871 %
K&B Information Score                  222.7923 bits      2.2059 bits/instance
Class complexity | order 0             242.481  bits      2.4008 bits/instance
Class complexity | scheme             3229.6634 bits     31.9769 bits/instance
Complexity improvement     (Sf)      -2987.1824 bits    -29.5761 bits/instance
Mean absolute error                      0.0159
Root mean squared error                  0.1154
Relative absolute error                  7.2523 %
Root relative squared error             35.0099 %
Total Number of Instances              101

=== Confusion Matrix ===
  a  b  c  d  e  f  g   <-- classified as
 41  0  0  0  0  0  0 |  a = mammal
  0 20  0  0  0  0  0 |  b = bird
  0  0  3  1  0  1  0 |  c = reptile
  0  0  0 13  0  0  0 |  d = fish
  0  0  0  0  4  0  0 |  e = amphibian
  0  0  0  0  0  5  3 |  f = insect
  0  0  0  0  0  0 10 |  g = invertebrate

=== Detailed Accuracy By Class ===
TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  1         0          1         1         1        mammal
  1         0          1         1         1        bird
  0.6       0          1         0.6       0.75     reptile
  1         0.011      0.929     1         0.963    fish
  1         0          1         1         1        amphibian
  0.625     0.011      0.833     0.625     0.714    insect
  1         0.033      0.769     1         0.87     invertebrate
```

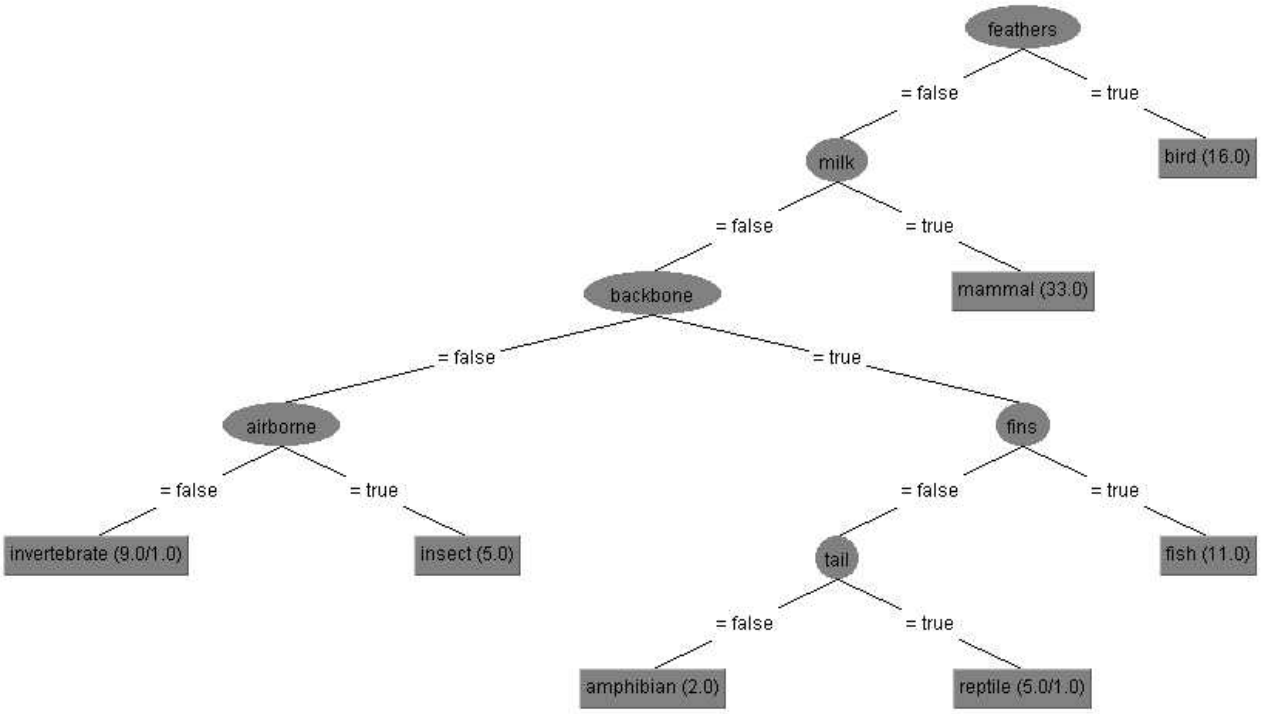


Figura 6.5 L'albero di decisione ottenuto dall'elaborazione con il J48Bean

6.4) Data mining in campo tecnico-economico

- Dataset utilizzato: elusage
- Provenienza: CMU StatLib Datasets Archive [25]
- Autori: Simonoff, J.S.
  Smoothing Methods in Statistics.
  New York: Springer-Verlag (1996). [26]
- Descrizione dei dati: i dati rappresentano un indice del consumo medio di elettricità in relazione ai mesi e alla temperatura media giornaliera. Di seguito è riportato un frammento del dataset:

```
average_temperature,month,average_electricity_usage
73,                  8,    24.828
67,                  9,    24.688
57,                 10,    19.31
43,                 11,    59.706
26,                 12,    99.667
41,                  1,    49.333
38,                  2,    59.375
```

- Componenti utilizzati: InputDataset, EMCluster.
- Risultati e commenti:

I risultati dell'analisi dei dati effettuata con il componente EMCluster sono riportati nelle seguenti righe e riassunti nel grafico di figura 6.6:

```
Informazioni statistiche sull'elaborazione:

EM
==

Number of clusters selected by cross validation: 5

Cluster: 0 Prior probability: 0.2006
Attribute: average_temperature
Normal Distribution. Mean = 44.6118 StdDev = 3.7934
Attribute: month
Discrete Estimator. Counts =  1.88 2.04 4.04 2 1 1 1 1 1 5.97 1.1  (Total = 23.03)
Attribute: average_electricity_usage
Normal Distribution. Mean = 48.6064 StdDev = 6.6241

Cluster: 1 Prior probability: 0.254
Attribute: average_temperature
Normal Distribution. Mean = 34.4241 StdDev = 4.645
Attribute: month
Discrete Estimator. Counts =  5.12 4.96 1.96 1 1 1 1 1 1 1.03 5.9  (Total = 25.97)
Attribute: average_electricity_usage
Normal Distribution. Mean = 76.5262 StdDev = 15.8349
```

```
Cluster: 2 Prior probability: 0.0545
Attribute: average_temperature
Normal Distribution. Mean = 54 StdDev = 0
Attribute: month
Discrete Estimator. Counts =  1 1 1 2 1 1 1 1 1 3 1 1  (Total = 15)
Attribute: average_electricity_usage
Normal Distribution. Mean = 46.6793 StdDev = 6.3422

Cluster: 3 Prior probability: 0.1295
Attribute: average_temperature
Normal Distribution. Mean = 58.0192 StdDev = 2.4419
Attribute: month
Discrete Estimator. Counts =  1 1 1 3 3.16 1 1 1 1.02 3.97 1 1  (Total = 19.14)
Attribute: average_electricity_usage
Normal Distribution. Mean = 25.2647 StdDev = 4.9615

Cluster: 4 Prior probability: 0.3614
Attribute: average_temperature
Normal Distribution. Mean = 71.0339 StdDev = 4.1524
Attribute: month
Discrete Estimator. Counts =  1 1 1 1 2.84 5 5 6 5.98 1.03 1 1  (Total = 31.86)
Attribute: average_electricity_usage
Normal Distribution. Mean = 22.8893 StdDev = 4.939
```

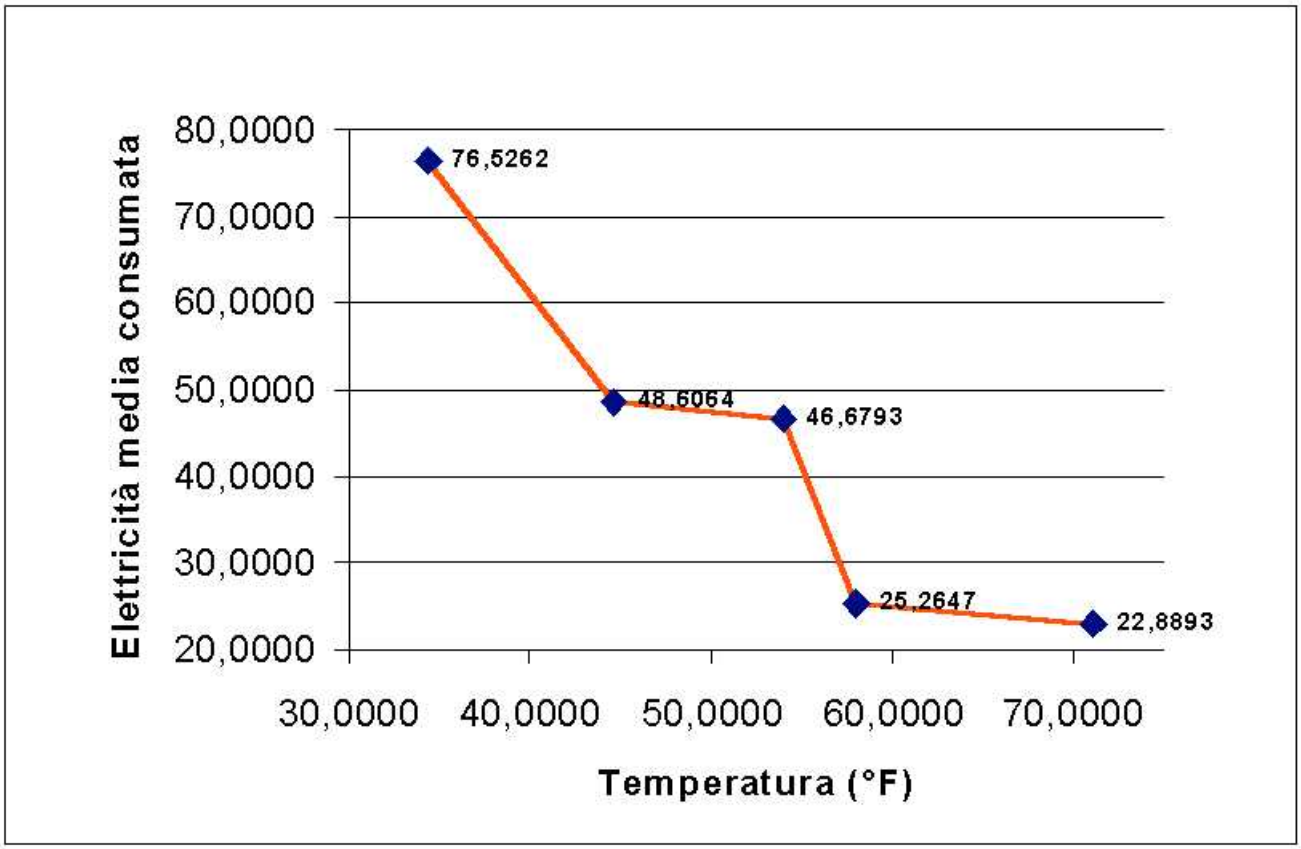


Figura 6.6 Variazione del consumo medio di elettricità in funzione della temperatura

Questo esempio mostra come l'aggregazione compiuta nel clustering può facilmente mettere in luce informazioni nascoste nei dati. Infatti dai cluster ottenuti appare chiaramente che al crescere della temperatura media si ha una diminuzione del consumo medio di elettricità, e questa informazione non si riesce a desumere così chiaramente guardando direttamente i dati. Questa analisi può essere utile sia per uno studio economico sui consumi (anche al fine di campagne preventive per la diminuzione dei consumi), sia per un eventuale utilizzo tecnico, al fine di prevedere i periodi di maggiore richiesta, e dimensionare opportunamente la produzione e l'offerta di energia elettrica.

6.5) Data mining in campo sociale

- Dataset utilizzato: vote
- Provenienza: CMU StatLib Datasets Archive [25]
- Autori: Jeff Schlimmer.
- Descrizione dei dati: i dati rappresentano le votazioni dei rappresentanti del Congresso statunitense in sedici provvedimenti chiave; lo scopo è cercare di individuare lo schieramento politico dei votanti conoscendo il voto che hanno espresso. Di seguito è riportato un frammento del dataset:

```
handicapped-infants,water-project-cost-sharing,adoption-of-the-budget-
resolution,physician-fee-freeze,el-salvador-aid,religious-groups-in-schools,anti-
satellite-test-ban,aid-to-nicaraguan-contras,mx-missile,immigration,synfuels-
corporation-cutback,education-spending,superfund-right-to-sue,crime,duty-free-
exports,export-administration-act-south-africa,Class {democrat,republican}
'y','y','y','n','n','n','y','y','y','n','n','n','n','n',?,?,'democrat'
'n','y','n','y','y','n','n','n','n','n',?,?,'y','y','n','n','republican'
'n','y','n','y','y','y','n','n','n','n','y',?,'y','y',?,?,'republican'
'n','y','y','n','n','n','y','y','y','n','n','n','y','n',?,?,'democrat'
'y','y','y','n','n','n','y','y','y',?,'y','y',?,'n','n','y',?,'democrat'
'n','y','n','y','y','y','n','n','n','n','n','y',?,?,'n',?,'republican'
'n','y','n','y','y','y','n','n','n','y','n','y','y',?,'n',?,'republican'
'y','n','y','n','n','y','n','y',?,'y','y','y',?,'n','n','y','democrat'
'y',?,'y','n','n','n','y','y','y','n','n','n','y','n','y','y','democrat'
'n','y','n','y','y','y','n','n','n','n','n',?,'y','y','n','n','republican'
```

- Componenti utilizzati: InputDataset, J48Bean.

- Risultati e commenti:

Questo esempio mostra come si possa tracciare un profilo sociale di uomini e donne sconosciuti a partire da alcune scelte da loro effettuate in settori chiave. E' una metodologia applicabile in diversi campi di indagine del data mining, come ad esempio il marketing e le scienze sociali.

I risultati dell'analisi dei dati effettuata con il componente J48Bean sono riportati nelle seguenti righe e in figura 6.7:

```
Informazioni statistiche sull'elaborazione:

