UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI SCIENZE STATISTICHE

CORSO DI LAUREA IN STATISTICA E TECNOLOGIE INFORMATICHE

RELAZIONE FINALE

STRUMENTI OPEN SOURCE PER DATA MINING

Confronto tra Weka e R

RELATORE

Ch.mo Prof.

Adelchi Azzalini

CANDIDATO

Davide Zamperin

Matricola 415258/STI

ANNO ACCADEMICO 2006/2007

A mia moglie Micol Sara
e ai miei figli Michele, Giovanni e Rachele:
le vere gioie della mia vita

Indice generale

Introduzione

Gli ultimi decenni sono stati caratterizzati da una costante innovazione tecnologica, soprattutto nel campo informatico. Ogni generazione di processori incrementa significativamente la potenza di calcolo dei computer: sintomatico è il fatto che oggigiorno tutte le maggiori catene di supermercati offrono a prezzi molto convenienti potenti personal computer che nella maggior parte degli acquisti superano l'effettiva necessità domestica. Similmente per quanto riguarda la memorizzazione dei dati, sono nate nuove tecnologie e nuovi supporti per contenere le informazioni: si pensi solo alla grandissima diffusione delle *pen-drive* che arrivano ad avere la capacità di 2 gigabyte, nonostante la dimensione analoga ad una gomma da disegno.

Questo trend ha comportato la creazione di banche dati le cui dimensioni erano in passato impensabili per problemi di costi e accessibilità ai dati contenuti: sfruttando il progresso tecnologico, i grandi costruttori di database hanno sviluppato prodotti tali da consentire la creazione di database da diversi terabyte su complesse architetture di calcolo parallelo (SMP) e clustering, al fine di ottimizzare i tempi di risposta nelle varie operazioni di accesso alle informazioni (interrogazioni *SQL*). I risultati sono prodotti come *Oracle Parallel Server*, *IBM DB2 Parallel Edition* e *Informix Parallel Data Query*.

Naturalmente la disponibilità di tanti dati ha suscitato notevole interesse in particolari ambiti in cui l'informazione è alla base di un processo conoscitivo. Un esempio molto importante sono le carte fedeltà distribuite dai grandi supermercati: la possibilità di memorizzare tutti gli scontrini unita alla rintracciabilità degli acquirenti offerta da tali tessere permette al settore marketing di profilare in maniera semplice la clientela. In cambio di sconti o regali

promozionali, tali aziende conoscono perfettamente le esigenza di spesa e il comportamento economico dei propri clienti.

Tuttavia in questo nuovo scenario si sono delineate delle caratteristiche nuove rispetto alla classica concezione di analisi, quali l'elevata numerosità e dimensionalità dei dati. Tornando all'esempio delle carte fedeltà, è stata sviluppata appositamente una tecnica per poter analizzare le milioni di transazioni rappresentanti i carrelli della spesa in formato elettronico. Se da una parte gli strumenti e le tecniche classiche di analisi risultarono inadeguate, dall'altra l'elevata potenza di calcolo dei nuovi computer e il successo dell'Intelligenza Artificiale stimolarono gli specialisti del settore a percorrere nuove soluzioni che pochi anni prima erano proibitive: si pensi alle tante scelte effettuate in passato per rendere gli algoritmi maggiormente veloci in termini di elaborazione e poco dispendiosi in termini di occupazione di memoria, proprio a causa della limitata potenza computazionale allora disponibile.

In questo nuovo panorama, il data mining si propone come disciplina principe per l'analisi dei dati: essa raccoglie tecniche e algoritmi sviluppati nell'ambito della statistica classica e dell'Intelligenza Artificiale (specialmente la branca del *Machine Learning*) al fine di offrire all'analista gli strumenti più all'avanguardia. Nel tempo sono nati molti programmi commerciali con l'obiettivo di offrire uno strumento completo in termini di qualità e velocità con cui apprendere conoscenza: tra i pacchetti statistici troviamo *Enterprise Miner* di SAS, *Insightful Miner* di Mathsoft/S-Plus, *Clementine* di SPSS, e *KnowledgeSTUDIO* di Angoss; tra i prodotti sviluppati per l'integrazione nativa con i database, citiamo *Darwin* di Oracle e *Intelligent Miner* di IBM.

L'obiettivo della relazione consiste nello studio di soluzioni *open source* per il data mining: l'interesse non consiste solo nell'offrire all'analista uno strumento per quanto più completo possibile nella

gamma di soluzioni disponibili liberamente, ma anche nella valutazione del grado di cooperazione in altri applicativi. Quest'ultimo aspetto rappresenta uno dei pilastri della filosofia *Free Open Source Software*: la possibilità di integrare soluzioni sviluppate da terzi parti in un proprio progetto software. Un esempio di questa nuova concezione di programmazione è costituito dall'ambizioso progetto *SpagoBI* promosso da Engineering S.p.A., una delle più grandi realtà del mercato IT in Italia: esso si propone come *Business Intelligence Free Platform*, ossia un ambiente per la Business Intelligence costituito dall'integrazione di soluzioni esterne al progetto per coprire tutti gli aspetti funzionali correlati, quali l'organizzazione dei dati, l'interrogazione, il data mining, la pubblicazione e il controllo strutturato e dinamico.

La relazione è strutturata come segue: la prima sezione è dedicata alla definizione di data mining; le successive due sezioni introducono i due prodotti open source selezionati per lo studio, rispettivamente Weka e R. La quarta sezione contiene i risultati di un confronto descrittivo e computazionale dei due prodotti; termina una sezione con le considerazioni finali.

1. Data Mining

A causa della recente nascita, non esiste nella letteratura classica una definizione univoca di data mining: spesso ci si riferisce con il termine *Knowledge Discovery in Database* (KDD) per sottolineare che l'analisi è eseguita principalmente nel database, inteso come contenitore dei dati d'interesse. Come sempre la verità sta nel mezzo: se da una parte l'analisi non necessariamente può essere vincolata alla presenza di un database che contenga i dati, dall'altra storicamente furono proprio i maggiori costruttori di database a trovarsi per primi ad affrontare operativamente il problema della numerosità dei dati, al fine di garantire prestazioni e affidabilità nella gestione dei dati stessi. Questo permise loro un significativo vantaggio nello sviluppo di nuove tecniche per l'analisi, dal momento che la gestione dei dati è affidata al proprio software: l'esempio più immediato è l'*IBM Market Basket* per l'analisi del paniere della spesa.

Un altro fattore, che certo non aiuta l'individualizzazione di un approccio univoco, è il vasto campo di applicazione: si spazia dal marketing alle diverse tipologie di business, dalla medicina alle applicazioni biomediche e biologiche, dalla ricerca mirata all'individualizzazione di illeciti o truffe al *text mining* e *web mining*.

Nel seguito sono proposte alcune definizioni:

> *Data Mining, or Knowledge Discovery in Databases (KDD) as it is also known, is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data. This encompasses a number of different technical approaches, such as clustering, data summarization, learning classification rules, finding dependency net works, analysing changes, and detecting anomalies*

Usama Fayyad, Gregory Piatetsky-Shapiro e Padhraic Smyth [1.01].

> *Data Mining is the search for relationships and global patterns that exist in large databases but are `hidden' among the vast amount of data, such*

> *as a relationship between patient data and their medical diagnosis. These relationships represent valuable knowledge about the database and the objects in the database and, if the database is a faithful mirror, of the real world registered by the database*

Marcel Holshemier e Arno Siebes [1.02].

> *[Data mining] using a variety of techniques to identify nuggets of information or decision-making knowledge in bodies of data, and extracting these in such a way that they can be put to use in the areas such as decision support, prediction, forecasting and estimation. The data is often voluminous, but as it stands of low value as no direct use can be made of it; it is the hidden information in the data that is useful*

Dalla guida di *Clementine*, il prodotto di data mining sviluppato da SPSS.

A parte la pittoresca azione, dedotta dal nome, di setacciare i dati al fine di trovare l'oro, il data mining identifica il processo di selezione, esplorazione e modellazione di grandi quantità di dati al fine di scoprire regolarità e relazioni non note o ignorate precedentemente. Molto vicino all'analisi statistica, il data mining si distingue per il massiccio utilizzo di più tecniche di apprendimento computerizzate per la generazione e il confronto di modelli, al fine di identificare eventuali relazioni, trend e pattern presenti nei dati stessi.

Questa semplice definizione aiuta nel difficile compito di trovare una formulazione comune del processo di data mining a prescindere dai diversi contesti in cui viene utilizzato. Spesso ci si trova in difficoltà di fronte a citazioni sulla *Business Intelligence* e *KDD*: entrambe le procedure utilizzano il data mining con enfasi diversa a seconda che il problema in questione appartenga al mondo del business (ad esempio CRM) o ad un generico contesto. Viene di seguito proposto l'illuminante schema di Usama Fayyad, Gregory Piatetsky-Shapiro e Padhraic Smyth [1.01] che rende palese come il complesso processo di KDD utilizzi il data mining proprio nell'accezione con cui è stato precedentemente definito.