J48 pruned tree
------------------

physician-fee-freeze = n: democrat (201.8/2.74)
physician-fee-freeze = y
|   synfuels-corporation-cutback = n: republican (119.01/4.03)
|   synfuels-corporation-cutback = y
|   |   mx-missile = n
|   |   |   adoption-of-the-budget-resolution = n: republican (17.33/3.3)
|   |   |   adoption-of-the-budget-resolution = y
|   |   |   |   anti-satellite-test-ban = n: democrat (5.07/0.03)
|   |   |   |   anti-satellite-test-ban = y: republican (2.2)
|   |   mx-missile = y: democrat (2.59/0.01)

Number of Leaves  :       6
Size of the tree : 11

=== Summary ===
Correctly Classified Instances         420               96.5517 %
Incorrectly Classified Instances        15                3.4483 %
Kappa statistic                          0.927
K&B Relative Info Score              38467.8016 %
K&B Information Score                  370.0518 bits      0.8507 bits/instance
Class complexity | order 0             418.628  bits      0.9624 bits/instance
Class complexity | scheme               92.1146 bits      0.2118 bits/instance
Complexity improvement     (Sf)        326.5134 bits      0.7506 bits/instance
Mean absolute error                      0.061
Root mean squared error                  0.1769
Relative absolute error                 12.8664 %
Root relative squared error             36.3236 %
Total Number of Instances              435

=== Confusion Matrix ===
   a   b   <-- classified as
 261   6 |   a = democrat
   9 159 |   b = republican

=== Detailed Accuracy By Class ===
TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.978     0.054      0.967     0.978     0.972    democrat
  0.946     0.022      0.964     0.946     0.955    republican
```

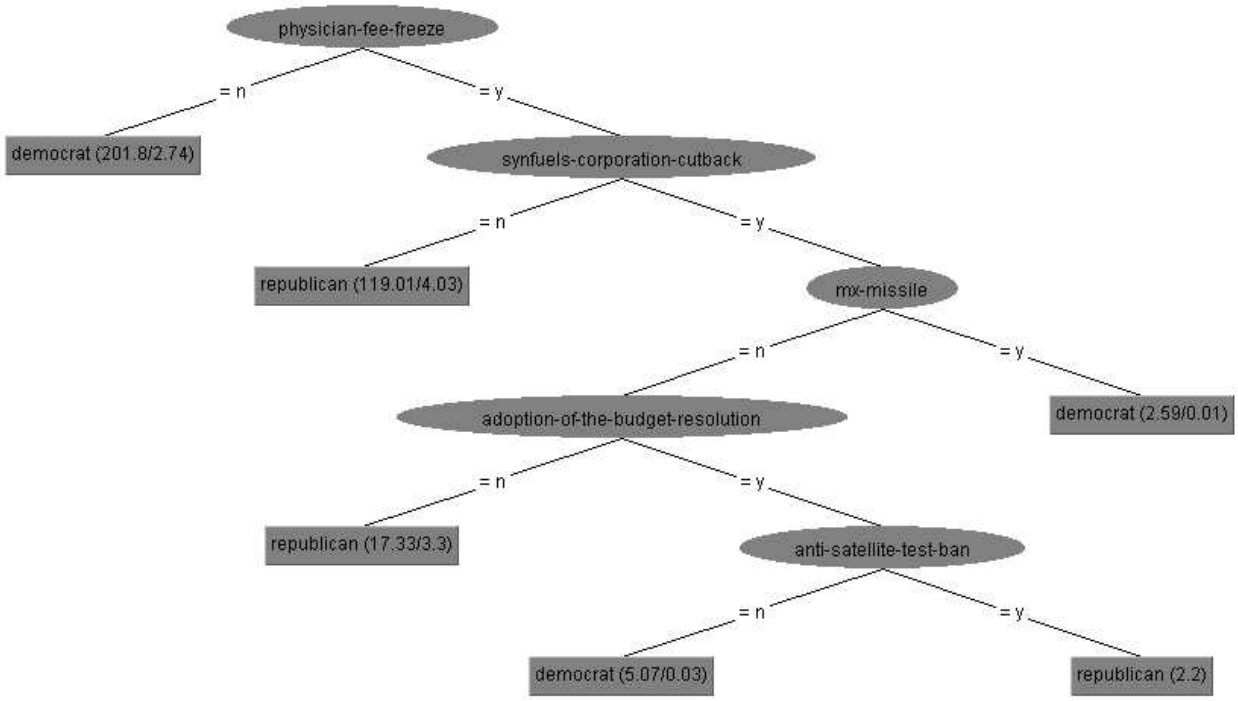


Figura 6.7 L'albero di decisione ottenuto dall'elaborazione con il J48Bean

CAPITOLO 7

CONCLUSIONI E POSSIBILI SVILUPPI

Il data mining è una disciplina impiegata su vasta scala e in continua evoluzione, ben assestata sotto l'aspetto algoritmico ed operativo, anche se in alcuni aspetti è ancora acerba, lasciando a desiderare soprattutto dal punto di vista dell'usabilità dei prodotti software, come è scaturito dall'analisi e dal confronto effettuato tra il software a pagamento (PolyAnalyst 4.5) e quello freeware (WEKA 3.2.3).

Il progetto del PSE visuale per data mining che si è realizzato dimostra come ci sia largo spazio per lo sviluppo di nuove applicazioni, che risultino migliori rispetto alle attuali soprattutto sul versante della flessibilità e della semplicità d'uso.

La differenza principale tra il PSE visuale progettato e gli altri software per data mining non sta tanto in ciò che fa, ma in che modo lo fa: con il PSE visuale l'applicativo per data mining viene discretizzato in tanti minuscoli componenti, che sfruttano appieno tutte le potenzialità e la flessibilità conferita loro dalla tecnologia *Java Beans*. Ne consegue che i componenti possono essere utilizzati tutti insieme o solo in parte; possono essere connessi in serie o in parallelo, utilizzando componenti diversi o lo stesso componente con impostazioni differenti; possono funzionare su computer distinti in modalità client-server; possono essere agevolmente inseriti in un altro applicativo Java, o in una pagina Web sotto forma di applet. Il tutto con i vantaggi di avere un piccolo e compatto componente software

indipendente e fortemente specializzato.

Nel progetto realizzato si è visto come applicando il principio del riutilizzo del software è agevole operare effettuando aggiunte e miglioramenti al numeroso software già esistente, piuttosto che partire da zero nella sua realizzazione. Mentre l'applicazione delle specifiche *Java Beans* garantisce l'ottenimento di un risultato robusto, altamente portabile, flessibile e facile da integrare. E la scelta di realizzare componenti GUI, al fine di non trascurare l'importanza delle interfacce grafiche, conferisce notevole semplicità d'uso.

Uno sviluppo ulteriore per il PSE visuale per data mining è quello di arrivare a un prodotto completo, potente ma di rapido utilizzo, che possa portare nel campo del data mining gli stessi vantaggi che si sono ottenuti in altri campi con l'introduzione di prodotti analoghi, come ad esempio nel campo dell'analisi dei sistemi con Simulink o nel campo della simulazione dei circuiti elettronici con Schematics: il primo, infatti, è un toolbox di MatLab che consente una costruzione grafica del sistema da analizzare mediante interconnessione di un'ampia gamma di blocchi funzionali predefiniti; il secondo, invece, è un toolbox di PSPICE che fornisce un foglio di lavoro grafico sul quale è possibile disegnare il circuito da analizzare avvalendosi di un'ampia gamma di simboli grafici corrispondenti ai vari componenti circuitali.

Ne consegue che è possibile percorrere numerose vie per sviluppare ulteriormente il progetto del PSE. Tra le principali si segnalano:

- sviluppare nuovi componenti che implementino algoritmi per data mining;
- sviluppare componenti che svolgano la funzionalità di filtri per la pre-elaborazione e la post-elaborazione;

- sviluppare componenti che implementino meta-schemi;
- sviluppare componenti che realizzino l'analisi esplorativa dei dati, anche in forma grafica (ad esempio con grafici a barre);
- sviluppare componenti che siano in grado di acquisire dati da altri tipi di sorgenti, come ad esempio file multimediali;
- realizzare, modificando BeanBox o Bean Builder o sviluppandolo ex novo, un ambiente di esecuzione ad hoc per il PSE visuale, in cui tra l'altro siano visibili i collegamenti tra i bean. Quest'ultima funzionalità si potrebbe ottenere anche sviluppando una apposita classe Java, che abbia il compito di realizzare e di visualizzare il collegamento tra due bean qualsiasi.

C'è da sottolineare, infine, che l'evoluzione del PSE visuale non è strettamente legata a quella di WEKA: grazie a Java e alla tecnologia *Java Beans* è possibile riusare qualsiasi classe Java, e inglobarne le funzionalità nei componenti che si intendono realizzare, per cui nel PSE è possibile integrare, insieme a quelle di WEKA, anche classi di altri software o librerie freeware, o è possibile partire da zero, realizzando anche il software di base.

A P P E N D I C E A

Il codice Java

In questa appendice viene riportato il codice sorgente Java del PSE visuale per data mining. Per semplicità si omette quello relativo alle classi BeanInfo.

Il componente InputDataset

File **InputDataset.java**:

```
package Input;

import java.beans.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.net.*;
import weka.core.*;
import weka.core.converters.*;
import weka.experiment.*;
import support.*;

/** Classe che "carica" un dataset e lo invia ad altre classi per l'elaborazione.
 *  Questo bean incapsula ed estende le funzionalità della classe weka.core.Instances */
public class InputDataset extends Panel implements java.io.Serializable {

    private String errorMessage;  // per comunicare messaggi di errore all'utente
    private boolean oneFile = true;  // si vogliono usare 1 o 2 file distinti per Test e
Train?
    private boolean soloTrain = false;  // si vuole usare solo un file per il Train?
    private boolean soloTest = false;  // si vuole usare solo un file per il Test?
    private File fileTrain;  // nome del file che si vuole usare per il train (e per il
test se oneFile è TRUE)
    private File fileTest;  // nome del file che si vuole usare per il test
    private File startPath;  // memorizza la directory di partenza
    private File actualPath;  // memorizza l'ultima directory visitata con JFileChooser
    private Instances sorgenteDatiTrain, sorgenteDatiTest;  // memorizzano i dataset
    transient private Reader readerTmp;  // è transient per consentire la serializzazione
    private Instance istanza;  // memorizza una istanza (una tupla) del dataset
    private int indexInstance = 0;
    private int positionAttr;
    private Attribute attr;
    private Vector instListeners = new Vector();
    private Vector datasetListeners = new Vector();  // vettore che gestisce la lista dei
listener dell'evento DatasetEventObject registrati presso questo bean
    private myPanel contenitoreIcona;
```

```
    private Button b1,b2,b3,b4;
    private GridBagLayout gbl;
    private GridBagConstraints gbc;
    private String oldDB, oldQuery;  // memorizzano eventuali precedenti interrogazioni
di un DB
    private String trainFileInUse;
    private String testFileInUse;
    private PropertyChangeSupport propertySupport;

    /** Creates new InputArff */
    public InputDataset() {
        ActionEventHandler handler = new ActionEventHandler();
        propertySupport = new PropertyChangeSupport( this );
        setSize( getPreferredSize() );
        gbl = new GridBagLayout();
        setLayout(gbl);
        gbc = new GridBagConstraints();
        gbc.insets = new Insets(5,5,5,5);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.weightx = 1.0;
        /* Visualiza l'icona del bean */
        Class cl = this.getClass();
        URL url = cl.getResource("inputDataset_32.GIF");
        Image img = null;
        if (url!=null) {
            img = Toolkit.getDefaultToolkit().getImage(url);
        }
        contenitoreIcona = new myPanel(img);
        /* Crea e aggiunge i Button */
        b1 = new Button("Nuova sessione");
        b2 = new Button("Apri File");
        b3 = new Button("Apri Test File");
        b4 = new Button("Inizia elaborazione");
        b1.addActionListener( handler );
        b2.addActionListener( handler );
        b3.addActionListener( handler );
        b4.addActionListener( handler );
        init();
    }

    /** Metodi per la visualizzazione del bean */
    public void paint( Graphics g ) {
        this.setBackground(Color.RED);
    }

    public Dimension getPreferredSize() {
       return new Dimension( 120, 180 );
    }

    public Dimension getMinimumSize() {
       return getPreferredSize();
    }

    public Dimension getMaximumSize() {
       return getPreferredSize();
    }

    /** Aggiunge un componente al bean utilizzando il GridBagLayout */
    private void addComponent(Component c, int x, int y, int width, int height) {
        gbc.gridx = x;
        gbc.gridy = y;
        gbc.gridheight = height; // altezza espressa in numero di righe
        gbc.gridwidth = width; // larghezza espressa in numero di colonne
        gbl.setConstraints(c, gbc);
        add(c);
    }

    /** Metodi tipici del funzionamento del bean */

    /* I seguenti metodi definiscono due proprietà bound (trainFileInUse e
     * testFileInUse), che ritornano i nomi dei file attualmente in uso, in modo che i
     * componenti registrati possano essere notificati in caso di cambiamento. Queste
```

```
     * proprietà sono read-only, cioè se ne può visualizzare il contenuto, ma non se ne
     * può modificare direttamente il valore (occorre far ricorso alla procedura guidata
     * dai pulsanti del Bean). */
    public String getTrainFileInUse() {
       if (fileTrain != null) {
           return ("File: " + fileTrain);
       }
       return "Nessun file ARFF è stato caricato";
    }

    public String getTestFileInUse() {
       if (fileTest != null) {
           return ("File: " + fileTest);
       }
       return "Nessun file ARFF è stato caricato";
    }