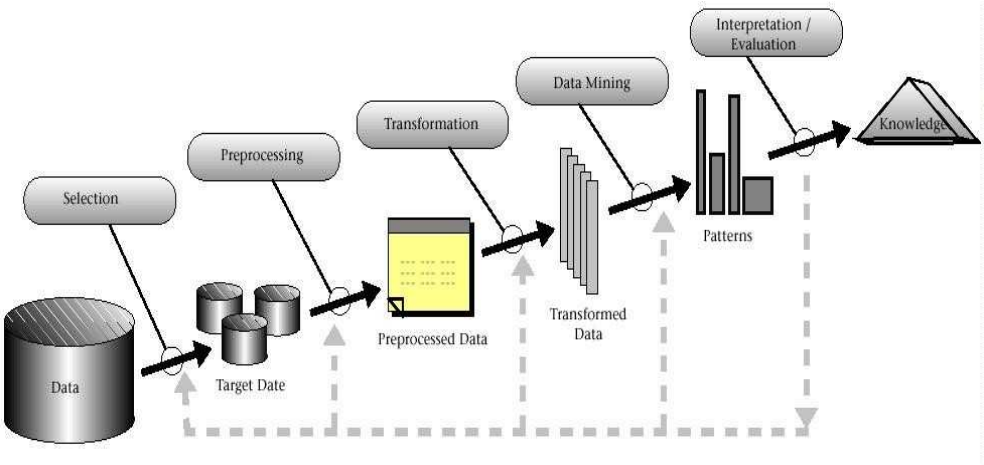

Figura 1.1: Le fase del KDD

Una volta focalizzato il punto di integrazione tra il data mining e altre procedure, l'attenzione vuole essere riportata sulla peculiarità dell'approccio di analisi proprio del data mining.

Anzitutto data mining non è sinonimo di magia: l'induzione di un modello o di una relazione partendo dai dati comporta un risultato strettamente legato alla qualità dei dati stessi. L'analista non solo deve padroneggiare le tecniche e gli algoritmi che tale disciplina mette a disposizione, ma deve anche possedere una profonda conoscenza del fenomeno che ha generato i dati oggetto dell'analisi. Gli specialisti del settore affermano che due sono le chiavi di successo del data mining: la precisa formulazione del problema da analizzare e l'utilizzo di dati "buoni". Solo un approccio sistematico garantisce dei risultati attendibili: per questo motivo molti produttori di software e organizzazioni di consulenza hanno tracciato un proprio percorso che assicuri il raggiungimento di buoni risultati. Ad esempio le società SPSS e SAS utilizzano rispettivamente 5A's (*Assess, Access, Analyze, Act and Automate*) e SEMMA (*Sample, Explorer, Modify, Model, Assess*). Tale urgenza è talmente sentita che la stessa Comunità Europea ha finanziato il progetto CRISP-DM (*Cross Industry Standard Process for Data Mining*) con il chiaro intento di definire un approccio standard ai progetti relativi al data mining: esso ha coinvolto un consorzio di produttori software del settore (tra cui

SPSS) e utilizzatori finali (come banche) in modo da concordare un modus operandi comune per la soluzione di problemi aziendali a prescindere dalla tipologia di business. Secondo il modello CRISP-DM, il ciclo di vita di un progetto di data mining è articolato in sei fasi non strettamente rigide e talvolta ricorsive:

1. Definizione del problema oggetto di analisi
2. Analisi esplorativa dei dati
3. Preparazione dei dati come input del modello
4. Sviluppo del modello
5. Valutazione del modello
6. Uso del modello e delle informazioni trovate

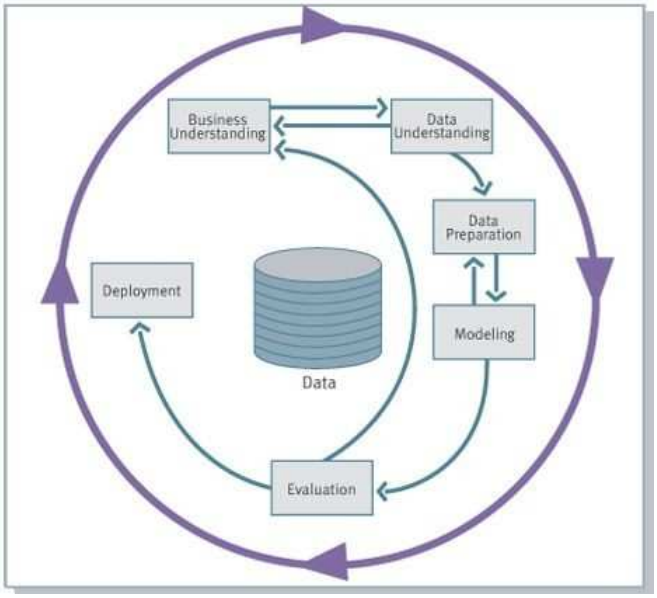

Figura 1.2: Le fasi definite dal CRISP-DM

Per sottolineare l'importanza di un approccio sistematico a prescindere dal tipo di prodotto di analisi o dalla bontà degli algoritmi a disposizione, vengono di seguito descritte le varie fasi del modello. L'esposizione non ha tuttavia l'obiettivo di riassumere il contenuto del documento, quanto riprendere i concetti più importanti di ciascuna fase, astraendoli dal contesto prettamente aziendale.

1. **Definizione del problema oggetto di analisi**: il prerequisito fondamentale per la ricerca di nuova conoscenza è la comprensione delle informazioni in possesso e del fenomeno che ha generato i dati stessi. Senza un'adeguata chiarezza del problema, nessun algoritmo per quanto sofisticato possa essere può restituire un buon risultato: il rischio di un'errata formulazione dell'obiettivo dell'analisi impedisce la scelta del modello corretto, genera leggerezza nella preparazione dei dati e difficoltà nell'interpretare i risultati finali. Ad esempio, se volessimo incrementare la risposta ad un sondaggio via mail, formulare l'obiettivo di incrementare il tasso di risposta piuttosto che incrementare il valore di una particolare risposta significa adottare due tipologie di modelli ben differenti. Infine la comprensione del problema di business permette la sua scomposizione in problemi più semplici da analizzare, modellare e spiegare.

2. **Analisi esplorativa dei dati**: in questa fase vengono dedicate attività mirate a "familiarizzare" con i dati di analisi. Gli specialisti del settore ritengono che il tempo e le risorse per la preparazione delle informazioni di input all'intero processo di data mining dovrebbero incidere dal 50% al 90% rispetto all'intera analisi. Questa fase inizia con la selezione delle fonti o database da cui estrarre i dati: l'idea è quella di disporre di un proprio dataset su cui poter lavorare liberamente. Sono diverse le motivazione che suffragano la scelta di creare un insieme di dati ad hoc: anzitutto il fatto che le informazioni possono provenire da più database differenti, alcuni applicativi di business (ad esempio le transazioni di carte di credito), altri esterni (dati disponibili on-line tramite Internet); anche qualora vi fosse la possibilità di utilizzare dati provenienti da un *data warehouse*, potrebbero insorgere problemi legati al pesante

carico computazionale per l'estrazione dei dati (interrogazioni SQL con selezione di dati da più tabelle), alla necessità di modificare dei valori o aggiungere nuove colonne. A seconda della numerosità dei dati è possibile utilizzare un semplice file o foglio di lavoro ed eventualmente, qualora la complessità lo richieda, un database.

Un altro vantaggio della creazione di un database indipendente è la fase di progettazione dello stesso: è consigliata una descrizione delle fonti, delle tabelle e della tipologia dei dati presenti in ogni colonna (unità di misura, l'eventuale lista di valori, ecc...).

La fase di popolamento del database permette il preventivo controllo della qualità dei dati: è possibile escludere valori errati e, qualora vi sia la necessità di integrare dati provenienti da diverse fonti, effettuare controlli di coerenza (lo stesso cliente può comparire con diversi indirizzi, con diversi identificativi o, peggio ancora, clienti associati allo stesso identificativo) e conversioni di quantità monetarie o di misura. Un altro importante vantaggio consiste nella gestione dei dati mancanti (*missing values*): sebbene le soluzioni siano molteplici, comunemente si risolve il problema mediante

- la creazione di una nuova variabile da inserire nel modello per segnalarne la presenza o l'assenza
- la sostituzione con un valore "medio"
- un valore che rispetti la distribuzione dei possibili valori (se nel database il 40% sono maschi e il 60% femmine, i valori mancanti saranno sostituiti rispettando tali percentuali)

Una volta costruito l'insieme dei dati di analisi, è necessario “familiarizzare” con i dati stessi in modo da identificare gli

attributi più importanti nella previsione e determinare le eventuali correlazioni tra valori. Gli strumenti più utili sono le informazioni di sintesi di ogni attributo (media, varianza, valore minimo e massimo, percentuale di valori nulli), l'analisi grafica mediante istogrammi, boxplot e diagrammi di relazione tra variabili. La visualizzazione dei dati facilita la percezione di relazioni, dipendenze, *outliers* (valori troppo diversi dalla distribuzione generale) e valori nulli rispetto ad una mera lista di numeri e stringhe letterali; tuttavia la rappresentazione grafica è limitata alle due dimensioni, visto che già la visualizzazione tridimensionale risulta molto meno intuitiva all'occhio umano.

Il data mining aggiunge altri due strumenti di tipo esplorativo: il clustering è focalizzato nella ricerca di attributi che rendano simili le osservazioni; l'associazione individua dipendenze e relazioni tra gli attributi.

Infine è bene ricordare che un'adeguata conoscenza dei dati limita l'uso di variabili *leaker*, ossia variabili surrogati rispetto a quella d'interesse.

3. **Preparazione dei dati come input del modello**: in tale fase si procede alla selezione delle variabili da utilizzare nel modello. E' necessario considerare che un numero elevato di variabili comporta maggior carico computazione e conseguentemente un tempo più lungo di elaborazione degli algoritmi. Si consiglia dunque di ponderarne l'uso in algoritmi che risentono di valori nulli ed eliminare variabili strettamente connesse (età e data di nascita).
4. **Sviluppo del modello**: tale fase è iterativa dal momento che il modello finale corrisponde a quello che risolve in maniera migliore il problema di analisi rispetto ad un certo numero di

alternative. Non sempre il modello migliore minimizza l'errore di previsione: molte volte modelli semplici sono molto più comprensibili rispetto ad altri più complessi, sebbene la scelta penalizzi l'adeguatezza della previsione. D'altra parte non bisogna confondere un buon modello con il vero modello: ad esempio un modello che spieghi la variazione dei prezzi senza considerare l'inflazione potrà risultare un modello valido finché tale componente resterà invariata.

In base al tipo di previsione è possibile scegliere tra diverse tecniche di analisi (adottare un albero decisionale piuttosto che una rete neurale), ciascuna delle quali può avere delle diverse implementazioni (un albero decisionale CART, C4.5, CHAID o particolari algoritmi proposti da un certo software di analisi): una visione più dettagliata sarà offerta nel paragrafo 2.1 relativamente alle tecniche e algoritmi di data mining.

5. **Valutazione del modello**: una volta costruito il modello, è necessario valutare l'effettiva bontà dei risultati ottenuti mediante l'utilizzo di particolati strumenti di misura atti a questo scopo.

   Nel caso della previsione di variabili continue (regressione) è intuitivo utilizzare lo scarto tra la previsione e il vero valore osservato: a tal proposito in ambito statistico sono stati sviluppati molti test per verificare la bontà di un modello, quali il confronto tra modelli diversi o annidiati e intervalli di confidenza sui parametri.

   Nel caso della previsione di variabili quantitative (classificazione), lo strumento maggiormente utilizzato è la curva di ROC (*Receiver Operating Characteristic*), utilizzata durante la II Guerra Mondiale per la verifica della bontà delle telecomunicazioni: si basa sul confronto tra il numero di

osservazioni previste in modo corretto ed errato. Più in dettaglio, la procedura consiste nel costruire una tabella a doppia entrata per il conteggio del risultato della classificazione relativamente a due possibili alternative, genericamente definite positiva e negativa: per tal motivo siffatta tabella è detta *matrice di errata classificazione* o *di confusione*. Le quantità n_{11} e n_{22} sono il totale di osservazioni (frequenze) previste con successo rispettivamente per l'esito negativo e positivo; le quantità n_{21} e n_{12} sono il totale di osservazioni previste erroneamente e dette rispettivamente *falsi negativi* (viene classificato come positiva un'osservazione negativa) e *falsi positivi* (viene classificato come negativa un'osservazione positiva). Le frequenze cumulate $n_{1.}$ e $n_{2.}$ sono relative al totale delle previsioni negative e positive, mentre le frequenze cumulate $n_{.1}$ e $n_{.2}$ sono calcolate rispetto alle osservazioni reali negative e positive. Intuitivamente una misura di bontà del modello consiste nel poter misurare la probabilità di classificazione di falsi positivi e di falsi negativi: a tal fine sono state introdotte le due misure

- *sensibilità*, definita come proporzione dei positivi previsti rispetto agli effettivi positivi, o più semplicemente 1-Probabilità{falso negativo};
- *specificità*, definita come proporzione dei negativi previsti rispetto agli effettivi negativi, ovvero la quantità 1-Probabilità{falso positivo}.

La curva ROC mette in relazione la sensibilità (ordinata) rispetto al complemento della specificità (ascissa): dal momento che tali probabilità non sono note, vengono stimate come

$$\text{sensibilità} \approx \frac{n_{22}}{n_{12}+n_{22}}, \qquad \text{specificità} \approx \frac{n_{11}}{n_{11}+n_{21}}$$

Di seguito viene presentata una curva ROC che si presenta lisciata mediante il raggruppamento dei dati in intervalli: da notare che la bontà della classificazione è tanto migliore quanto più la curva risulta essere lontana dalla bisettrice dell'angolo all'origine (corrisponde alla classificazione casuale delle osservazioni)

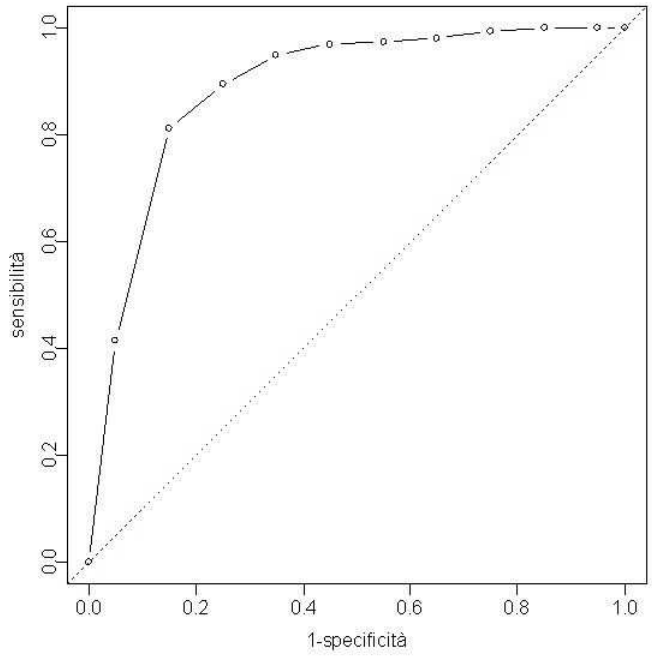

Figura 1.3: Curva di ROC

6. **Uso del modello e delle informazioni trovate**: se il modello ha il fine di produrre conoscenza sul fenomeno che ha generato i dati, il valore aggiunto apportato dall'applicazione del modello deve essere presentato in maniera tale da poter essere compreso ed utilizzato dal committente dell'analisi.

Di seguito viene riportata una schematizzazione della sequenza dei passi previsti dal CRISP-DM, proprio come formalizzato nella documentazione ufficiale del progetto [1.03]: ogni passo è ben definito e strutturato in una serie di precisi compiti da eseguire. Nella tabella seguente ogni colonna corrisponde ad una fase di analisi: le righe in grassetto identificano i vari compiti da eseguire e in corsivo i risultati

pratici da ottenere.

Conoscenza del Business	Conoscenza dei dati	Preparazione dei dati	Modellazione	Verifica	Realizzazione
Obiettivi di business *Contesto* *Obiettivi di business* *Criteri di valutazione rispetto al business* **Valutazione della situazione** *Analisi delle risorse* *Requisiti* *Assunzioni e vincoli* *Rischi e contingenze* *Costi e benefici* **Scopo del Data Mining** *Scopi dell'analisi* *Criteri per raggiungere gli obiettivi* **Creazione di un piano di progetto** *Piano di progetto* *Valutazione iniziale sugli strumenti e tecniche*	**Raccolta iniziale dei dati** *Raccolta iniziale dei dati* *Reportistica* **Descrizione dei dati** *Documento descrittivo dei dati* **Esplorazione dei dati** *Documento esplorativo dei dati* **Verifica qualità dei dati** *Documento di verifica della qualità dei dati*	*Creazione del dataset* *Descrizione del dataset* **Selezione dei dati** *Selezione dei dati da includere o escludere* **Pulizia dei dati** *Documento sulla pulizia dei dati* **Costruzione dei dati** *Selezione degli attributi* *Generazione dei dati come records* **Integrazione dei dati** *Integrare dati da fonti diverse* **Formattazione dei dati** *Riformattare i dati*	**Selezione della tecnica di modellazione** *Tecnica di modellazione* *Presupposti di modellazione* **Generazione del piano di test** *Piano di test* **Costruzione del modello** *Parametri di configurazione* *Modelli* *Descrizione dei modelli* **Verifica del modello** *Valutazione del modello* *Revisione dei parametri di configurazione*	**Risultati di verifica** *Valutazione dei dati di analisi rispetto ai criteri di valutazione preposti* *Modelli approvati* **Revisione del processo** *Revisione del processo* **Determinare i successivi passi di analisi** *Lista delle possibili azioni* *Decisione*	**Piano di realizzazione** *Deployment Plan* **Piano per il monitoraggio e il mantenimento** *Monitoring and Maintenance Plan* **Redazione del prospetto finale** *Prospetto finale* *Presentazione finale* **Revisione del progetto** *Experience Documentation*

Tabella 1.1: Compiti nelle varie fasi del modello CRISP-DM

Una volta fissata la sequenza di passi da seguire per intraprendere una corretta analisi, è necessario decidere come utilizzare gli strumenti di data mining: IBM ha identificato due tipi di modelli tipici del data mining, detti *Verification models* e *Discovery Models*.