    /* Le seguenti due coppie di metodi definiscono la proprietà classIndex di Instances
     * per i file di Train e di Test.
     * Tale proprietà rappresenta l'indice dell'attributo del dataset che fungerà da
     * classe durante la classificazione del dataset (la classe è l'attributo in base al
     * quale avviene la classificazione delle tuple). La numerazione degli attributi
     * parte da zero, e per default WEKA utilizza l'ultimo attributo come classe.
     * Naturalmente il dataset di Train e il dataset di Test dovrebbero avere uguale
     * header, e quindi uguale classIndex. */
    public void setTrainClassIndex(int i) {
        try {
            if (sorgenteDatiTrain != null)
                sorgenteDatiTrain.setClassIndex(i);
        }
        catch (Exception e) {
             System.err.println("Error: " + e);
             System.err.println("Impossibile impostare questo classIndex sul dataset cor-
rente!");
        }
    }

    public int getTrainClassIndex() {
       if (sorgenteDatiTrain != null)
           return sorgenteDatiTrain.classIndex();
       else return 0;
    }

    public void setTestClassIndex(int i) {
        try {
            if (sorgenteDatiTest != null)
                sorgenteDatiTest.setClassIndex(i);
        }
        catch (Exception e) {
             System.err.println("Error: " + e);
             System.err.println("Impossibile impostare questo classIndex sul dataset cor-
rente!");
        }
    }

    public int getTestClassIndex() {
       if (sorgenteDatiTest != null)
           return sorgenteDatiTest.classIndex();
       else return 0;
    }

    /* Il seguente metodo definisce la proprietà (read-only) numClasses.
     * Tale proprietà rappresenta il numero di possibili "etichette" (ossia valori)
     * per la classe (attributo discriminante) del dataset. */
    public int getTrainNumClasses() {
       if (sorgenteDatiTrain != null)
          try {
             return sorgenteDatiTrain.numClasses();
          }
          catch (Exception e) {
             System.err.println("Error: " + e);
             return 0;
```

```
            }
         else return 0;
      }

      public int getTestNumClasses() {
         if (sorgenteDatiTest != null)
            try {
               return sorgenteDatiTest.numClasses();
            }
            catch (Exception e) {
               System.err.println("Error: " + e);
               return 0;
            }
         else return 0;
      }

      /* Il seguente metodo definisce la proprietà (read-only) numInstances.
       * Tale proprietà rappresenta il numero di tuple contenute nel dataset. */
      public int getTrainNumInstances() {
         if (sorgenteDatiTrain != null)
             return sorgenteDatiTrain.numInstances();
         else return 0;
      }

      public int getTestNumInstances() {
         if (sorgenteDatiTest != null)
             return sorgenteDatiTest.numInstances();
         else return 0;
      }

      /* Il seguente metodo definisce la proprietà (read-only) numAttributes.
       * Tale proprietà rappresenta il numero di attributi del dataset. */
      public int getTrainNumAttributes() {
         if (sorgenteDatiTrain != null)
             return sorgenteDatiTrain.numAttributes();
         else return 0;
      }

      public int getTestNumAttributes() {
         if (sorgenteDatiTest != null)
             return sorgenteDatiTest.numAttributes();
         else return 0;
      }

      /* Il seguente metodo definisce la proprietà (read-only) contStrAttr.
       * Tale proprietà è TRUE se il dataset contiene attributi di tipo stringa (non
       * tutti i classificatori e i filtri sono in grado di manipolare attributi di
       * tipo stringa). */
      public boolean getTrainContStrAttr() {
         if (sorgenteDatiTrain != null)
             return sorgenteDatiTrain.checkForStringAttributes();
         else return false;
      }

      public boolean getTestContStrAttr() {
         if (sorgenteDatiTest != null)
             return sorgenteDatiTest.checkForStringAttributes();
         else return false;
      }

      /** Determina l'origine del Dataset e chiama i corrispondenti metodi per il
       *  caricamento */
      public void loadDataset(boolean first)   {

               JOptionPane askInput = new JOptionPane();
               Object[] possibleValues = { "File .ARFF", "File di testo CSV" , "Query in DB
registrato in ODBC" };
               Object selectedValue = JOptionPane.showInputDialog(null, "Seleziona la sor-
gente per il Dataset", "Seleziona la sorgente per il Dataset",
                   JOptionPane.QUESTION_MESSAGE, null,possibleValues, possibleValues[0]);
               if ( selectedValue == possibleValues[0] ) {
                   if (first) loadTrainFile();
```

```
                else loadTestFile();
            }
            else if ( selectedValue == possibleValues[1] ) {
                loadCSVFile(first);
            }
            else if ( selectedValue == possibleValues[2] ) {
                loadFromDB(first);
            }
            else {  }  // E' stato scelto il pulsante ANNULLA
    }

    /** Crea un oggetto Instances inizializzandolo con un file .arff */
    public void loadTrainFile()   {
        String oldFile;

        if ( fileTrain != null )
            oldFile = fileTrain.toString();
        else
            oldFile = null;
        openFile(getParent(), true);
        trainFileInUse = fileTrain.toString();
        propertySupport.firePropertyChange("trainFileInUse",oldFile,trainFileInUse);
        if (fileTrain != null) {
           try {
              readerTmp = new FileReader(fileTrain);
              sorgenteDatiTrain = new Instances(readerTmp);
              setTrainClassIndex(sorgenteDatiTrain.numAttributes() - 1);
              b2.setEnabled(false);
              if ( oneFile ) {
                  b4.setEnabled(true);
                  if ( soloTest ) {    // Si deve caricare un solo file, ma questo è di
Test, non di Train
                      sorgenteDatiTest = sorgenteDatiTrain;
                      sorgenteDatiTrain = null;
                  }
                  else if ( !soloTrain ) {    // Si deve caricare un solo file, ma questo
serve sia per il Train che per il Test
                      sorgenteDatiTest =  sorgenteDatiTrain;
                  }
              }   // il file è stato caricato, e l'elaborazione può iniziare
              else { b3.setEnabled(true); }     // bisogna caricare il secondo file
           }
           catch (FileNotFoundException exc) {
              errorMessage = "File non trovato!.\n (loadTrainFile - 01)";
              JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOptionPa-
ne.ERROR_MESSAGE);
              System.err.println("Error: File non trovato!\n" + exc);
              exc.printStackTrace();
           }
           catch (Exception e) {
              errorMessage = "Si è verificato un errore.\n (loadTrainFile - 02)";
              JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOptionPa-
ne.ERROR_MESSAGE);
              System.err.println("Error: " + e);
              e.printStackTrace();
           }
        }
    }

    public void loadTestFile()   {
        String oldFile;

        if ( fileTest != null )
            oldFile = fileTest.toString();
        else
            oldFile = null;
        openFile(getParent(), false);
        testFileInUse = fileTest.toString();
        propertySupport.firePropertyChange("testFileInUse",oldFile,testFileInUse);
        if (fileTest != null) {
           try {
              readerTmp = new FileReader(fileTest);
```

```
                sorgenteDatiTest = new Instances(readerTmp);
                setTestClassIndex(sorgenteDatiTest.numAttributes() - 1);
               if (!(sorgenteDatiTrain.equalHeaders(sorgenteDatiTest))) {
                    errorMessage = "Train and test dataset are not compatible";
                    JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOptionPa-
ne.ERROR_MESSAGE);
                    System.err.println("Error: " + errorMessage);
                }
                else {
                    b3.setEnabled(false);
                    b4.setEnabled(true);  // i file sono stati caricati entrambi, e l'ela-
borazione può iniziare
                }
            }
            catch (FileNotFoundException exc) {
               errorMessage = "File non trovato!.\n (loadTestFile - 01)";
               JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOptionPa-
ne.ERROR_MESSAGE);
               System.err.println("Error: File non trovato!\n" + exc);
               exc.printStackTrace();
            }
            catch (Exception e) {
               errorMessage = "Si è verificato un errore.\n (loadTestFile - 02)";
               JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOptionPa-
ne.ERROR_MESSAGE);
               System.err.println("Error: " + e);
               e.printStackTrace();
            }
        }
    }

    /** Visualizza una finestra di dialogo per l'apertura di un file .arff */
    public void openFile(Component parent, boolean firstFile)   {
       File fileName;

       if (startPath == null) {
           startPath = new File(".");
           actualPath = startPath;
       }
       JFileChooser chooser = new JFileChooser(actualPath.getAbsolutePath()); //apre la
finestra iniziando dalla directory corrente
       ArffFileFilter filter = new ArffFileFilter();
       chooser.addChoosableFileFilter(filter);
       chooser.setDialogType(JFileChooser.OPEN_DIALOG);
       chooser.setFileSelectionMode(JFileChooser.FILES_ONLY );
       int returnVal = chooser.showOpenDialog(parent);
       // user clicked Cancel button on dialog
       if ( returnVal == JFileChooser.CANCEL_OPTION ) {
            System.err.println("You chose CANCEL option");
       }
       else
          if(returnVal == JFileChooser.APPROVE_OPTION) {
             fileName = chooser.getSelectedFile();
             actualPath = fileName.getAbsoluteFile();
             System.err.println("You chose to open this file: " + choo-
ser.getSelectedFile().getName());
             if (firstFile) { fileTrain = fileName; }
             else { fileTest = fileName; }
          }
    }

    /** Crea un oggetto Instances inizializzandolo con un file di testo
      * in cui i campi delle tuple sono delimitati con una virgola o con
      * un TAB. Presuppone che la prima riga contenga i nomi degli attributi,
      * e le successive le tuple con i propri valori. */
    public void loadCSVFile( boolean firstFile )   {
       String oldFile;
       File fileName;
       CSVLoader csv = new CSVLoader();

       if (startPath == null) {
           startPath = new File(".");
```

```
            actualPath = startPath;
        }
        JFileChooser chooser = new JFileChooser(actualPath.getAbsolutePath()); //apre la
finestra iniziando dalla directory corrente
        chooser.setDialogType(JFileChooser.OPEN_DIALOG);
        chooser.setFileSelectionMode(JFileChooser.FILES_ONLY );
        int returnVal = chooser.showOpenDialog(null);
        // user clicked Cancel button on dialog
        if ( returnVal == JFileChooser.CANCEL_OPTION ) {
             System.err.println("Open Dialog - You chose CANCEL option");
        }
        else if(returnVal == JFileChooser.APPROVE_OPTION) {
            fileName = chooser.getSelectedFile();
            actualPath = fileName.getAbsoluteFile();
            System.err.println("You chose to open this file: " + choo-
ser.getSelectedFile().getName());
            if (firstFile) {
                if ( fileTrain != null )
                    oldFile = fileTrain.toString();
                else
                    oldFile = null;
                fileTrain = fileName;
                trainFileInUse = fileTrain.toString();
                propertySup-
port.firePropertyChange("trainFileInUse",oldFile,trainFileInUse);
                if (fileTrain != null) {
                   try {
                      csv.reset();
                      csv.setSource(fileTrain);
                      sorgenteDatiTrain = csv.getDataSet();
                      setTrainClassIndex(sorgenteDatiTrain.numAttributes() - 1);
                      b2.setEnabled(false);
                      if ( oneFile ) {
                          b4.setEnabled(true);
                          if ( soloTest ) {    // Si deve caricare un solo file, ma questo
è di Test, non di Train
                              sorgenteDatiTest = sorgenteDatiTrain;
                              sorgenteDatiTrain = null;
                          }
                          else if ( !soloTrain ) {    // Si deve caricare un solo file, ma
questo serve sia per il Train che per il Test
                              sorgenteDatiTest =  sorgenteDatiTrain;
                          }
                      }   // il file è stato caricato, e l'elaborazione può iniziare
                      else { b3.setEnabled(true); }    // bisogna caricare il secondo file
                   }
                   catch (FileNotFoundException exc) {
                      System.err.println("Error: File non trovato!\n" + exc);
                      exc.printStackTrace();
                   }
                   catch (Exception e) {
                      errorMessage = "Si è verificato un errore.\n (loadCSVFile - 01)";
                      JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                      System.err.println("Error: " + e);
                      e.printStackTrace();
                   }
                }
            }
            else {
                if ( fileTest != null )
                    oldFile = fileTest.toString();
                else
                    oldFile = null;
                fileTest = fileName;
                testFileInUse = fileTest.toString();
                propertySupport.firePropertyChange("testFileInUse",oldFile,testFileInUse);
                if (fileTest != null) {
                   try {
                      csv.reset();
                      csv.setSource(fileTest);
                      sorgenteDatiTest = csv.getDataSet();
```

```
                    setTestClassIndex(sorgenteDatiTest.numAttributes() - 1);
                    b3.setEnabled(false);
                    b4.setEnabled(true);  // i file sono stati caricati entrambi, e
l'elaborazione può iniziare
                 }
                 catch (FileNotFoundException exc) {
                    System.err.println("Error: File non trovato!\n" + exc);
                    exc.printStackTrace();
                 }
                 catch (Exception e) {
                    errorMessage = "Si è verificato un errore.\n (loadCSVFile - 02)";
                    JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                    System.err.println("Error: " + e);
                    e.printStackTrace();
                 }
              }
           }
       }


    }

    /** Crea un oggetto Instances inizializzandolo con il risultato di una
      * query su un DB. Viene usato il driver JDBC-ODBC, quindi per effettuare
      * la query è necessario registrare il DB in ODBC; inoltre bisogna riportare
      * il tale nome di registrazione in un apposito file (), che deve trovarsi
      * nella directory di partenza del programma. */
    public void loadFromDB( boolean firstFile )   {
       String oldFile;
       InstanceQuery iq;
       AskForQuery aq;

       if (startPath == null) {
           startPath = new File(".");
           actualPath = startPath;
       }
       /* Richiesta della query all'utente */
       aq = new AskForQuery(oldDB, oldQuery);
       aq.show();
       if ( aq.isOK() ) {
           try {
               oldDB = aq.getDB();
               oldQuery = aq.getQuery();
               System.err.println("\nDB e Query: \n" + aq.getDB() + "\n" + aq.getQuery()
+ "\n");
               iq = new InstanceQuery();
               iq.setDatabaseURL( aq.getDB() );
               iq.setQuery( aq.getQuery() );
               iq.connectToDatabase();
               if (firstFile) {     // caricamento del dataset Train
                   if ( iq.isConnected() ) {
                       if ( fileTrain != null )
                           oldFile = fileTrain.toString();
                       else
                           oldFile = null;
                       trainFileInUse = iq.getDatabaseURL();
                       propertySup-
port.firePropertyChange("trainFileInUse",oldFile,trainFileInUse);
                       try {
                          sorgenteDatiTrain = iq.retrieveInstances();
                          setTrainClassIndex(sorgenteDatiTrain.numAttributes() - 1);
                          b2.setEnabled(false);
                          if ( oneFile ) {
                              b4.setEnabled(true);
                              if ( soloTest ) {    // Si deve caricare un solo file, ma
questo è di Test, non di Train
                                  sorgenteDatiTest = sorgenteDatiTrain;
                                  sorgenteDatiTrain = null;
                              }
                              else if ( !soloTrain ) {    // Si deve caricare un solo fi-
le, ma questo serve sia per il Train che per il Test
```