I *Verification models* sono utilizzati per verificare le ipotesi formulate a priori dall'utente sulla base dei dati disponibili: in questo scenario di analisi il ruolo cruciale è rappresentato dall'analista a cui è affidato il compito di deduzione del modello dai dati.

I *Discovery Models* delegano al sistema il compito di individuare le informazioni importanti nascoste nei dati. Essi risultano di gran lunga i più interessanti dal momento che questo tipo di approccio è caratteristico del data mining.

1.1. Contesto

Le discipline scientifiche che hanno determinato e ancora oggi determinano l'evoluzione della teoria e dell'implementazione algoritmica del data mining sono essenzialmente l'informatica e la statistica: gli episodi di cooperazione tra i due mondi sono numerosi, come nel caso dello sviluppo e dell'applicazione degli alberi decisionali e delle reti neurali.

La statistica offre un significativo insieme di modelli e tecniche supportate da un solido fondamento teorico e metodologico analitico: tuttavia la quantità elevata di dati, la numerosa cardinalità delle variabili e l'assenza di un piano di campionamento che assicuri una certa distribuzione dei dati limitano il loro utilizzo.

L'informatica è un settore molto ampio e in continua crescita: i campi di maggior interesse sono l'organizzazione e la gestione delle basi di dati (nella loro implementazione classica e come data warehouse) e l'Intelligenza Artificiale. Di quest'ultima disciplina è molto seguita l'area del Machine Learning che si occupa della realizzazione di sistemi basati su osservazioni o esempi per la sintesi di nuova conoscenza (classificazioni, generalizzazioni, riformulazioni). L'obiettivo è rispondere a quelle situazioni di difficile soluzione mediante algoritmi tradizionali, tipicamente dovute alla presenza di uno o più dei seguenti fattori:

- difficoltà di formalizzazione;
- elevato numero di variabili;
- mancanza di teoria: per esempio non esistono leggi

matematiche note che regolino con esattezza l'andamento dei mercati finanziari;

- necessità di personalizzazione: ad esempio è intuitivo supporre che la catalogazione di documenti sulla base del loro contenuto debba dipendere dal soggetto stesso che esegue l'operazione.

Il data mining eredita la suddivisione classica utilizzata nel machine learning per suddividere gli approcci con cui operano i vari algoritmi: essi sono detti *supervised learning* o *supervised learning*.

L'apprendimento supervisionato (supervised learning) consiste nel definire i dati di input e di output, lasciando al sistema il compito di individuare la funzione che associa ad ogni dato in ingresso la sua risposta corretta. L'obiettivo è la ricerca di un'approssimazione della funzione che ha generato i dati di esempio di cui si conosce il risultato (output): sebbene questo non sia reale, esistono molte condizioni in cui questa semplificazione è accettabile. E' facilmente intuibile che il buon funzionamento di questi algoritmi dipende in modo significativo dai dati in ingresso: se con pochi dati l'algoritmo potrebbe non aver abbastanza esperienza, viceversa con molti dati potrebbero risultare troppo legato ai dati in ingresso (*overfitting*) e lento a causa della funzione eccessivamente complessa.

L'apprendimento non supervisionato (unsupervised learning) cerca di estrarre conoscenza in modo automatico dai dati, senza alcuna informazione esterna sulla struttura o contenuto dei dati stessi: l'approccio di questi algoritmi si basa sul confronto dei dati, sulla ricerca di similarità o differenze secondo particolari metriche. Ad esempio i motori di ricerca, comunemente usati in Internet, sono in grado di creare una lista di riferimenti alle pagine web più attinenti e interessanti sulla base dei criteri di ricerca impostati sul browser. Tali algoritmi sono molto efficienti con elementi di tipo numerico a causa dell'ordinamento intrinseco sfruttato da molte tecniche statistiche;

altresì sono meno efficienti o addirittura fallimentari per dati di tipo discreto privi di una qualche forma di ordinamento implicito.

1.2. Tecniche e algoritmi

Il data mining offre numerosi strumenti per l'analisi dei dati: in questo paragrafo verranno descritte le tecniche e presentati gli algoritmi più importanti che implementano ciascuna tecnica. Tuttavia l'esposizione non vuole essere esaustiva: la letteratura specialistica propone numerosi algoritmi, spesso risultato di combinazioni di più tecniche diverse o varianti di algoritmi più noti al fine di ottimizzare determinati aspetti critici presenti nella formulazione originale.

Prima di iniziare, è utile definire le tipologie di problemi a cui il data mining risponde: le prime due soluzioni appartengono all'insieme delle tecniche denominate supervised learning.

- **classificazione**: è l'operazione che determina l'appartenenza di un'osservazione ad una classe sulla base dei valori degli attributi che caratterizzano l'osservazione stessa. L'approccio consiste nell'etichettare ciascuna osservazione con un certo valore discreto tra un insieme ben definito di possibilità: data una nuova osservazione il classificatore (ossia il modello costruito con tale tecnica) indicherà l'etichetta corretta sulla base dei valori degli attributi. La costruzione del classificatore si basa su un insieme di osservazioni di stima o training, di cui è già noto il valore dell'etichetta: a partire da tali esempi, il modello apprende le regole implicite che determinano l'appartenenza ad una certa classe.
- **regressione**: simile alla classificazione, la regressione è specializzata nella previsione di variabili quantitative. Molti risultati provengono dalla statistica, ambito in cui il problema della regressione è stato ampiamente studiato: il concetto base

su cui poggia l'intera teoria è la possibilità di approssimare la vera funzione che ha generato le osservazioni. La previsione è verificata sugli scarti d'errore generati dal modello rispetto ai veri valori.

Seguono altre due soluzioni appartenenti all'area unsupervised learning. In questo ambito non è richiesto alcunché dall'utente: il sistema provvede automaticamente all'estrazione della conoscenza.

- **associazione**: è basata sulla ricerca di relazioni e dipendenze tra determinati attributi delle osservazioni. L'obiettivo non consiste nel prevedere un certo valore ma informare l'utente della presenza di particolari affinità tra determinati attributi, valide per un sottoinsieme significativo dei dati disponibili. Spesso l'obiettivo dell'analisi consiste nel ricercare relazioni causa/effetto: in ambito medico ad esempio vengono usate tali tecniche per definire delle regole con cui assegnare la priorità di pronto soccorso ai pazienti sulla base delle loro caratteristiche fisiche e sintomatiche. Tuttavia l'applicazione più importante di tale tecnica è senza dubbio l'analisi del paniera della spesa (*market basket analisys*), il cui scopo è la ricerca di associazioni tra i prodotti acquistati dai clienti in un supermercato: la scoperta di tali relazioni diventa strategica all'interno del supermercato stesso e si concretizza ad esempio nella pianificazione di campagne promozionali o nella particolare disposizione dei reparti merceologici. A tal proposito, uno dei primi risultati ottenuti con tale tecnica in America fu la scoperta che l'acquisto dei pannolini per neonati era strettamente connesso all'acquisto di birra, fenomeno dovuto alla diffusa consuetudine delle madri americane di lasciare al proprio patner il compito di fare la spesa durante i primi mesi di vita del figlio.

  Spesso la letteratura si riferisce a tale approccio come ricerca

dei cosiddetti *sequential/temporal patterns*: il primo investiga su relazioni di tipo sequenziale, ossia una successione di particolari valori. Ad esempio per identificare frodi causate dalla clonazione di carte di credito o Bancomat vengono controllati i tipi di acquisti effettuati: una carta utilizzata principalmente per l'acquisto di carburante è quasi sicuramente oggetto di frode qualora si registri un acquisto di gioielli. Il secondo si concentra sulle relazioni di tipo temporale, come successione di eventi: tipicamente è il campo di ricerca delle serie storiche.

- **segmentazione**: detta anche clusterizzazione, essa mira alla separazione delle osservazioni in gruppi (clusters) o classi significative, in cui tutti i membri hanno caratteristiche simili. A differenza della classificazione, i gruppi non sono noti a priori: inoltre, basandosi su un apprendimento non supervisionato, l'utente non deve fornire nessun esempio precedentemente classificato. La peculiarità di tale approccio consiste nel lasciare al sistema la scoperta delle caratteristiche che accomunano le osservazioni, basandosi su particolari criteri di similarità e prossimità, dette metriche. Oggigiorno molte applicazioni utilizzano la tecnica della segmentazione: l'analisi del paniera della spesa (*market basket analisys*) permette la profilatura della clientela mediante l'utilizzo di carte fedeltà, l'analisi della vulnerabilità dei clienti (*risk analisys*) in ambito assicurativo o bancario per individuare i soggetti tendenzialmente inadempienti, la gestione del portafoglio finanziario, l'individuazione delle truffe (*fraud detection*) in contesti come la telefonia cellulare e le carte di credito.

1.2.1. Regressione lineare

La regressione è la più classica delle tecniche per la previsione di

valori continui: ampiamente studiata in ambito statistico, la regressione si presenta come una soluzione molto intuitiva anche nelle formulazioni più complesse. La trattazione in questo paragrafo non vuole essere esaustiva ma mira a dare una panoramica sui risultati più importanti: per una corretta esposizione, viene prima esposto l'approccio parametro e successivamente quello non parametrico.