```
                                        sorgenteDatiTest =  sorgenteDatiTrain;
                                    }
                                }   // il file è stato caricato, e l'elaborazione può iniziare
                                else { b3.setEnabled(true); }     // bisogna caricare il secon-
do file
                            }
                            catch (Exception e) {
                                errorMessage = "Si è verificato un errore.\n (loadFromDB -
01)";
                                JOptionPane.showMessageDialog(null, errorMessage, "Error!",
JOptionPane.ERROR_MESSAGE);
                                System.err.println("Error: " + e);
                                e.printStackTrace();
                            }
                            finally {
                                iq.disconnectFromDatabase();
                            }
                        }
                    }
                    else {     // caricamento del dataset Test
                        if ( iq.isConnected() ) {
                            if ( fileTest != null )
                                oldFile = fileTest.toString();
                            else
                                oldFile = null;
                            testFileInUse = iq.getDatabaseURL();
                            propertySup-
port.firePropertyChange("testFileInUse",oldFile,testFileInUse);
                            try {
                                sorgenteDatiTest = iq.retrieveInstances();
                                setTestClassIndex(sorgenteDatiTest.numAttributes() - 1);
                                b3.setEnabled(false);
                                b4.setEnabled(true);  // i file sono stati caricati entrambi,
e l'elaborazione può iniziare
                            }
                            catch (Exception e) {
                                errorMessage = "Si è verificato un errore.\n (loadFromDB -
02)";
                                JOptionPane.showMessageDialog(null, errorMessage, "Error!",
JOptionPane.ERROR_MESSAGE);
                                System.err.println("Error: " + e);
                                e.printStackTrace();
                            }
                            finally {
                                iq.disconnectFromDatabase();
                            }
                        }
                    }
                }
                catch (Exception e) {
                    System.err.println("Error: " + e);
                    e.printStackTrace();
                }
            }

        }

        public void init()   {
            fileTrain = null;
            fileTest = null;
            trainFileInUse = null;
            testFileInUse = null;
            addComponent(contenitoreIcona, 0, 0, 1, 2);
            addComponent(b1, 0, 2, 1, 1);
            validate();
        }

        public void oneFileLayout()   {
            oneFile = true;
            init();
            b2.setLabel("Apri File");
            b2.setEnabled(true);
```

```
        b4.setEnabled(false);
        addComponent(b2, 0, 3, 1, 1);
        addComponent(b4, 0, 4, 1, 1);
        validate();
    }

    public void twoFileLayout()   {
        oneFile = false;
        init();
        b2.setLabel("Apri Train File");
        b2.setEnabled(true);
        b3.setEnabled(false);
        b4.setEnabled(false);
        addComponent(b2, 0, 3, 1, 1);
        addComponent(b3, 0, 4, 1, 1);
        addComponent(b4, 0, 5, 1, 1);
        validate();
    }

    /** Metodi per la gestione degli eventi propri del bean */
    public void addDatasetListener(DatasetListener listener) {
        datasetListeners.addElement(listener);
    }

    public void removeDatasetListener(DatasetListener listener) {
        datasetListeners.removeElement(listener);
    }

    /** Il seguente metodo ritorna l'evento DatasetEventObject lanciato da questo Bean.
      * Tale metodo è necessario per il funzionamento del bean in BeanBuilder: in tale
      * ambiente sostituisce il metodo startDatasetStream(). */
    public DatasetEventObject obtainDatasetStream() {
        DatasetEventObject dso;
        if ( soloTrain ) { dso = new DatasetEventObject(this, sorgenteDatiTrain, null); }
        else if ( soloTest ) { dso = new DatasetEventObject(this, null, sorgenteDati-
Test); }
        else { dso = new DatasetEventObject(this, sorgenteDatiTrain, sorgenteDatiTest); }
        return dso;
    }

    /** Il seguente metodo invia ai bean registrati i dataset di Train e di Test
      * utilizzando un evento DatasetEventObject. */
    public void startDatasetStream() {
        DatasetEventObject dso;

        System.out.println("InputArff - Inizio metodo startDatasetStream");
        if ( soloTrain ) { dso = new DatasetEventObject(this, sorgenteDatiTrain, null); }
        else if ( soloTest ) { dso = new DatasetEventObject(this, null, sorgenteDati-
Test); }
        else { dso = new DatasetEventObject(this, sorgenteDatiTrain, sorgenteDatiTest); }
        for (int i = 0; i < datasetListeners.size(); i++) {
            System.out.println("InputArff - Inizio passo " + i + " del ciclo di startDa-
tasetStream");
            DatasetListener dl = (DatasetListener) datasetListeners.elementAt(i);
            dl.dataset_EventPerformed(dso);
            System.out.println("InputArff - Fine passo " + i + " del ciclo di startData-
setStream");
        }
        System.out.println("InputArff - Fine metodo startDatasetStream");
    }

    /** Metodi forniti per default al bean da NetBeans */
    public void addPropertyChangeListener(PropertyChangeListener listener) {
        propertySupport.addPropertyChangeListener(listener);
    }

    public void removePropertyChangeListener(PropertyChangeListener listener) {
        propertySupport.removePropertyChangeListener(listener);
    }

    private class myPanel extends Panel {
```

```
        private ImageIcon iconaBean;

        public myPanel (Image img) {
            iconaBean = new ImageIcon(img);
            setSize( getPreferredSize() );
        }

        public void paint( Graphics g ) {
            iconaBean.paintIcon(this,g,(this.getWidth()-32)/2,(this.getHeight()-32)/2);
        }
        public Dimension getPreferredSize() {
           return new Dimension( 50, 50 );
        }

        public Dimension getMinimumSize() {
           return getPreferredSize();
        }
    }

    private class ActionEventHandler implements ActionListener, java.io.Serializable  {
      private JOptionPane oneOrTwo;

      public void actionPerformed( ActionEvent e )
      {
         if ( e.getSource() == b1 ) {
             oneOrTwo = new JOptionPane();
             Object[] possibleValues = { "Un unico file per il Train e il Test", "Due fi-
le distinti per il Train e per il Test",
                                         "Solo file di Train", "solo file di Test"};
             Object selectedValue = JOptionPane.showInputDialog(null, "Quanti file inten-
di usare?", "Modalità di Input",
                 JOptionPane.QUESTION_MESSAGE, null,possibleValues, possibleValues[0]);
             if ( selectedValue == possibleValues[0] ) {
                 soloTrain = false;
                 soloTest = false;
                 removeAll();
                 oneFileLayout();
             }
             else if ( selectedValue == possibleValues[1] ) {
                 soloTrain = false;
                 soloTest = false;
                 removeAll();
                 twoFileLayout();
             }
             else if ( selectedValue == possibleValues[2] ) {
                 soloTrain = true;
                 soloTest = false;
                 removeAll();
                 oneFileLayout();
             }
             else if ( selectedValue == possibleValues[3] ) {
                 soloTrain = false;
                 soloTest = true;
                 removeAll();
                 oneFileLayout();
             }
             else {  }  // E' stato scelto il pulsante ANNULLA
         }
         else if ( e.getSource() == b2 ) {
             loadDataset(true);
         }
         else if ( e.getSource() == b3 ) {
             loadDataset(false);
         }
         else if ( e.getSource() == b4 ) {
             startDatasetStream();
         }
      }
   }
}
```

File **ArffFileFilter.java**:

```
/** Una implementazione della classe FileFilter (specifica per filtrare i file .ARFF)
*   Sviluppata sull'esempio del file ExampleFileFilter.java (demo di SUN)

* Esempio d'uso:
*
*     JFileChooser chooser = new JFileChooser();
*     ArffFileFilter filter = new ArffFileFilter();
*     chooser.addChoosableFileFilter(filter);
*     chooser.showOpenDialog(this);
*
*/

package Input;

import java.io.File;
import java.util.Hashtable;
import java.util.Enumeration;
import javax.swing.*;
import javax.swing.filechooser.*;

public class ArffFileFilter extends FileFilter {

    private static String TYPE_UNKNOWN = "Type Unknown";
    private static String HIDDEN_FILE = "Hidden File";

    private Hashtable filters = null;
    private String extension = null;
    private String description = null;
    private String fullDescription = null;

    //Costruttore
    public ArffFileFilter() {
       extension = "arff";
       description = "Arff Weka File";
       addExtension(extension);
       setDescription(description);
    }

    /* Metodo obbligatorio da implementare per le classi che estendono la classe
     * FileFilter. Determina se un file soddisfa o meno i requisiti imposti dal filtro
     * (ossia se il file può essere mostrato nel pannello o no) */
    public boolean accept(File f) {
       if(f != null) {
          if(f.isDirectory()) {
             return true;
          }
          String extension = getExtension(f);
          if(extension != null && filters.get(getExtension(f)) != null) {
             return true;
          };
      }
      return false;
    }

     /* Ritorna la parte di un file contenente la sua estensione
      * (questo metodo è usato dal metodo accept) */
     public String getExtension(File f) {
        if(f != null) {
           String filename = f.getName();
           int i = filename.lastIndexOf('.');
           if(i>0 && i<filename.length()-1) {
              return filename.substring(i+1).toLowerCase();
           };
        }
        return null;
```

```
     }

    // Aggiunge una voce all'elenco delle estensioni di file che verranno filtrate
    public void addExtension(String extension) {
       if(filters == null) {
          filters = new Hashtable(5);
       }
       filters.put(extension.toLowerCase(), this);
       fullDescription = null;
    }


    /* Metodo obbligatorio da implementare per le classi che estendono la classe
     * FileFilter. Ritorna una stringa che descrive il tipo di file che si sta
     * selezionando. */
    public String getDescription() {
       if(fullDescription == null) {
          Enumeration extensions = filters.keys();
          fullDescription = description==null ? "(" : description + " (";

          // il seguente if tratta il caso generico (considera anche
          // la possibilità che extensions sia null o abbia più di un elemento)
          if(extensions != null) {
             fullDescription += "." + (String) extensions.nextElement();
             while (extensions.hasMoreElements()) {
                fullDescription += ", ." + (String) extensions.nextElement();
             }
          }

          fullDescription += ")";
       }
       return fullDescription;
    }

    // Imposta la descrizione del file che si intende filtrare
    public void setDescription(String descr) {
       this.description = descr;
       fullDescription = null;
    }
}
```

File **AskForQuery.java**:

```
package Input;

/** Questa classe visualizza una finestra di dialogo che appare quando si devono caricare
 *  i dati da un database. Essa consente all'utente di inserire l'indirizzo del database
 *  registrato come origine dati ODBC e la query SQL da sottoporre al database. */
public class AskForQuery extends javax.swing.JDialog {

    boolean pressedOK = false;

    /** Creates new form AskForQuery */
    public AskForQuery(String oldDB, String oldQuery) {
        super();
        initComponents();
        if ( oldDB != null ) {
           jTextField1.setText(oldDB);
        }
        if ( oldQuery != null ) {
           jTextField2.setText(oldQuery);
        }
    }

    /** This method is called from within the constructor to initialize the form. */
    private void initComponents() {//GEN-BEGIN:initComponents
        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
```

```
        jTextField1 = new javax.swing.JTextField();
        jLabel2 = new javax.swing.JLabel();
        jTextField2 = new javax.swing.JTextField();
        jButton1 = new javax.swing.JButton();

        setTitle("Open Data Base");
        setModal(true);
        setResizable(false);
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                closeDialog(evt);
            }
        });

        jPanel1.setLayout(null);

        jPanel1.setMinimumSize(new java.awt.Dimension(600, 260));
        jPanel1.setPreferredSize(new java.awt.Dimension(600, 260));
        jLabel1.setFont(new java.awt.Font("Dialog", 1, 14));
        jLabel1.setText("Data Base URL");
        jPanel1.add(jLabel1);
        jLabel1.setBounds(50, 40, 102, 19);

        jTextField1.setFont(new java.awt.Font("Dialog", 0, 14));
        jTextField1.setText("jdbc:odbc:?");
        jTextField1.setToolTipText("Inserire la locazione del DB");
        jTextField1.setMargin(new java.awt.Insets(2, 2, 2, 2));
        jPanel1.add(jTextField1);
        jTextField1.setBounds(48, 65, 500, 27);

        jLabel2.setFont(new java.awt.Font("Dialog", 1, 14));
        jLabel2.setText("Query");
        jPanel1.add(jLabel2);
        jLabel2.setBounds(50, 130, 42, 19);

        jTextField2.setFont(new java.awt.Font("Dialog", 0, 14));
        jTextField2.setText("SELECT * FROM ?");
        jTextField2.setToolTipText("Inserire la query che si vuole eseguire sul DB");
        jTextField2.setMargin(new java.awt.Insets(2, 2, 2, 2));
        jPanel1.add(jTextField2);
        jTextField2.setBounds(48, 150, 500, 27);

        jButton1.setFont(new java.awt.Font("Dialog", 1, 14));
        jButton1.setText("OK");
        jButton1.setBorder(new ja-
vax.swing.border.BevelBorder(javax.swing.border.BevelBorder.RAISED));
        jButton1.setFocusPainted(false);
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                handButt1(evt);
            }
        });

        jPanel1.add(jButton1);
        jButton1.setBounds(250, 210, 100, 30);

        getContentPane().add(jPanel1, java.awt.BorderLayout.CENTER);

        pack();
    }//GEN-END:initComponents

    private void handButt1(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_handButt1
        pressedOK = true;
        setVisible(false);
        dispose();
    }//GEN-LAST:event_handButt1