Modello lineare

Il modello lineare studia le relazioni tra la variabile risposta y continua e le variabili esplicative $x_1,x_2,...,x_p$ con $p \geq 1$: la formulazione più generale è

$$\begin{aligned} y &= f(X;\beta)+\epsilon \\ &= X\beta+\epsilon \end{aligned}$$

dove y è vettore (n x 1) delle variabili risposta, X è matrice (n x p) di regressione, β è vettore (p x 1) dei parametri di regressione e ε è il vettore (n x 1) delle componenti della variabile d'errore. La suddetta formulazione sottolinea il fatto che non esiste alcuna restrizione sulla parametrizzazione se non il fatto di essere lineare nei parametri.

La procedura inferenziale si basa sulle seguenti condizioni: la matrice X è non stocastica a rango pieno p mentre la componente stocastica/casuale si suppone a media nulla e varianza costante pari a σ^2: la stima di β utilizza il criterio dei minimi quadrati per individuare i valori di β per cui è minima la funzione obiettivo

$$D(\beta)=\sum_{i=1}^{n}[y_i-f(x_i;\beta)]^2=\|y-f(x;\beta)\|$$

La stima di β si ottiene mediante

$$\hat{\beta}=(X^TX)^{-1}X^Ty \quad \text{con} \quad E(\hat{\beta})=\beta$$

La stima di σ^2 è calcolata come segue:

$$S^2=\frac{D(\hat{\beta})}{n-p}=\frac{1}{n-p}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2=\frac{(y-X\hat{\beta})^T(y-X\hat{\beta})}{n-p}$$

dove la quantità a numeratore è detta *devianza*, ossia somma dei quadrati dei residui (SQE). Siccome il modello lineare si scompone in parte sistematica ($X\beta$) e parte casuale (ε), una volta stimati i parametri di regressione è possibile suddividere la devianza totale come somma tra la devianza di regressione e la devianza dei residui, ossia SQT=SQR+SQE.

Qualora si aggiunga l'ipotesi che il termine di errore ε abbia distribuzione normale (ipotesi del II° ordine), le stime ottenute godono delle seguenti approssimazioni:

$$\hat{\beta} \sim N_p\left(\beta, \sigma^2 (X^T X)^{-1}\right)$$
$$\frac{S^2(n-p)}{\sigma^2} \sim \chi^2_{n-p}$$

Sulla base delle suddette approssimazioni sui parametri, è possibile condurre i seguenti test:

- Analisi della varianza: viene confrontata la perdita di bontà del modello con l'aggiunta di $p>p_0$ parametri sulla base della componente d'errore

$$F = \frac{(SQE_{p_0} - SQE_p)/(p-p_0)}{SQE_p/(n-p)} \sim F_{p-p_0, n-p}$$

- Test su modelli annidiati: è un risultato dell'inferenza statistica basata sul concetto di verosimiglianza. Posto $\beta = (\beta_{MR}, \beta_{MC})$, si saggia l'ipotesi $\beta_{MC} = 0$ mediante il test di Wald costruito sul valore della log-verosimiglianza associata ai due modelli

$$W = 2\{l(\hat{\beta}) - l(\hat{\beta_{MR}})\} \sim \chi^2_{p-p_0}$$

  dove p sono i parametri di β e p_0 quelli di β_{MR}.

GLM (Generalized Linear Models)

I modelli lineari generalizzati estendono i modelli lineari classici superando i vincoli restrittivi sulla distruzione normale degli errori, pur mantenendo un'analoga struttura di costruzione e di analisi: tale

approccio apre alla possibilità di includere modelli con funzioni non lineari ed errori non normali. Un modello GLM è caratterizzato da:

- **componente casuale**: y appartiene ad una famiglia esponenziale, ossia ha funzione di densità o probabilità del tipo

$$f(Y;\theta,\phi)=\exp\{\frac{Y\theta-b(\theta)}{\phi}+c(Y,\phi)\}$$
$$\Rightarrow Y_i \sim EF(b(\theta_i),\phi) \qquad \text{con} \qquad E(Y_i)=b'(\theta_i)=\mu_i$$
$$Var(Y_i)=\phi b''(\theta_i)$$

- **componente sistematica**: viene definita come relazione lineare $\eta_i = x_i^{\mathrm{T}}\beta$, detta *predittore*.

- **funzione legame**: una funzione $g \mid g(\mu_i) = \eta_i$ e $\mu_i = g^{-1}(\eta_i)$ per ogni i=1,2,..., n

Alla definizione di modello GLM viene usualmente aggiunta la definisce della funzione di varianza, $V(\mu) = b''(\theta)$.

Come approccio alle procedure di inferenza viene usata la massima verosimiglianza:

$$l(\beta)=\sum_{i=1}^{n}\{\frac{Y_i\theta_i-b(\theta_i)}{\phi}+c(Y_i,\phi)\}=\sum_{i=1}^{n} l_i(\beta)$$

poiché in generale le equazioni di massima verosimiglianza per β non ammettono una soluzione esplicita, la stima di β è ottenuta mediante l'applicazione del cosiddetto algoritmo dei minimi quadrati pesati iterati. Dalla scrittura di $Var(Y_i)$ è intuitivo stimare l'altro parametro incognito come

$$\hat{\phi}=\frac{1}{n-p}\sum_{i=1}^{n}\frac{(Y_i-\hat{\mu}_i)^2}{V(\hat{\mu}_i)} \quad dove\ \hat{\mu}_i=g^{-1}(x_i^T\hat{\beta})$$

Per costruzione, è immediato verificare che un'approssimazione del parametro β è data da

$$\hat{\beta}\sim N_p\left(\beta,\phi\left(X^T\hat{W}X\right)^{-1}\right) \quad con \quad W=\mathrm{diag}(w_{1,\dots},w_n),\ w_j=\frac{1}{V(\mu_i)(g'(\mu_i))^2}$$

E' possibile definire un intervallo di confidenza per il parametro β_j definito da

$$\hat{\beta}_j \pm z_{1-\frac{\alpha}{2}} \sqrt{\phi [X^T \hat{W} X]^{-1}_{j,j}}$$

Il test di Wald per la verifica di modelli annidiati segue la costruzione e i risultati dei modelli lineari. Inoltre esso suggerisce l'utilizzo della log-verosimiglianza per la stima della cosiddetta devianza associata al modello, ossia la differenza della log-verosimiglianza tra il modello saturo e un generico sotto-modello. Usando una notazione matematica, la funzione di devianza è definita come

$$D(y;\hat{\theta}) = 2\phi\{l(\tilde{\theta}) - l(\hat{\theta})\} = \phi \sum_{i=1}^{n} D_i$$

con

$$D_i = 2\{y_i(\tilde{\theta}_i - \hat{\theta}_i) - b(\tilde{\theta}_i) + b(\hat{\theta}_i)\}$$

Nell'impianto teorico dei GLM la devianza è molto importante: essa rappresenta una misura del tutto analoga alla somma dei quadrati dei residui (SQE) dei modelli lineari.

Modello logistico

Caso particolare di GLM, il modello logistico fornisce la corretta impostazione da seguire nel caso in cui si debba prevedere una variabile dicotomica: essa mette in relazione la probabilità π dell'evento di interesse con un insieme di variabili esplicative $x=(x_1,x_2,\ldots,x_p)$

Riprendendo le notazioni dei GLM, $\eta(x)$ rappresenta una qualsiasi combinazione di variabili esplicative lineari nei parametri, Y si suppone abbia distribuzione binomiale $Y \sim \text{Bi}(1,\pi)$ e funzione legame *logit*

$$g(\pi) = \text{logit}(\pi) = \log \frac{\pi}{(1-\pi)}$$

Naturalmente è possibile estendere il caso in cui la variabile risposta assuma m valori modificando la distribuzione di Y: in tale scenario $Y \sim \text{Bi}(m,\pi)$.

1.2.2. Regressione locale

Nell'ambito della previsione di variabili quantitative, la tecnica della regressione locale utilizza un semplice approccio non parametrico. Rispetto all'usuale impostazione di regressione, per cui

$$y = f(x) + \epsilon \quad \text{con } E(\epsilon) = 0$$

viene aggiunta la condizione che la funzione $f(x)$ sia derivabile con derivata continua in ogni punto x. Sotto questa ulteriore ipotesi è possibile sfruttare lo sviluppo in serie di Taylor per poter approssimare la funzione in un generico punto x_0 con una retta passante per $(x_0, f(x_0))$. Per la stima dei parametri viene usualmente applicato il criterio dei minimi quadrati pesati, in modo da poter attribuire maggior importanza ai punti più vicini a x_0 . Inoltre è possibile definire una funzione di stima per ogni generico x, ossia

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{[a_2(x;h) - a_1(x;h)(x_i - x)] w_i y_i}{a_2(x;h) a_0(x;h) - a_1(x;h)^2}$$

dove $a_r(x;h) = \frac{1}{n} \sum_{i=1}^{n} (x_i - x)^r w_i$ con r = 0,1,2 mentre le quantità w_i costituiscono i pesi associati ad ogni x. In particolare è possibile esprimere i pesi mediante la notazione matematica

$$w_i = \frac{1}{h} w\left(\frac{x_i - x_0}{h} \right)$$

che evidenzia come essi siano composti da una funzione w di densità simmetrica attorno all'origine (*nucleo*) e un fattore di scala h (*ampiezza di banda* o *parametro di lisciamento*). Se da una parte la scelta della funzione nucleo è indifferente in un certo insieme (nucleo normale, rettangolare, Epanechnikov, biquadratico e tricubico), il

valore del parametro di lisciamento determina l'adattamento del modello di stima rispetto ai dati: tanto più h è piccolo quanto più la funzione si avvicina alle osservazioni di stima; viceversa tanto più h è grande, quanto più la funzione è liscia. Interessanti considerazioni sono emerse nello studio sull'approssimazione della media e varianza della funzione di stima: la distorsione è un multiplo di h^2, mentre la varianza è un multiplo di $1/(nh)$. Dunque è necessario individuare una soluzione di compromesso tra distorsione e varianza al fine di non far degenerare una delle due quantità.