    /** Closes the dialog */
    private void closeDialog(java.awt.event.WindowEvent evt) {//GEN-
FIRST:event_closeDialog
        setVisible(false);
        dispose();
```

```
    }//GEN-LAST:event_closeDialog

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        new AskForQuery(null, null).show();
    }

    /** Ritorna nome e locazione del DB */
    public String getDB() {
        return jTextField1.getText();
    }

    /** Ritorna la query inserita dall'utente */
    public String getQuery() {
        return jTextField2.getText();
    }

    /** Ritorna TRUE se è stato premuto il tasto OK */
    public boolean isOK() {
        return pressedOK;
    }


    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JButton jButton1;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField1;
    // End of variables declaration//GEN-END:variables
}
```

Il componente J48Bean

File **J48Bean.java**:

```
package classificatori.j48;

import java.beans.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.net.*;
import weka.core.*;
import weka.classifiers.j48.*;
import weka.classifiers.*;
import weka.classifiers.evaluation.*;
import weka.gui.treevisualizer.*;
import support.*;

/** Classe che elabora un dataset secondo l'algoritmo C4.5, ricavandone un albero di
 *  decisione (in formato visuale e stringa). Questo Bean incapsula ed estende le
 *  funzionalità della classe weka.classifiers.j48.J48 */
public class J48Bean extends Canvas implements java.io.Serializable {

    private ImageIcon iconaBean;
    private Instances trainDataset;  // dataset da elaborare (per costruire il classifi-
```

```
catore)
    private Instances testDataset;  // dataset da elaborare (per testare il classificato-
re)
    private J48 classif; // oggetto classificatore (contiene l'algoritmo per elaborare il
dataset)
    transient private Evaluation eva; // oggetto che utilizza il classificatore per clas-
sificare dataset
    private String errorMessage;  // per comunicare messaggi di errore all'utente

    /** Creates new j48Bean */
    public J48Bean() {
        setSize( getPreferredSize() );
        repaint();
    }

    /** Metodi per la visualizzazione del Bean */
    public void paint( Graphics g ) {
        this.setBackground(Color.GREEN);
        Class cl = this.getClass();
        URL url = cl.getResource("j48_32.GIF");
        if (url!=null) {
            Image img = Toolkit.getDefaultToolkit().getImage(url);
            iconaBean = new ImageIcon(img);
            iconaBean.paintIcon(this,g,(this.getWidth()-32)/2,(this.getHeight()-32)/2);
        }
    }

    public Dimension getPreferredSize() {
       return new Dimension( 50, 50 );
    }

    public Dimension getMinimumSize() {
       return getPreferredSize();
    }

    public Dimension getMaximumSize() {
       return getPreferredSize();
    }

    /** Metodi tipici del funzionamento del Bean */

    /* La seguente coppia di metodi definisce la proprietà unpruned della classe J48 */
    public void setUnpruned(boolean u) {
        classif.setUnpruned(u);
    }

    public boolean getUnpruned() {
       if (classif != null)
           return classif.getUnpruned();
       else return false;
    }

    /* La seguente coppia di metodi definisce la proprietà reducedErrorPruning */
    public void setReducedErrorPruning(boolean u) {
        classif.setReducedErrorPruning(u);
    }

    public boolean getReducedErrorPruning() {
       if (classif != null)
           return classif.getReducedErrorPruning();
       else return false;
    }

    /* La seguente coppia di metodi definisce la proprietà binarySplits */
    public void setBinarySplits(boolean u) {
        classif.setBinarySplits(u);
    }

    public boolean getBinarySplits() {
       if (classif != null)
           return classif.getBinarySplits();
       else return false;
```

```
    }

    /* La seguente coppia di metodi definisce la proprietà subtreeRaising */
    public void setSubtreeRaising(boolean u) {
        classif.setSubtreeRaising(u);
    }

    public boolean getSubtreeRaising() {
       if (classif != null)
           return classif.getSubtreeRaising();
       else return true;
    }

    /* La seguente coppia di metodi definisce la proprietà numFolds della classe J48,
     * che definisce il numero di "folds" nel caso di "reduced error pruning" */
    public void setNumFolds(int n) {
        classif.setNumFolds(n);
    }

    public int getNumFolds() {
       if (classif != null)
           return classif.getNumFolds();
       else return 3;
    }

    /* La seguente coppia di metodi definisce la proprietà minNumObj della classe J48,
     * che rappresenta il minimo numero di Instance per ciascuna foglia dell'albero */
    public void setMinNumObj(int n) {
        classif.setMinNumObj(n);
    }

    public int getMinNumObj() {
       if (classif != null)
           return classif.getMinNumObj();
       else return 2;
    }

    /* La seguente coppia di metodi definisce la proprietà confidenceFactor (CF) */
    public void setConfidenceFactor(float v) {
        classif.setConfidenceFactor(v);
    }

    public float getConfidenceFactor() {
       if (classif != null)
           return classif.getConfidenceFactor();
       else return 0.25f;
    }

    /* Metodi per la gestione degli eventi propri del Bean */
    /** Metodo che elabora uno o due dataset giunti tramite un DatasetEventObject */
    public void dataset_EventPerformed(DatasetEventObject dso) {
        String grph;

        System.out.println("J48 - Inizio metodo dataset_EventPerformed");
        try {
            if (dso.getTrainDataset() != null) {  // conserva il vecchio dataset nel caso
in cui si riceve solo i dataset di Test
                trainDataset = dso.getTrainDataset();
            }
            testDataset = dso.getTestDataset();
            System.out.println("\nJ48 - Train dataset ricevuto tramite apposito evento:
\n\n" + trainDataset + "\n\n");
            System.out.println("\nJ48 - Test dataset ricevuto tramite apposito evento:
\n\n" + testDataset + "\n\n");
            if (testDataset == null) {
                /* Solo dataset di Train */
                try {
                    classif = null;  // azzera un precedente classificatore, contrasse-
gnandolo per il garbage collector
                    classif = new J48();
                    classif.buildClassifier(trainDataset);
                    eva = new Evaluation(trainDataset);
```

```
                    eva.evaluateModel(classif, trainDataset);
                    grph = ((Drawable)classif).graph();
                    visualizeTree2(grph,"J48 Tree Mode 2");
                }
                catch (Exception e) {
                    errorMessage = "Si è verificato un errore!\n(dataset_EventPerformed -
01)";
                    JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                    System.err.println("J48 - Error: " + e);
                    e.printStackTrace();
                }
            }
            else if (trainDataset == null) {
                /* Solo dataset di Test */
                try {
                    if ( classif.toString() != "No classifier built") {
                        eva = new Evaluation(trainDataset);
                        eva.evaluateModel(classif, testDataset);
                        grph = ((Drawable)classif).graph();
                        visualizeTree2(grph,"J48 Tree Mode 2");
                    }
                    else {
                        errorMessage = "Prima è necessario costruire un classificatore\n
utilizzando un dataset di Train!";
                        JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                    }
                }
                catch (Exception e) {
                    errorMessage = "Si è verificato un errore!\n(dataset_EventPerformed -
02)";
                    JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                    System.err.println("J48 - Error: " + e);
                    e.printStackTrace();
                }
            }
            else if (trainDataset == testDataset) {
                /* Un unico dataset per Train e per Test */
                JOptionPane askMethod = new JOptionPane();
                Object[] possibleValues = { "Cross Validation", "Percentage Split" };
                Object selectedValue = JOptionPane.showInputDialog(null,  "Come creare i
dataset per Train e Test?", "Train & Test",
                    JOptionPane.QUESTION_MESSAGE, null,possibleValues, possibleVa-
lues[0]);
                if ( selectedValue == possibleValues[0] ) { // Cross Validation
                    String folds = JOptionPane.showInputDialog("Number of Folds in Cross
Validation\n (must be greater than 1)");
                    int numFolds;
                    try {
                        numFolds = Integer.parseInt(folds);
                    }
                    catch (Exception e) {           // l'input non è corretto
                        errorMessage = "Invalid value!\n I'm setting Folds to 2";
                        JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                        System.err.println("Error: " + e);
                        e.printStackTrace();
                        numFolds = 2;
                    }
                    classif = null;  // azzera un precedente classificatore, contrasse-
gnandolo per il garbage collector
                    classif = new J48();
                    trainDataset.randomize(new Random());
                    eva = new Evaluation(trainDataset);
                    if ( numFolds > 1 && numFolds <= trainDataset.numInstances()) {
                        eva.crossValidateModel(classif, trainDataset, numFolds);
                    }
                    else {
                        errorMessage = "Invalid value!\n Number of Folds must be > 1\n" +
                            "and <= of the number of instances (" + trainData-
```

```
set.numInstances() +
                        ")\n I'm setting Folds to 2";
                    JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                    eva.crossValidateModel(classif, trainDataset, 2);
                }
                System.out.println("\nGraph: " + classif.graph());
                grph = ((Drawable)classif).graph();
                visualizeTree2(grph,"J48 Tree Mode 2");
                System.out.println("\nPrefix: " + classif.prefix() + "\n\n");
            }
            else if ( selectedValue == possibleValues[1] ) {   // Percentage Split
                String perc = JOptionPane.showInputDialog("Insert percentage split of
Train\n ( must be >0 and <100 )");
                int percentage;
                try {
                    percentage = Integer.parseInt(perc);
                }
                catch (Exception e) {            // l'input non è corretto
                    errorMessage = "Invalid value!\n I'm setting percentage to 50";
                    JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                    System.err.println("Error: " + e);
                    e.printStackTrace();
                    percentage = 50;
                }
                if ( percentage > 0 && percentage < 100) {
                    classif = null;  // azzera un precedente classificatore, contras-
segnandolo per il garbage collector
                    classif = new J48();
                    Instances provv = trainDataset;
                    provv.randomize(new Random());
                    int trainSize = provv.numInstances() * percentage / 100;
                    int testSize = provv.numInstances() - trainSize;
                    testDataset = new Instances(provv, trainSize, testSize);
                    trainDataset = new Instances(provv, 0, trainSize);
                    classif.buildClassifier(trainDataset);
                    eva = new Evaluation(trainDataset);
                    eva.evaluateModel(classif, trainDataset);
                    eva.evaluateModel(classif, testDataset);
                    grph = ((Drawable)classif).graph();
                    visualizeTree2(grph,"J48 Tree Mode 2");
                }
                else {
                    errorMessage = "Invalid value!\n Percentage must be >0 and
<100\n";
                    JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                }
            }
            else {  }  // E' stato scelto il pulsante ANNULLA
        }
        else  {
            /* Due distinti dataset per Train e per Test */
            try {
                classif = null;  // azzera un precedente classificatore, contrasse-
gnandolo per il garbage collector
                classif = new J48();
                classif.buildClassifier(trainDataset);
                eva = new Evaluation(trainDataset);
                eva.evaluateModel(classif, trainDataset);
                eva.evaluateModel(classif, testDataset);
                grph = ((Drawable)classif).graph();
                visualizeTree2(grph,"J48 Tree Mode 2");
            }
            catch (Exception e) {
                errorMessage = "Si è verificato un errore!\n(dataset_EventPerformed -
03)";
                JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                System.err.println("J48 - Error: " + e);
                e.printStackTrace();
```

```
                }
            }
        }
        catch (Exception e) {
           errorMessage = "Si è verificato un errore!\n(dataset_EventPerformed - 04)";
           JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOptionPa-
ne.ERROR_MESSAGE);
           System.err.println("J48 - Error: " + e);
           e.printStackTrace();
        }
        System.out.println("J48 - Fine metodo dataset_EventPerformed");
    }

    /* Metodo che visualizza una finestra contenente l'albero e le principali
     * informazioni statistiche riguardanti l'elaborazione. */
    protected void visualizeTree2(String dottyString, String treeName) {
        final javax.swing.JFrame jf = new javax.swing.JFrame("Tree Visualizer: " + tree-
Name);
        jf.setSize(600,500);
        jf.getContentPane().setLayout(new BorderLayout());
        TreeVisualizer tv = new TreeVisualizer(null, dottyString, new PlaceNode2());
        jf.getContentPane().add(tv, BorderLayout.CENTER);
        TextArea ta = new TextArea(8,25);
        ta.setBackground(Color.WHITE);
        ta.setEditable(false);
        ta.setFont(new Font("TimesRoman",Font.PLAIN,16));
        jf.getContentPane().add(new JScrollPane(ta), BorderLayout.SOUTH);
        ta.setText("Informazioni statistiche sull'elaborazione:\n\n");
        try {
            ta.append("\n" + classif);
            ta.append(eva.toSummaryString(true));
            ta.append("\n\n" + eva.toMatrixString());
            ta.append("\n\n" + eva.toClassDetailsString());
        }
        catch (Exception e) {
            System.err.println("Error: " + e);
            e.printStackTrace();
        }
        jf.addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent e) {
                jf.dispose();
            }
        });
        jf.setVisible(true);
     }
}
```

Il componente NeuralN

File **NeuralN.java**:

```
package classificatori.neural;

import java.beans.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.net.*;
import weka.core.*;
import weka.classifiers.neural.*;
```

```
import weka.classifiers.*;
import weka.classifiers.evaluation.*;
import support.*;

/** Classe che elabora un dataset utilizzando un algoritmo di classificazione
 *  che fa uso di una rete neurale back propagation. Questo Bean incapsula ed estende le
 *  funzionalità della classe weka.classifiers.neural.NeuralNetwork . */
public class NeuralN extends Canvas implements java.io.Serializable {

    private ImageIcon iconaBean;
    private Instances trainDataset;  // dataset da elaborare (per costruire il classifi-
catore)
    private Instances testDataset;  // dataset da elaborare (per testare il classificato-
re)
    private NeuralNetwork classif;     // oggetto classificatore (contiene l'algoritmo
per elaborare il dataset)
    transient private Evaluation eva;     // oggetto che utilizza il classificatore per
classificare dataset
    private String errorMessage;  // per comunicare messaggi di errore all'utente

    /** Creates new NeuralN */
    public NeuralN() {
        setSize( getPreferredSize() );
        repaint();
    }

    /** Metodi per la visualizzazione del Bean */
    public void paint( Graphics g ) {
        this.setBackground(Color.ORANGE);
        Class cl = this.getClass();
        URL url = cl.getResource("neural_32.GIF");
        if (url!=null) {
            Image img = Toolkit.getDefaultToolkit().getImage(url);
            iconaBean = new ImageIcon(img);
            iconaBean.paintIcon(this,g,(this.getWidth()-32)/2,(this.getHeight()-32)/2);
        }
    }

    public Dimension getPreferredSize() {
       return new Dimension( 50, 50 );
    }

    public Dimension getMinimumSize() {
       return getPreferredSize();
    }

    public Dimension getMaximumSize() {
       return getPreferredSize();
    }

    /** Metodi tipici del funzionamento del Bean */

    /* La seguente coppia di metodi definisce la proprietà trainingTime
     * che definisce il numero di epoche attraverso cui avviene l'addestramento. */
    public void setTrainingTime(int n) {
       if (classif != null)
          classif.setTrainingTime(n);
    }

    public int getTrainingTime() {
       if (classif != null)
           return classif.getTrainingTime();
       else return 500;
    }

    /* Metodi per la gestione degli eventi propri del Bean */
    /** Metodo che elabora uno o due dataset giunti tramite un DatasetEventObject */
    public void dataset_EventPerformed(DatasetEventObject dso) {

        System.out.println("NeuralN - Inizio metodo dataset_EventPerformed");
        try {
            if (dso.getTrainDataset() != null) {  // conserva il vecchio dataset nel caso
```

```
in cui si riceve solo i dataset di Test
                trainDataset = dso.getTrainDataset();
            }
            testDataset = dso.getTestDataset();
            System.out.println("\nNeuralN - Train dataset ricevuto tramite apposito even-
to: \n\n" + trainDataset + "\n\n");
            System.out.println("\nNeuralN - Test dataset ricevuto tramite apposito even-
to: \n\n" + testDataset + "\n\n");
            if (testDataset == null) {
                /* Solo dataset di Train */
                try {
                    classif = null;  // azzera un precedente classificatore, contrasse-
gnandolo per il garbage collector
                    classif = new NeuralNetwork();
                    classif.buildClassifier(trainDataset);
                    eva = new Evaluation(trainDataset);
                    eva.evaluateModel(classif, trainDataset);
                    visualizeResult();
                }
                catch (Exception e) {
                    errorMessage = "Si è verificato un errore!\n(dataset_EventPerformed -
01)";
                    JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                    System.err.println("NeuralN - Error: " + e);
                    e.printStackTrace();
                }
            }
            else if (trainDataset == null) {
                /* Solo dataset di Test */
                try {
                    if ( classif != null) {
                        eva = new Evaluation(trainDataset);
                        eva.evaluateModel(classif, testDataset);
                        visualizeResult();
                    }
                    else {
                        errorMessage = "Prima è necessario costruire un classificatore\n
utilizzando un dataset di Train!";
                        JOptionPane.showMessageDialog(null, errorMessage, "Errore!", JOp-
tionPane.ERROR_MESSAGE);
                    }
                }
                catch (Exception e) {
                    errorMessage = "Si è verificato un errore!\n(dataset_EventPerformed -
02)";
                    JOptionPane.showMessageDialog(null, errorMessage, "Errore!", JOp-
tionPane.ERROR_MESSAGE);
                    System.err.println("NeuralN - Errore: " + e);
                    e.printStackTrace();
                }
            }
            else if (trainDataset == testDataset) {
                /* Un unico dataset per Train e per Test */
                JOptionPane askMethod = new JOptionPane();
                Object[] possibleValues = { "Cross Validation", "Percentage Split" };
                Object selectedValue = JOptionPane.showInputDialog(null,  "Come creare i
dataset per Train e Test?", "Train & Test",
                    JOptionPane.QUESTION_MESSAGE, null,possibleValues, possibleVa-
lues[0]);
                if ( selectedValue == possibleValues[0] ) { // Cross Validation
                    String folds = JOptionPane.showInputDialog("Number of Folds in Cross
Validation\n (must be greater than 1)");
                    int numFolds;
                    try {
                        numFolds = Integer.parseInt(folds);
                    }
                    catch (Exception e) {            // l'input non è corretto
                        errorMessage = "Invalid value!\n I'm setting Folds to 2";
                        JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                        System.err.println("Error: " + e);
```

```
                        e.printStackTrace();
                        numFolds = 2;
                    }
                    classif = null;  // azzera un precedente classificatore, contrasse-
gnandolo per il garbage collector
                    classif = new NeuralNetwork();
                    trainDataset.randomize(new Random());
                    eva = new Evaluation(trainDataset);
                    if ( numFolds > 1 && numFolds <= trainDataset.numInstances()) {
                        eva.crossValidateModel(classif, trainDataset, numFolds);
                    }
                    else {
                        errorMessage = "Invalid value!\n Number of Folds must be > 1\n" +
                            "and <= of the number of instances (" + trainData-
set.numInstances() +
                            ")\n I'm setting Folds to 2";
                        JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                        eva.crossValidateModel(classif, trainDataset, 2);
                    }
                    visualizeResult();
                }
                else if ( selectedValue == possibleValues[1] ) {   // Percentage Split
                    String perc = JOptionPane.showInputDialog("Insert percentage split of
Train\n ( must be >0 and <100 )");
                    int percentage;
                    try {
                        percentage = Integer.parseInt(perc);
                    }
                    catch (Exception e) {            // l'input non è corretto
                        errorMessage = "Invalid value!\n I'm setting percentage to 50";
                        JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                        System.err.println("Error: " + e);
                        e.printStackTrace();
                        percentage = 50;
                    }
                    if ( percentage > 0 && percentage < 100) {
                        classif = null;  // azzera un precedente classificatore, contras-
segnandolo per il garbage collector
                        classif = new NeuralNetwork();
                        Instances provv = trainDataset;
                        provv.randomize(new Random());
                        int trainSize = provv.numInstances() * percentage / 100;
                        int testSize = provv.numInstances() - trainSize;
                        testDataset = new Instances(provv, trainSize, testSize);
                        trainDataset = new Instances(provv, 0, trainSize);
                        classif.buildClassifier(trainDataset);
                        eva = new Evaluation(trainDataset);
                        eva.evaluateModel(classif, trainDataset);
                        eva.evaluateModel(classif, testDataset);
                        visualizeResult();
                    }
                    else {
                        errorMessage = "Invalid value!\n Percentage must be >0 and
<100\n";
                        JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                        }
                }
                else {  }  // E' stato scelto il pulsante ANNULLA
            }
            else  {
                /* Due distinti dataset per Train e per Test */
                try {
                    classif = null;  // azzera un precedente classificatore, contrasse-
gnandolo per il garbage collector
                    classif = new NeuralNetwork();
                    classif.buildClassifier(trainDataset);
                    eva = new Evaluation(trainDataset);
                    eva.evaluateModel(classif, trainDataset);
                    eva.evaluateModel(classif, testDataset);
```

```
                    visualizeResult();
                }
                catch (Exception e) {
                    errorMessage = "Si è verificato un errore!\n(dataset_EventPerformed -
03)";
                    JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                    System.err.println("NeuralN - Error: " + e);
                    e.printStackTrace();
                }
            }
        }
        catch (Exception e) {
           errorMessage = "Si è verificato un errore!\n(dataset_EventPerformed - 04)";
           JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOptionPa-
ne.ERROR_MESSAGE);
           System.err.println("NeuralN - Error: " + e);
           e.printStackTrace();
        }
        System.out.println("NeuralN - Fine metodo dataset_EventPerformed");
    }

    /* Metodo che visualizza una finestra contenente le principali
     * informazioni statistiche riguardanti l'elaborazione. */
    protected void visualizeResult() {
        final javax.swing.JFrame jf = new javax.swing.JFrame("Neural Network Algorithm");
        jf.setSize(600,500);
        jf.getContentPane().setLayout(new BorderLayout());
        TextArea ta = new TextArea(8,25);
        ta.setBackground(Color.WHITE);
        ta.setEditable(false);
        ta.setFont(new Font("TimesRoman",Font.PLAIN,16));
        jf.getContentPane().add(new JScrollPane(ta), BorderLayout.CENTER);
        ta.setText("Informazioni statistiche sull'elaborazione:\n\n");
        try {
            ta.append("\n" + classif);
            ta.append(eva.toSummaryString(true));
            ta.append("\n\n" + eva.toMatrixString());
            ta.append("\n\n" + eva.toClassDetailsString());
        }
        catch (Exception e) {
            System.err.println("Error: " + e);
            e.printStackTrace();
        }
        jf.addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent e) {
                jf.dispose();
            }
        });
        jf.setVisible(true);
     }
}
```

Il componente EMCluster

File **EMCluster.java**:

```
package clusterer;

import java.beans.*;
import javax.swing.*;
import java.awt.*;
```

```
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.net.*;
import weka.core.*;
import weka.clusterers.*;
import support.*;

/** Classe che elabora un dataset utilizzando l'algoritmo di clustering EM (estimation
 *  maximization). Questo Bean incapsula ed estende le funzionalità della classe
 *  weka.clusterers.EM. */
public class EMCluster extends Canvas implements java.io.Serializable {

    private ImageIcon iconaBean;
    private Instances trainDataset;  // dataset da elaborare (per costruire il modello)
    private Instances testDataset;  // dataset da elaborare (per testare il modello)
    private EM clust;  // oggetto clusterer (contiene l'algoritmo per elaborare il data-
set)
    transient private ClusterEvaluation eva;     // oggetto che utilizza il dataset per
costruire il modello
    private String errorMessage;  // per comunicare messaggi di errore all'utente


    /** Creates new EMCluster */
    public EMCluster() {
        setSize( getPreferredSize() );
        repaint();
    }

    /** Metodi per la visualizzazione del Bean */
    public void paint( Graphics g ) {
        this.setBackground(Color.MAGENTA);
        Class cl = this.getClass();
        URL url = cl.getResource("EM_32.GIF");
        if (url!=null) {
            Image img = Toolkit.getDefaultToolkit().getImage(url);
            iconaBean = new ImageIcon(img);
            iconaBean.paintIcon(this,g,(this.getWidth()-32)/2,(this.getHeight()-32)/2);
        }
    }

    public Dimension getPreferredSize() {
       return new Dimension( 50, 50 );
    }

    public Dimension getMinimumSize() {
       return getPreferredSize();
    }

    public Dimension getMaximumSize() {
       return getPreferredSize();
    }

    /** Metodi tipici del funzionamento del Bean */

    /* La seguente coppia di metodi definisce la proprietà minStdDev della classe EM,
     * che rappresenta la minima deviazione standard ammissibile. */
    public void setMinStdDev(double d) {
        clust.setMinStdDev(d);
    }

    public double getMinStdDev() {
       if (clust != null)
           return clust.getMinStdDev();
       else return 1.0E-6;
    }

    /* La seguente coppia di metodi definisce la proprietà maxIterations della classe EM,
     * che rappresenta il massimo numero di iterazioni da effettuare. */
    public void setMaxIterations(int n) {
       try {
           clust.setMaxIterations(n);
```

```
            }
            catch (Exception e) {            // l'input non è corretto
                System.err.println("EMCluster - Errore nel settare maxIterations: \n" + e);
                System.err.println("Settaggio al valore di default: 100");
                try {
                    clust.setMaxIterations(100);
                }
                catch (Exception ex) {            // l'input non è corretto
                   System.err.println("EMCluster - Errore nel settare maxIterations: " + ex);
                   System.err.println("Impossibile settare al valore di default");
                }
            }
        }

        public int getMaxIterations() {
            if (clust != null)
                return clust.getMaxIterations();
            else return 100;
        }

        /* La seguente coppia di metodi definisce la proprietà numClusters della classe EM,
         * che fissa il numero di cluster. Se il valore è posto a -1 il numero di cluster
         * viene selezionato automaticamente dall'algoritmo mediante Cross Validation. */
        public void setNumClusters(int n) {
            try {
                clust.setNumClusters(n);
            }
            catch (Exception e) {            // l'input non è corretto
                System.err.println("EMCluster - Errore nel settare numClusters: \n" + e);
                System.err.println("Settaggio al valore di default: -1");
                try {
                    clust.setNumClusters(-1);
                }
                catch (Exception ex) {            // l'input non è corretto
                   System.err.println("EMCluster - Errore nel settare numClusters: \n" + ex);
                   System.err.println("Impossibile settare al valore di default");
                }
            }
        }

        public int getNumClusters() {
            if (clust != null)
                return clust.getNumClusters();
            else return -1;
        }

        /* La seguente coppia di metodi definisce la proprietà seed della classe EM,
         * che rappresenta il "seme" per la generazione di numeri random. */
        public void setSeed(int n) {
            clust.setSeed(n);
        }

        public int getSeed() {
            if (clust != null)
                return clust.getSeed();
            else return 100;
        }


        /* Metodi per la gestione degli eventi propri del Bean */
        /** Metodo che elabora uno o due dataset giunti tramite un DatasetEventObject */
        public void dataset_EventPerformed(DatasetEventObject dso) {

            try {
                if (dso.getTrainDataset() != null) {  // conserva il vecchio dataset nel caso
in cui si riceve solo i dataset di Test
                    trainDataset = dso.getTrainDataset();
                }
                testDataset = dso.getTestDataset();
                System.out.println("\nEMCluster - Train dataset ricevuto tramite apposito
evento: \n\n" + trainDataset + "\n\n");
                System.out.println("\nEMCluster - Test dataset ricevuto tramite apposito
```