Loess

E' una tecnica che si basa sulla regressione locale, aggiungendo delle ottimizzazioni particolari. Anzitutto il parametro di lisciamento non è costante ma è in funzione del grado di sparsità dei punti x_i. Infine il loess è stato studiato per offrire una stima robusta: a tal fine è stato scelto come nucleo la funzione tricubica.

1.2.3. Spline

Tale tecnica si basa sull'idea di dividere il dominio della variabile continua di risposta in sotto-aree in cui le osservazioni siano interpolate da funzioni polinomiali: il termine *spline* è usato in matematica proprio per approssimare funzioni di cui si conoscono solo alcuni punti.

Più precisamente tra due punti ξ_i e ξ_{i+1} (detti *nodi*), la funzione $f(x)$ viene approssimata da un polinomio di grado d con il vincolo di congiungersi bene ai polinomi vicini, di cui ξ_i e ξ_{i+1} sono rispettivamente il punto iniziale e finale. Tale requisito garantisce che la successione dei polinomi dia effettivamente l'impressione di una funzione continua: operativamente si pone che la $f(x)$ abbia derivate dal grado 0 al grado d-1 continue in ciascun punto di suddivisione ξ_i, con i=1,2,...,K. Quasi universalmente d = 3: si parla quindi di *spline*

cubiche. Per poter identificare univocamente la funzione si aggiungono altri due vincoli, generalmente supponendo che le derivate seconde dei polinomi nei nodi estremi ξ_1 e ξ_K siano nulle: spline siffatte sono dette *spline cubiche naturali*.

Tale approccio può essere utilizzato nei due contesti presentati di seguito.

- Stima parametrica: applicando le spline al caso di regressione classico $y = f(x; \beta)+\varepsilon$ mediante la selezione di K ascisse, β rappresenta i parametri non vincolati dei K+1 polinomi che interpolano i nodi prescelti. Nel caso particolare di spline cubiche, β contiene K+4 componenti da stimare applicando il criterio dei minimi quadrati. La soluzione è costituita da una opportuna combinazione lineare di una base di funzioni $\{h_j(x), j=1,...,K+4\}$, costituita in parte da polinomi elementari e parte da funzioni del tipo $\max(0, (x-\xi)^3)$.
- Stima non parametrica: in tal caso l'approccio da seguire parte dal criterio dei minimi quadrati penalizzati, la cui forma algebrica è

$$D(f,\lambda)=\sum_{i=1}^{n}[y_i-f(x_i)]^2+\lambda\int[f''(t)]^2\,dt$$

con λ parametro positivo di penalizzazione del grado di irregolarità della curva f, quantificato dall'integrale di $f''(x)^2$. La soluzione al problema di minimizzazione è costituita da una funzione di tipo spline cubica naturale i cui nodi sono i punti x_i distinti ossia

$$\hat{f}(x)=\sum_{j=1}^{n_0}\theta_j N_j(x)$$

dove n_0 è il numero dei x_i distinti e gli $N_j(x)$ sono le basi delle spline cubiche naturali.

MARS

Una particolare variante delle spline di regressione ottimizzata per la gestione di molte variabili esplicative sono le spline di regressione multidimensionali adattative (*Multivariate Adaptive Regression Splines*).

Definite basi le coppie di funzioni lineari a tratti del tipo $(x\text{-}\xi)_+$ e $(\xi\text{-}x)_+$ con un solo nodo ξ, per ogni variabile esplicativa x_i (i=1,2,...,p) si determina una coppia di basi con il nodo in ciascun valore osservato su quella variabile x_{ij} con j=1,2,...,p. Il modello MARS è del tipo

$$f(x)=\beta_0+\sum_{k=1}^{K}\beta_k h_k(x)$$

dove $h_k(x)$ sono le funzioni appartenenti all'insieme di basi di funzioni (valide per tutte le possibili combinazioni delle variabili esplicative) o come prodotti di due o più di tali funzioni. Una volta scelte le $h_k(x)$ e il loro numero K, i parametri β_k sono stimati minimizzando la somma dei residui: a tal proposito viene utilizzato un procedimento ricorsivo.

Tale tecnica ha il pregio di gestire anche le variabili qualitative: infatti introducendo tante variabili esplicative binarie quante le modalità di ciascuna variabile esplicativa quantitativa, ciascuna di esse genera una coppia di basi di funzioni costanti a tratti che indicano l'appartenenza ad una particolare modalità.

1.2.4. GAM

Il modello additivo è una tecnica formulata per rispondere al meglio ai problemi di stima derivati da un numero molto grande di variabili esplicative (la cosiddetta maledizione della dimensionalità) e alla necessità di avere uno strumento quanto più semplice possibile. La formulazione matematica per $f(x)$ è

$$f(x_1, x_2, \ldots, x_p)=\alpha+\sum_{j=1}^{p} f_j(x_j)$$

supponendo f_1, f_2, ..., f_p funzioni in una variabile dall'andamento sufficientemente regolare e centrate sullo 0; α è una costante. Le funzioni f_1, f_2, ..., f_p sono stimate sulla base di un metodo non parametrico che costituisce una variante dell'algoritmo di Gauss-Seidel, detto *backfitting*.

Dal momento che un modello additivo non riesce a cogliere le interazioni tra variabili esplicative, si introduce la generalizzazione del modello, detto appunto GAM (*Generalized Additive Model*), sulla falsa riga dei modelli GLM per i modelli lineari: tale formulazione è espressa come

$$g(\mathrm{E}[Y|x_1,\ldots,x_p])=\alpha+\sum_{j=1}^{p} f_j(x_j)$$

Non essendo più necessariamente lineari, le funzioni di un modello GAM sono stimate mediante un'opportuna combinazione tra l'algoritmo di backfitting e quello dei minimi quadrati pesati iterati utilizzato per i GLM.

1.2.5. Classificatori basati sul teorema di Bayes

La formulazione corretta per impostare il problema della classificazione consiste nel considerare una variabile categoriale che rappresenti la classe di appartenenza dell'osservazione e una variabile p-dimensionale X come l'insieme di attributi dell'osservazione stessa. Dette K le possibili classi, si vuole determinare sulla base di una certa combinazione x_0 di X quale sia la classe C_k di appartenenza dell'osservazione: intuitivamente in corrispondenza della massima probabilità della generica classe C_j rispetto a x_0

$$P(C_i|x_0)>P(C_j|x_0) \quad \text{per ogni } i\neq j$$

Sfruttando il teorema di Bayes per cui la probabilità a posteriori di un certo evento casuale A condizionatamente all'evento B è calcolabile sulla base della probabilità a priori di A e B e la probabilità di B

condizionatamente ad A, possiamo scrivere

$$P(C_k \mid x_0) = \frac{P(x_0 \mid C_k) P(C_k)}{P(x_0)}$$

Siccome l'obiettivo consiste nell'individuare la classe C_k che massimizza la probabilità $P(C_k|x_0)$, l'analisi è focalizzata nella massimizzazione della quantità a numeratore. Se da una parte la quantità $P(C_k)$ è naturalmente approssimabile mediante il rapporto tra il totale delle osservazioni appartenenti alla classe *k*-esima e la numerosità dell'intero campione, dall'altra la probabilità $P(x_0|C_k)$ risulta difficilmente stimabile: infatti il campione dei dati dovrebbe contenere tutte le possibili combinazione dei valori degli attributi $x_1,x_2....,x_p$ e in una quantità tale da poter giustificare una approssimazione basata sui dati.

Spesso il confronto tra le varie classi avviene attraverso il calcolo della cosiddetta funzione discriminante, definita come

$$d_k(x_0) = logP(C_k) + logP(x_0 \mid C_k)$$

Sono stati proposti numerosi criteri di stima, sia mediante approccio parametrico che non parametrico: nel proseguo della relazione verranno affrontati solo le soluzioni più interessanti in campo parametrico, visto che l'altro approccio non ha di fatto trovato larga applicazione a causa dei limiti nella trattazione di combinazioni di variabili continue e discrete e di osservazioni con molti attributi.