```
evento: \n\n" + testDataset + "\n\n");
            if (testDataset == null) {
                /* Solo dataset di Train */
                try {
                    clust = null;  // azzera un precedente modello, contrassegnandolo per
il garbage collector
                    clust = new EM();
                    clust.buildClusterer(trainDataset); // Costruisce il modello
                    eva = new ClusterEvaluation();
                    eva.setClusterer(clust); // Utilizza un modello già costruito
                    visualizeResult(false);
                }
                catch (Exception e) {
                    errorMessage = "Si è verificato un errore!\n(dataset_EventPerformed -
01)";
                    JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                    System.err.println("EMCluster - Error: " + e);
                    e.printStackTrace();
                }
            }
            else if (trainDataset == null) {
                /* Solo dataset di Test */
                try {
                    if ( clust != null) {
                        eva = new ClusterEvaluation();
                        eva.setClusterer(clust); // Utilizza un modello già costruito
                        eva.evaluateClusterer(testDataset);
                        visualizeResult(true);
                    }
                    else {
                        errorMessage = "Prima è necessario costruire un modello\n utiliz-
zando un dataset di Train!";
                        JOptionPane.showMessageDialog(null, errorMessage, "Errore!", JOp-
tionPane.ERROR_MESSAGE);
                    }
                }
                catch (Exception e) {
                    errorMessage = "Si è verificato un errore!\n(dataset_EventPerformed -
02)";
                    JOptionPane.showMessageDialog(null, errorMessage, "Errore!", JOp-
tionPane.ERROR_MESSAGE);
                    System.err.println("EMCluster - Errore: " + e);
                    e.printStackTrace();
                }
            }
            else if (trainDataset == testDataset) {
                /* Un unico dataset per Train e per Test */
                JOptionPane askMethod = new JOptionPane();
                Object[] possibleValues = { "Cross Validation", "Percentage Split" };
                Object selectedValue = JOptionPane.showInputDialog(null,  "Come creare i
dataset per Train e Test?", "Train & Test",
                    JOptionPane.QUESTION_MESSAGE, null,possibleValues, possibleVa-
lues[0]);
                if ( selectedValue == possibleValues[0] ) { // Cross Validation
                    clust = null;  // azzera un precedente modello, contrassegnandolo per
il garbage collector
                    clust = new EM();
                    trainDataset.randomize(new Random());
                    clust.buildClusterer(trainDataset); // Costruisce il modello
                    eva = new ClusterEvaluation();
                    eva.setClusterer(clust); // Utilizza un modello già costruito
                    eva.evaluateClusterer(trainDataset);
                    visualizeResult(true);
                }
                else if ( selectedValue == possibleValues[1] ) {   // Percentage Split
                    String perc = JOptionPane.showInputDialog("Insert percentage split of
Train\n ( must be >0 and <100 )");
                    int percentage;
                    try {
                        percentage = Integer.parseInt(perc);
                    }
```

```
                    catch (Exception e) {            // l'input non è corretto
                        errorMessage = "Invalid value!\n I'm setting percentage to 50";
                        JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                        System.err.println("Error: " + e);
                        e.printStackTrace();
                        percentage = 50;
                    }
                    if ( percentage > 0 && percentage < 100) {
                        clust = null;  // azzera un precedente modello, contrassegnandolo
per il garbage collector
                        clust = new EM();
                        Instances provv = trainDataset;
                        provv.randomize(new Random());
                        int trainSize = provv.numInstances() * percentage / 100;
                        int testSize = provv.numInstances() - trainSize;
                        testDataset = new Instances(provv, trainSize, testSize);
                        trainDataset = new Instances(provv, 0, trainSize);
                        clust.buildClusterer(trainDataset); // Costruisce il modello
                        eva = new ClusterEvaluation();
                        eva.setClusterer(clust); // Utilizza un modello già costruito
                        eva.evaluateClusterer(testDataset);
                        visualizeResult(true);
                    }
                    else {
                        errorMessage = "Invalid value!\n Percentage must be >0 and
<100\n";
                        JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                    }
                }
                else {  }  // E' stato scelto il pulsante ANNULLA
            }
            else  {
                /* Due distinti dataset per Train e per Test */
                try {
                    clust = null;  // azzera un precedente modello, contrassegnandolo per
il garbage collector
                    clust = new EM();
                    clust.buildClusterer(trainDataset); // Costruisce il modello
                    eva = new ClusterEvaluation();
                    eva.setClusterer(clust); // Utilizza un modello già costruito
                    eva.evaluateClusterer(testDataset);
                    visualizeResult(true);
                }
                catch (Exception e) {
                    errorMessage = "Si è verificato un errore!\n(dataset_EventPerformed -
03)";
                    JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOp-
tionPane.ERROR_MESSAGE);
                    System.err.println("EMCluster - Error: " + e);
                    e.printStackTrace();
                }
            }
        }
        catch (Exception e) {
           errorMessage = "Si è verificato un errore!\n(dataset_EventPerformed - 04)";
           JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOptionPa-
ne.ERROR_MESSAGE);
           System.err.println("EMCluster - Error: " + e);
           e.printStackTrace();
        }
        System.out.println("EMCluster - Fine metodo dataset_EventPerformed");
    }

    /* Metodo che visualizza una finestra contenente le principali
     * informazioni statistiche riguardanti l'elaborazione. */
    protected void visualizeResult(boolean evaluated) {
        final javax.swing.JFrame jf = new javax.swing.JFrame("EM Algorithm");
        jf.setSize(600,500);
        jf.getContentPane().setLayout(new BorderLayout());
        TextArea ta = new TextArea(8,25);
```

```
        ta.setBackground(Color.WHITE);
        ta.setEditable(false);
        ta.setFont(new Font("TimesRoman",Font.PLAIN,16));
        jf.getContentPane().add(new JScrollPane(ta), BorderLayout.CENTER);
        ta.setText("Informazioni statistiche sull'elaborazione:\n\n");
        try {
            if (evaluated)
               ta.append(eva.clusterResultsToString());
            else ta.append(clust.toString());
        }
        catch (Exception e) {
            System.err.println("Error: " + e);
            e.printStackTrace();
        }
        jf.addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent e) {
                jf.dispose();
            }
        });
        jf.setVisible(true);
    }
}
```

Il componente DataStreamMonitor

File **DataStreamMonitor.java**:

```
package monitor;

import java.beans.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.net.*;
import weka.core.*;
import weka.filters.*;
import support.*;

/** Classe in grado di visualizza il flusso di dati tra altri Bean. Questo Bean incapsula
 *  ed estende le funzionalità della classe weka.filters.AllFilter */
public class DataStreamMonitor extends Canvas implements java.io.Serializable {

    private boolean visualize = true;  // determina se questo Bean deve visualizzare o no
i dataset in transito
    private ImageIcon iconaBean;
    private Instances sorgenteTrain, sorgenteTest;  // dataset monitorati
    private AllFilter fil;    // classe WEKA usata per manipolare il dataset e le istanze
    private Instance istanza;  // memorizza una istanza (una tupla) del dataset
    private Vector instListeners = new Vector();
    private Vector datasetListeners = new Vector();  // memorizza l'elenco dei listener
registrati per l'evento DatasetEventObject
    private Button b1, b2;
    private File startPath;  // memorizza la directory di partenza
    private File actualPath;  // memorizza l'ultima directory visitata con JFileChooser
    private String errorMessage;  // per comunicare messaggi di errore all'utente

    /** Creates new InstanceMonitor */
    public DataStreamMonitor() {
        setSize( getPreferredSize() );
        repaint();
```

```
    }

    /** Metodi per la visualizzazione del Bean */
    public void paint( Graphics g ) {
        this.setBackground(Color.BLUE);
        Class cl = this.getClass();
        URL url = cl.getResource("monitor_32.GIF");
        if (url!=null) {
            Image img = Toolkit.getDefaultToolkit().getImage(url);
            iconaBean = new ImageIcon(img);
            iconaBean.paintIcon(this,g,(this.getWidth()-32)/2,(this.getHeight()-32)/2);
        }
    }

    public Dimension getPreferredSize() {
       return new Dimension( 50, 50 );
    }

    public Dimension getMinimumSize() {
       return getPreferredSize();
    }

    public Dimension getMaximumSize() {
       return getPreferredSize();
    }

    /** Metodi per la gestione degli eventi propri del Bean */
    public void addDatasetListener(DatasetListener listener) {
        datasetListeners.addElement(listener);
    }

    public void removeDatasetListener(DatasetListener listener) {
        datasetListeners.removeElement(listener);
    }

    /** Il seguente metodo invia ai Bean registrati i dataset di Train e di Test
      * ricevuti tramite evento DatasetEventObject. */
    public void startDatasetStreamOutput() {
        System.out.println("Monitor - Inizio metodo startDatasetStream");
        DatasetEventObject dso = new DatasetEventObject(this, sorgenteTrain, sorgente-
Test);
        for (int i = 0; i < datasetListeners.size(); i++) {
            System.out.println("Monitor - Inizio passo " + i + " del ciclo di startData-
setStream");
            DatasetListener dl = (DatasetListener) datasetListeners.elementAt(i);
            dl.dataset_EventPerformed(dso);
            System.out.println("Monitor - Fine passo " + i + " del ciclo di startDataset-
Stream");
        }
        System.out.println("Monitor - Fine metodo startDatasetStream");
    }

    /** Il seguente metodo ritorna l'evento DatasetEventObject captato da questo Bean.
      * Tale metodo è necessario per il funzionamento del Bean in BeanBuilder: in tale
      * ambiente sostituisce il metodo startDatasetStreamOutput(). */
    public DatasetEventObject obtainDatasetStream() {
        DatasetEventObject dso = new DatasetEventObject(this, sorgenteTrain, sorgente-
Test);
        return dso;
    }

    /** Il seguente metodo visualizza i dataset captati dal monitor */
    public void dataset_EventPerformed(DatasetEventObject dso) {
        System.out.println("Monitor - Inizio metodo dataset_EventPerformed");
        sorgenteTrain = dso.getTrainDataset();
        sorgenteTest = dso.getTestDataset();
        if ( visualize ) {
            openInfoWindow();
        }
        System.out.println("Monitor - Trasmissione del dataset");
        startDatasetStreamOutput();
        System.out.println("Monitor - Fine trasmissione. Fine metodo data-
```

```
set_EventPerformed");
    }

    /** Il seguente metodo apre una finestra per visualizzare il dataset captato dal
     *  monitor */
    public void openInfoWindow()   {
        final javax.swing.JFrame jf = new javax.swing.JFrame("Watching Dataset Stream");
        Panel panelButt = new Panel();
        ButtonHandler bh = new ButtonHandler();
        TextArea ta = new TextArea(10,25);

        b1 = new Button("Save Train File");
        b2 = new Button("Save Test File ");
        panelButt.add(b1);
        panelButt.add(b2);
        b1.addActionListener(bh);
        b2.addActionListener(bh);
        jf.setSize(450,550);
        jf.getContentPane().setLayout(new BorderLayout());
        ta.setBackground(Color.WHITE);
        ta.setEditable(false);
        ta.setFont(new Font("TimesRoman",Font.PLAIN,16));
        ta.setText("Dataset for training:\n\n");
        if (sorgenteTrain != null) { ta.append(sorgenteTrain.toString()); }
        else {
            ta.append("Inesistente (Null Pointer).\n\n");
            b1.setEnabled(false);
        }
        ta.append("\n\n\nDataset for testing:\n\n");
        if (sorgenteTest != null) { ta.append(sorgenteTest.toString()); }
        else {
            ta.append("Inesistente (Null Pointer).\n\n");
            b2.setEnabled(false);
        }
        jf.getContentPane().add(new JScrollPane(ta), BorderLayout.CENTER);
        jf.getContentPane().add(panelButt, BorderLayout.SOUTH);
        jf.addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent e) {
                jf.dispose();
            }
        });
        jf.setVisible(true);
    }

    /* I seguenti 3 metodi consentono di cambiare la proprietà visualization del Bean.
     * tale proprietà è legata alla variabile visualize, e determina se il Bean deve
     * o no aprire una finestra per visualizzare i dataset in transito. */
    public boolean getVisualization() {
        return visualize;
    }

    public void setVisualization( boolean status ) {
        visualize = status;
    }

    public void changeVisualization() {
        if ( visualize ) setVisualization(false);
        else setVisualization(true);
    }

private class ButtonHandler implements ActionListener, java.io.Serializable  {
      public void actionPerformed( ActionEvent e )
      {
         File fileName;

         if (startPath == null) {
             startPath = new File(".");
             actualPath = startPath;
         }
         FileDialog fd = new FileDialog(new Frame(), "Save Dataset", FileDialog.SAVE);
         fd.setDirectory(actualPath.getPath());
         fd.show();
```

```
            String fname = fd.getFile();
            if (fname != null) {     // L'utente ha premuto il pulsante SALVA
                try {
                    String dname = fd.getDirectory();
                    fileName = null;  // contrassegna per il Garbage Collector il precedente
oggetto fileName
                    fileName = new File(dname, fname);
                    actualPath = fileName.getAbsoluteFile();
                    PrintWriter out = new PrintWriter(new BufferedWriter(new FileWri-
ter(fileName)));
                    if ( e.getSource() == b1 ) {
                        out.println(sorgenteTrain.toString());
                        out.close();
                        b1.setEnabled(false);
                        if (sorgenteTrain == sorgenteTest) {
                            /* Un unico dataset per Train e per Test */
                            b2.setEnabled(false);
                        }
                    }
                    else if ( e.getSource() == b2 ) {
                        out.println(sorgenteTest.toString());
                        out.close();
                        b2.setEnabled(false);
                        if (sorgenteTrain == sorgenteTest) {
                            /* Un unico dataset per Train e per Test */
                            b1.setEnabled(false);
                        }
                    }
                }
                catch (Exception exc) {
                   errorMessage = "Si è verificato un errore.\n (ButtonHandler - 01)";
                   JOptionPane.showMessageDialog(null, errorMessage, "Error!", JOptionPa-
ne.ERROR_MESSAGE);
                   System.err.println("Error: " + exc);
                   exc.printStackTrace();
                }
            }
            else {  // L'utente ha premuto il pulsante ANNULLA
                System.err.println("Save Dialog - You chose CANCEL option");
            }
        }
    }
}
```

L'evento DatasetEventObject

File **DatasetEventObject.java**:

```
package support;

import java.util.*;
import weka.core.*;

public class DatasetEventObject extends EventObject {

    private Instances trainDataset, testDataset;

    /** Creates a new instance of DatasetEventObject */
    public DatasetEventObject(Object source, Instances train, Instances test) {
        super(source);
        this.trainDataset = train;
        this.testDataset = test;
```

```
    }

    public Instances getTrainDataset() {
        return trainDataset;
    }

    public Instances getTestDataset() {
        return testDataset;
    }

}
```

File **DatasetListener.java**:

```
package support;

import java.util.*;
import java.awt.event.*;

public interface DatasetListener extends EventListener {

    public void dataset_EventPerformed (DatasetEventObject e);

}
```

A P P E N D I C E B

Modifiche apportate al BeanBox

Per poter visualizzare anche le proprietà read-only di un bean, oltre alle proprietà read-write, limitatamente però a quelle proprietà read-only che possono essere visualizzate in un campo stringa, sono state effettuate le seguenti modifiche al codice sorgente del BeanBox:

File **PropertySheet.java**:

E' stato modificato il metodo *setTarget*:

- Il codice delle righe 107-110 originariamente era:

```
// Only display read/write properties.
if (getter == null || setter == null) {
  continue;
}
```

ed è stato modificato nel seguente modo:

```
// Only display read/write and read-only properties.
if (getter == null) {
  continue;
}
```

in quanto il BeanBox, conformemente alle regole dell'introspezione, identifica le proprietà read-only come quelle proprietà che non hanno metodo *set* (`setter == null`), e le proprietà read-write come quelle proprietà che hanno sia il metodo *set* che il metodo *get* (`setter != null && getter != null`). Le proprietà che non hanno metodo *get* (`getter == null`) sono le proprietà wri-

te-only.

- Il codice delle righe 162-173 originariamente era:

```
if (editor.isPaintable() && editor.supportsCustomEditor()) {
    view = new PropertyCanvas(frame, editor);
} else if (editor.getTags() != null) {
    view = new PropertySelector(editor);
} else if (editor.getAsText() != null) {
    String init = editor.getAsText();
    view = new PropertyText(editor);
} else {
    System.err.println("Warning: Property \"" + name
            + "\" has non-displayabale editor. Skipping.");
    continue;
}
```

ed è stato modificato nel seguente modo:

```
if (editor.isPaintable() && editor.supportsCustomEditor()) {
    if (setter == null) {continue;}
    view = new PropertyCanvas(frame, editor);
} else if (editor.getTags() != null) {
    if (setter == null) {continue;}
    view = new PropertySelector(editor);
} else if (editor.getAsText() != null) {
    String init = editor.getAsText();
    view = new PropertyText(editor);
    if (setter == null) {
       view = new PropertyText(editor,false);
    }
    else {view = new PropertyText(editor,true);}
} else {
    System.err.println("Warning: Property \"" + name
            + "\" has non-displayabale editor. Skipping.");
    continue;
}
```

per visualizzare unicamente le proprietà read-only che possono essere mostrate in un campo stringa e per rendere non editabili i campi stringa relativi

alle proprietà read-only.

File **PropertyText.java**:

E' stato modificato il costruttore:

- Originariamente era:

```
PropertyText(PropertyEditor pe) {
    super(pe.getAsText());
    editor = pe;
    addKeyListener(this);
    addFocusListener(this);
}
```

ed è stato modificata nel seguente modo:

```
PropertyText(PropertyEditor pe, boolean modificabile) {
    super(pe.getAsText());
    editor = pe;
    if (modificabile) {
         addKeyListener(this);
         addFocusListener(this);
    }
    setEditable(modificabile);
}
```

per rendere non editabili i campi stringa relativi alle proprietà read-only.

Effettuate le sopracitate modifiche, è necessario compilare i file Java e sostituire i file *.class* ottenuti agli originali, che si trovano nella sottodirectory *classes\sun\beanbox*, considerando come origine la directory in cui si trova il file **run.bat** che lancia il BeanBox.

A P P E N D I C E C

Uso dei componenti Java Beans

Modalità di utilizzo dei JavaBean in NetBeans

Caricamento di un bean in una Palette del GUI Editor :

1) Fare click col tasto destro del mouse su *Filesystems* (nel riquadro a sinistra), selezionare *Mount* *Local Directory*, e selezionare la cartella contenente i file .**JAR** dei bean.

2) Per ciascun file **.JAR**, fare click col tasto destro del mouse su di esso e selezionare *Mount JAR*. A questo punto il file .**JAR** compare nel *Filesystems* ; navigando nelle sue sottocartelle raggiungere il bean (è il file .**class** che ha lo stesso nome del file .**JAR**), fare click col tasto destro del mouse su di esso, e selezionare *Tools* *Add to Component Palette…* per aggiungere il bean ad una Palette del GUI Editor.

In alternativa, se il bean è stato realizzato con NetBeans, o se si dispone dei file sorgenti, è possibile caricare il bean in una Palette del GUI Editor direttamente dai file sorgente:

1) Fare click col tasto destro del mouse su *Filesystems* (nel riquadro a sinistra), selezionare *Mount* *Local Directory*, e selezionare la cartella contenente i file sorgenti.

2) Fare click col tasto destro del mouse sul file sorgente, e selezionare *Tools* *Add to Component Palette…* per aggiungere il bean ad una Palette del GUI Editor. Ripetere l'operazione per ciascun bean.

Realizzazione di una applicazione:

1) Fare click col tasto destro del mouse su una cartella "montata" nel *Filesystems*, selezionare *New Java GUI Forms JFrame*, e dopo aver scelto il nome fare click su *Finish*.

2) Aggiungere un *JPanel* al *JFrame* . Sostituire il *FlowLayout* del *JPanel* con un *Null Layout* (facendo click col tasto destro del mouse sul layout). Nel tab *Other Properties* del *JPanel*, cambiare i valori di *preferredSize* (ponendoli ad es. a [400, 300]).

Il *JPanel* serve a dare una dimensione non nulla alla *JFrame*. Il *Null Layout* consente di collocare i bean in qualsiasi posizione nel *JPanel* senza che vengano ridimensionati automaticamente.

3) Collocare i bean nel *JPanel* (ignorare l'eventuale messaggio di errore che compare piazzando il bean *InputDataset*).

4) Collegare i bean: dopo aver premuto il pulsante *Connection Mode* (a sinistra delle palette del *GUI Editor*), cliccare sul bean sorgente (ad es. *InputDataset*) e poi sul bean target (ad es. *J48Bean*). Compare un wizard che guida nella connessione dei due bean: nella prima schermata (*Select Source Event*) selezionare *dataset_EventPerformed* da *datasetListener;* nella seconda schermata (*Specify Target Operation*) selezionare *dataset_EventPerformed* dal gruppo *Method Call*; nella terza schermata (*Enter Parameters*) selezionare la voce *User Code* e scrivere **evt** (nel codice che verrà generato automaticamente, controllare che **evt** sia il nome del parametro di tipo *support.DatasetEventObject*).

5) A questo punto è sufficiente selezionare *Execute* dal menu *Build* per far partire l'applicazione.

Modalità di utilizzo dei JavaBean in BeanBox

1) Assicurarsi che nel file **Run.bat** la riga contenente l'istruzione:

set CLASSPATH

contenga anche il percorso del file **.JAR** di WEKA (ad es. `c:\programmi\Weka-3-2\weka.jar;`).

2) Nel menu *File*, selezionare la voce *LoadJar…* e caricare i bean.

3) Dopo aver selezionato i bean nella finestra *ToolBox*, collocarli nella finestra principale del BeanBox. Per collegarli, selezionare il bean sorgente dell'evento (ad es. *InputDataset*), poi dal menu *Edit* selezionare la voce *Events* *datasetListener* *dataset_EventPerformed*, ed infine cliccare sul bean target (ad es. *J48Bean*). Compare la finestra *EventTargetDialog*, in cui si deve selezionare il metodo del bean target che si vuole lanciare quando si verifica l'evento: scegliere *dataset_EventPerformed* e fare click su OK. A questo punto il BeanBox genera e compila una *Adaptor Class* (l'oggetto che ha il compito di connettere i due bean).

4) A questo punto l'applicazione è già funzionante (anche in Design Mode), e può essere salvata selezionando la voce *Save…* dal menu *File*. Quando si richiama un progetto precedentemente salvato (selezionando la voce *Load…* dal menu *File*), assicurarsi che i bean siano presenti nel *ToolBox*, altrimenti prima è necessario procedere al caricamento dei corrispondenti file **.JAR** (vedi punto 2).

Modalità di utilizzo dei JavaBean in BeanBuilder

1) Assicurarsi che nel file **Run.bat** la riga contenente:

set USE_JAVA …

sia preceduta da una riga contenente:

set JAVA_HOME=*percorso_in _cui_è_installato_JAVA*

Inoltre inserire nell'istruzione

set CLASSPATH

il percorso del file **.JAR** di WEKA

(ad es. `c:\programmi\Weka-3-2\weka.jar;`)

2) Se lanciando il file **Run.bat** il BeanBuilder non parte e compare la dicitura "Spazio di ambiente esaurito", fare click col tasto destro del mouse sull'icona del file **Run.bat**, selezionare *Proprietà*, e nel tab *Memoria* selezionare la massima memoria possibile per *Memoria convenzionale Ambiente iniziale*.

3) Nel menu *File*, selezionare la voce *Load Jar File* e caricare i bean.

4) Collocare i bean nella GUI vuota, e collegarli tramite i *connection handles* (i quattro quadratini scuri). Compare il wizard di connessione. Nella prima schermata (*Select Event Method …*) selezionare *dataset_EventPerformed* (da *datasetListener*); nella seconda schermata (*Select Target Method …*) selezionare *dataset_EventPerformed* dal gruppo *Event Adapter;* nella terza schermata (*Arguments for …*) selezionare *obtainDatasetStream()*. Attenzione: a causa di un probabile bug, la terza schermata consente di premere *Finish* anche se non è stato selezionato alcun metodo, per cui può sembrare che, quando compare un solo metodo, la selezione avvenga automaticamente; in realtà ciò non avviene, e se si preme *Finish* senza aver selezionato alcun metodo il BeanBuilder va in loop.

5) A questo punto l'applicazione è già funzionante (anche in Design Mode, come nel BeanBox).

BIBLIOGRAFIA

Database e Data mining:

[01] Paolo Giudici (2001), *Data mining. Metodi statistici per le applicazioni aziendali*, Ed. McGraw-Hill.

[02] Donato Venditti (2001), *Breve guida al Data Mining* [http://manuali.caltanet.it].

[03] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth (1996): *From data mining to Knowledge discovery in databases*. Menlo Park, Calif.: AAAI Press.

[04] *KDnuggets* [http://www.kdnuggets.com/index.html]. KDnuggets.

[05] Autori vari, *Data Mining* [http://www.megaputer.com/dm/dm101.php3]. Megaputer Intelligence, Inc.

[06] Ian H. Witten, Eibe Frank, *Data Mining: Practical machine learning tools with Java implementations*, Morgan Kaufmann, San Francisco.

[07] Antonio Picariello *Data Warehousing* [http://cds.unina.it/~picus/ sistemi_informativi.htm]. Dipartimento di Informatica e Sistemistica, Università di Napoli "Federico II".

[08] P.Atzeni, S.Ceri, S.Paraboschi, R.Tolone (1999), *Basi di dati (2ª Edizione)* Ed. McGraw-Hill Italia.

Linguaggio Java e Java Beans:

[09] Harvey M. Deitel, Paul J. Deitel (1999), *Java. Fondamenti di programmazione*, Ed. Apogeo.

[10] Harvey M. Deitel, Paul J. Deitel (1999), *Java. Tecniche avanzate di programmazione*, Ed. Apogeo.

[11] Autori vari, *Manuale Java*, [http://www.mokabyte.it]. Mokabyte s.r.l.

[12] Suzette Pelouch (2000), *Java Language Specification - Second Edition (Html version)*, [http://java.sun.com/docs/books/jls/second_edition/html/jTOC.doc.html]. Sun Microsystems, Inc.

[13] *Core Java Technologies Tech Tips* [http://java.sun.com]. Sun Microsystems, Inc.

[14] *JavaWorld Tips* [http://www.javaworld.com]. JavaWorld.com, an IDG company.

[15] *Javabeans component architecture documentation (Specifications, Tutorials, Articles)* [http://java.sun.com/products/javabeans/docs/]. Sun Microsystems, Inc.

[16] *Javabeans component architecture FAQ (Frequently Asked Questions)* [http://java.sun.com/products/javabeans/FAQ.html]. Sun Microsystems, Inc.

[17] *Javabeans FAQ Index By Topic* [http://www.jguru.com/faq/ topicindex.jsp?topic=JavaBeans]. jGuru.com.

Software:

[18] *PolyAnalyst 4.5* [http://www.megaputer.com]. Megaputer Intelligence, Inc.

[19] Ian H. Witten, Eibe Frank, *Weka 3: Machine Learning Software in Java* [http://www.cs.waikato.ac.nz/ml/weka/]. Department of Computer Science, University of Waikato, Hamilton, New Zealand.

[20] Alex Seewald, *Introduction to using Weka* [http://www.oefai.at/~alexsee/WEKA/].

[21] *NetBeans IDE 3.4.1* [http://www.netbeans.org]. Netbeans.org.

[22] *BeanBox (JavaBeans Development Kit - BDK 1.1)* [http://java.sun.com/products/javabeans/software/]. Sun Microsystems, Inc.

[23] *The Bean Builder v1.0 Beta* [http://java.sun.com/products/javabeans/software/]. Sun Microsystems, Inc.

Dataset:

[24] Blake, C.L. & Merz, C.J. (1998). *UCI Repository of machine learning databases* [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

[25] *CMU StatLib Datasets Archive* [http://stat.cmu.edu/datasets]. Department of Statistics, Carnegie Mellon University.

[26] Jeffrey S. Simonoff (1996). *Smoothing Methods in Statistics* [http://stat.cmu.edu/datasets/smoothmeth]. Springer-Verlag, New York.

[27] *Statistical Engineering Division of National Institute of Standards and Tecnology (NIST)* [http://www.itl.nist.gov/div898/education/datasets.htm]. Gaithersburg, MD and Boulder, CO.