Classificatore Naïve Bayes

Una prima soluzione al problema di stima della probabilità di un certo insieme di attributi *X* rispetto alla classe di appartenenza consiste nell'aggiungere l'ipotesi semplificativa detta *Naīve Bayes*, secondo cui tutti gli attributi osservati $x_1,x_2....,x_p$ sono indipendenti: dunque la probabilità condizionata può essere scritta come

$$P(x_1, x_2, ... x_p \mid C_k) = P(x_1 \mid C_k) P(x_2 \mid C_k) ... P(x_p \mid C_k)$$

Questo permette di stimare la generica probabilità $P(x_j|C_k)$ basandosi sui dati disponibili: per gli attributi discreti la stima coincide con il rapporto tra le osservazioni aventi un certo valore di x_j appartenenti alla classe C_k rispetto al totale delle osservazioni appartenenti alla sub-popolazione C_k.

Infatti, sfruttando le basi del calcolo di probabilità secondo cui

$$\sum_{k=1}^{K} P(C_k | x_1, x_2, \ldots x_p) = 1$$

è possibile calcolare $P(x_1, x_2, \ldots, x_p|C_k)$ mediante un sistema di K+1 equazioni: posto $p_k = P(C_k|x_1,x_2,\ldots,x_p)$, $z = P(x_1,x_2,\ldots, x_p)$ e $s_k = P(x_1,x_2,\ldots,x_p|C_k)\mathrm{P}(C_k)$, il suddetto sistema è composto dalle seguenti relazioni

$$\sum_{k=1}^{K} p_k = 1$$
$$p_j z = s_j \quad \text{per ogni } j = 1,2,\ldots,K$$

E' immediato verificare che la soluzione è

$$p_j = \frac{s_j}{\sum_{k=1}^{K} s_k} \quad \text{ossia} \quad P(C_j|X) = \frac{P(X|C_j)P(C_j)}{\sum_{k=1}^{K} P(X|C_k)P(C_k)}$$

Analisi discriminante

Nel caso di variabili continue si assume che l'attributo abbia una distribuzione normale multivariata $N_p(\mu_k,\Sigma_k)$ con media e varianza calcolata sulla base delle osservazioni della sub-popolazione C_k. Nel caso in cui tutte le matrici di varianza siano uguali ad una certa matrice Σ, la funzione discriminante diventa una funzione lineare in x, da cui il nome di *analisi discriminante lineare* (LDA):

$$d_k(x) = logP(C_i) - \frac{1}{2}\mu_k^T \Sigma^{-1} \mu_k + x^T \Sigma^{-1} \mu_k$$

Nel generico caso in cui non vi sia nessun vincolo sulle matrici di varianza Σ_k, otteniamo che la funzione discriminante diventa una

funzione quadratica in x

$$d_k(x) = logP(C_i) - \frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k) - \frac{1}{2}\log det[\Sigma_k]$$

da cui il nome della procedura *analisi discriminante quadratica* (QDA).

In conclusione si sottolinea come questa tecnica di classificazione generi risultati molto competitivi anche rispetto ad algoritmi più evoluti: tuttavia qualora l'assunto di indipendenza sia violato, la bontà della previsione degrada rapidamente. Ciò nonostante, l'impianto teorico è talmente condiviso da vantare in letteratura numerosi casi di tentativi in cui il naïve Bayes è stato combinato con altre tecniche al fine di incrementare i risultati laddove la dipendenza tra attributi è più marcata. Alcuni esempi sono il classificatore semi-bayesiano (Kononenko, 1991), le reti bayesiane (Friedman e Goldszmidt, 1996; Sahami, 1996; Singh e Provan, 1995), il classificatore ricorsivo bayesiano (RBC) (Langley, 1993) e gli alberi naïve Bayes (NBTree) (Kohavi, 1996).

1.2.6. Alberi

Tale tecnica è nata nella teoria delle decisioni: la selezione tra possibili alternative è valutata sulla massimizzazione del valore restituito da una funzione di rischio in corrispondenza di ciascuna scelta. Nel data mining, la sua applicazione naturale è relativa ai problemi di classificazione: nel caso in cui la classificazione interessi variabili discrete, l'albero è detto *decisionale*; nel caso di variabili continue è detto di *regressione*.

L'idea consiste nel suddividere (*splitting*) l'insieme dei dati in partizioni dove le osservazioni abbiano in determinati attributi valori simili: i punti di divisione dell'albero sono detti *nodi* e corrispondono ad affermazioni di tipo logico sui valori assunti da uno o più attributi dell'osservazione. Il primo nodo è detto *radice*: in base all'esito della

proposizione logica del nodo si scende ai nodi sottostanti, dalla parte sinistra se l'esito è positivo o dalla parte destra se l'esito è negativo; i nodi finali sono detti *foglie* e contengono la funzione approssimante della classe di appartenenza.

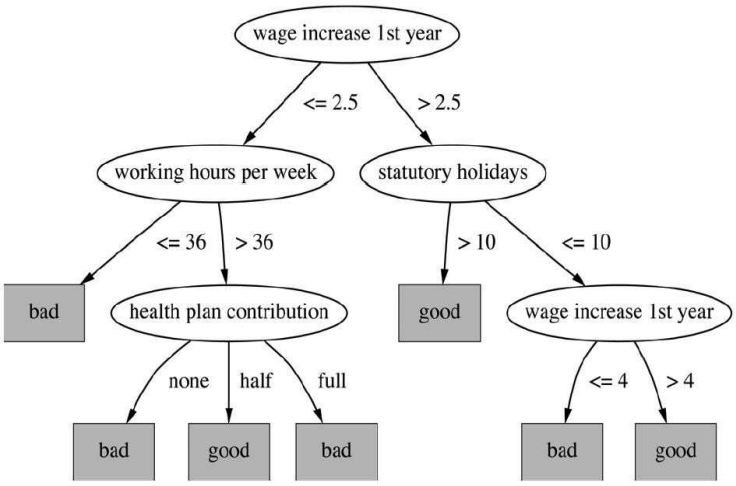

Figura 1.4: esempio di albero

La costruzione di un albero si articola nei seguenti punti:

- **Selezione delle variabili di splitting**: la suddivisione dell'insieme dei dati mira all'individualizzazione delle variabili che spiegano maggiormente le osservazioni: un desiderata della tecnica consiste nel situare le variabili più importanti nei nodi più vicini alla radice. Questo significa che ad ogni nodo l'algoritmo deve cercare quale attributo produce la suddivisione migliore rispetto alle osservazioni disponibili in quel particolare nodo, ossia rispetto all'insieme dei dati che verificano le condizioni sui nodi padri precedenti.
- **Numero di ramificazioni (branch)**: ogni nodo padre può generare due o più nodi figli: la maggior parte degli algoritmi operano una suddivisione binaria in modo da non ridurre pesantemente la numerosità di osservazioni in ciascun nodo figlio (cosa che comprometterebbe i risultati statistici ottenuti); d'altra parte la suddivisione in più nodi figli può essere

riprodotta mediante una sequenza di *split* binari.

- **Valutazione della migliore suddivisione**: l'individualizzazione dell'attributo e del relativo valore soglia che comporta la migliore suddivisione è basata sulla massimizzazione di qualche misura di separazione delle osservazioni del nodo corrente. Non necessariamente però tale criterio comporta la selezione ottimale in senso assoluto, ma spesso individua la migliore solo localmente: in tal senso è detto miopico.
- **Tipologia di partizione**: la maggior parte degli algoritmi operano una partizione ricorsiva nel senso che ciascuna operazione di suddivisione viene eseguita con interesse locale, senza badare alle suddivisioni future. D'altronde la ricerca della suddivisione migliore in termini assoluti comporta un maggior costo computazionale, dal momento che si dovrebbero generare tutti i possibili sotto-alberi seguenti prima di confermare ciascuna partizione.
- **Termine della procedura di crescita**: la crescita, ossia la successione delle operazioni di suddivisione, deve necessariamente terminare prima di conformare troppo l'albero rispetto ai dati di stima: generalmente vengono usate regole basate sul numero minimo di foglie, sul numero minimo di osservazioni in ciascuna foglia o sulla profondità dell'albero. Questa fase è molto delicata: infatti se un albero poco sviluppato può incorrere in errori dovuti alla mancanza di addestramento (*training*), quello troppo "folto" rischia di generare errori dovuti al modello previsivo eccessivamente conformato ai dati di stima.
- **Potatura**: la quasi totalità degli algoritmi di ultima generazione prevedono una fase successiva alla creazione dell'albero, detta di *potatura*. Tale fase consiste nel ripercorrere l'albero generato

e ottimizzare la sua dimensione in base a diversi criteri, quali

1. l'utilizzo di un insieme di dati di verifica su cui testare il modello creato;
2. l'utilizzo di un test statistico usando i dati stessi di stima, al fine di stabilire se tenere o meno un nodo;
3. l'utilizzo di una funzione costo-complessità associata all'albero.

- **Valori mancanti**: la gestione del problema dei dati mancanti è sempre molto importante specialmente in quegli insiemi di dati in cui il tasso di presenza è molto alto. A seconda degli algoritmi le soluzioni implementate possono essere

    1. ignorare le osservazioni, talvolta con una pesante ripercussione sulla numerosità campionaria;
    2. creazione di una classe apposita per valori mancanti, sebbene possano essere controproducenti quando la numerosità del nuovo fattore influenzi le procedure di splitting;
    3. utilizzo di variabili surrogate a quella di interesse: ad esempio utilizzare la variabile "regione" qualora manchi il valore della variabile "città"
    4. associare i valori mancanti in ciascun nodo rispettando la proporzione delle osservazioni associate ai nodi della stessa profondità.

Di seguito vengono proposti gli algoritmi più importanti che implementano la tecnica degli alberi.

AID (Automatic Interaction Detection)

James Morgan e John Sonquist pubblicarono la loro metodologia per la costruzione di un albero decisionale in un articolo del *Journal of the*

American Statistical Association nel 1963: nacque il programma AID (seguito da SEARCH nel 1971) per la classificazione di variabili discrete sulla base di variabili discrete e quantitative. L'idea è quella di suddividere passo passo l'insieme di stima in gruppi: ciascun gruppo viene suddiviso sulla base dell'attributo che comporta il minor valore della somma dei quadrati all'interno delle nuove suddivisioni. Nel caso l'attributo sia quantitativo, per ogni valore viene calcolata la somma dei quadrati rispetto alla media del gruppo, al fine di trovare il valore che rappresenta nel modo migliore il contributo della variabile stessa; nel caso di un attributo discreto con k categorie, viene effettuato un confronto tra tutte le 2^k-1 possibilità. Questo aspetto spiega il termine "interaction" presente nel nome AID: infatti ciascuna suddivisione incorpora naturalmente l'interazione tra gli attributi. La procedura termina quando la riduzione della somma dei quadrati è inferiore ad una certa soglia (default 0.006) rispetto a tutti gli attributi delle osservazioni. I valori mancanti sono esclusi dalla stima del modello.

CHAID (Chi-Square Automatic Interaction Detection)

Kass (1980) propose una modifica al modello AID utilizzando come metodo per la selezione dell'attributo di suddivisione il test di indipendenza Chi-Quadro. L'algoritmo è limitato alle variabili discrete: la costruzione dell'albero avviene mediante partizione ricorsiva delle osservazioni disponibili usando delle suddivisioni non necessariamente binarie in corrispondenza degli attributi il cui valore del test Chi-Quadro sia significativo rispetto ai nodi generati.

Per ogni attributo l'algoritmo calcola la tabella di contingenza rispetto alla variabile risposta: vengono quindi memorizzati i valori relativi al χ^2-value e al *p*-value. Una volta confrontati i risultati, viene selezionato l'attributo con il *p*-value più basso e comunque inferiore alla soglia prestabilita per la suddivisione; qualora nessun attributo abbia una correlazione significativa rispetto alla variabile risposta,

viene creata una foglia con l'etichetta di classe di maggioranza.

Nel caso l'attributo abbia più di due valori, vengono raggruppati via via tutti i valori meno significativi (con *p*-value più alto), calcolando il *p*-value relativo con la seguente formula

$$B_{free} = \sum_{i=1}^{r-1} (-1)^i \frac{(r-i)^c}{r!(r-i)!}$$

dove c indica il numero di valori dell'attributo ed r i valori ottenuti mediante il raggruppamento. Tale meccanismo tende a mettere insieme valori omogenei rispetto alla variabile risposta: dunque la situazione è identica rispetto alla discretizzazione.

CART (Classification And Regression Trees)

Agli inizi del 1970 Leo Breiman (Berkeley), Jerome Friedman (Stanford), Richard Olshen (Stanford) e Charles Stone (Berkeley) iniziarono una collaborazione per la stesura di un algoritmo che permettesse la costruzione di un modelli ad albero competitivi rispetto alle classiche procedure: il loro lavoro fu pubblicato nel 1984 insieme ad un software commerciale. Come evidente dal nome, l'algoritmo supporta la creazione di alberi sia di tipo decisionale che di regressione.

Per quanto riguarda la costruzione di alberi decisionali, la procedura si basa sulla cosiddetta *funzione di impurità* che definisce la massima omogeneità tra nodi figli: siccome l'impurità del nodo padre t_p è costante per ognuna delle possibili suddivisioni, la massima omogeneità tra i nodi figli (t_r per il nodo destro e t_l per il nodo sinistro) equivale alla massimizzazione della variazione della funzione di impurità

$$\begin{aligned} \Delta i(t) &= i(t_p) - E[i(t_c)] \\ &= i(t_p) - P_l i(t_l) - P_r i(t_r) \end{aligned}$$

dove t_c rappresentano i nodi destro e sinistro del nodo padre t_p, P_r e P_l

sono rispettivamente le probabilità del nodo destro e sinistro.

Perciò ciascun nodo dell'algoritmo CART massimizza la seguente quantità

$$\underset{x_j \leqslant x_j^R,\ j=1,2,\ldots,M}{arg\,max} \quad [i(t_p) - P_l i(t_l) - P_r i(t_r)]$$

dove x_j rappresenta l'attributo j-esimo e x_j^R il miglior valore di suddivisione di x_j.

In teoria è possibile utilizzare diverse funzioni di impurità: tuttavia la formulazione originale prevede l'utilizzo dell'indice di Gini e Twoing (termine coniato da Breiman). A tal proposito gli autori hanno dimostrato che il modello finale non risente assolutamente della scelta della funzione di impurità a causa della successiva fase di potatura dell'albero.

Detta p_i la probabilità che un'osservazione appartenenza alla classe i-esima (rispetto ai possibili K valori) in termini di frequenza rispetto alla numerosità dell'insieme T, l'indice di Gini è definito come

$$gini(T) = 1 - \sum_{i=1}^{K} p_i^2 \qquad con\ 0 \leqslant gini(T) \leqslant \frac{1}{K}$$

L'indice Twoing invece ricerca le due classi tali per cui la somma delle osservazioni appartenenti alle due sub-popolazioni è superiore del 50% rispetto all'intero insieme dei dati. Sebbene tale procedura permetta la creazione di alberi maggiormente bilanciati, il suo utilizzo è limitato a causa della maggior lentezza di calcolo rispetto all'indice di Gini.

Per quanto riguarda gli alberi di regressione, le operazioni di suddivisione eseguono la selezione dell'attributo che minimizza i quadrati dei residui o più precisamente che rende minima la somma delle varianze dei due nodi figli generati:

$$\underset{x_j \leqslant x_j^R,\ j=1,2,\ldots,M}{arg\,min} \quad [P_l Var(Y_l) + P_r Var(Y_r)]$$

I valori mancanti sono esclusi durante la ricerca della suddivisione.

L'altra grande innovazione introdotta dall'algoritmo CART è la fase di potatura: in un periodo in cui l'attenzione era focalizzata nello studio di un metodo per individuare la soglia di crescita migliore, gli autori furono i primi a dichiarare inappropriata la scelta di tale valore perché fortemente sensibile alla qualità e alla numerosità dei dati. Come metodologia per identificare il modello ottimo, l'algoritmo si basa sulla convalida incrociata e una funzione di costo-complessità basata sulla dimensione dell'albero e il numero di errori di classificazione ottenuti.

ID3, C4.5 e C5

Gli algoritmi presentati in questo paragrafo sono frutto del lavoro di Ross Quinlan e godono di una notevole importanza nell'ambito dell'Intelligenza Artificiale: ad esempio ID3 fu a lungo utilizzato come motore di applicativi sperimentali e professionali per il gioco degli scacchi.

Nel 1978 Quinlan scrisse il programma ID3 (*Iterative Dichotomizer 3rd*) come soluzione al problema della classificazione di variabili discrete con attributi discreti; nel 1993 pubblicò la seconda versione del programma, denominata C4.5, con l'estensione alle variabili continue. L'attuale versione C5 implementa nuove ottimizzazioni nella selezione degli attributi durante la fase di crescita dell'albero: a differenza delle precedenti versioni, il programma è diventato commerciale.

La novità dell'approccio proposto da Quinlan consiste nel basare la procedura di crescita dell'albero sul valore dell'entropia. Il concetto di entropia fu introdotto da Shannon (1948) nella teoria delle comunicazioni: indica il numero medio di bit necessari per identificare un certo messaggio tra n possibili da una sorgente inaffidabile. Formalmente, detta S una sorgente che può produrre n messaggi $\{m_1,m_2,..., m_n\}$ ciascuno generato indipendentemente dagli altri e p_i la

probabilità del *i*-esimo messaggio m_i, viene definita entropia della probabilità di un certo messaggio $P=(p_1,p_2,...,p_n)$ la quantità ottenuta dalla seguente funzione

$$\mathrm{H}(P)=-\sum_{i=1}^{n} p_i \log_2(p_i)$$

Intuitivamente, il numero di bit richiesti per individuare una monetina nascosta in una scatola tra 8 possibili è 3, ossia $3 = \log_2 8 = -\log_2 1/8$ dove la probabilità di ciascuna alternativa è costante e pari a 1/8.

Nel caso della classificazione in cui ciascun nodo possiede un insieme T di osservazioni partizionato in k classi $\{C_1,C_2,...,C_k\}$, la probabilità di distribuzione delle classi è stimata come

$$P_T=\left(\frac{|C_1|}{|T|},\frac{|C_2|}{|T|},...,\frac{|C_k|}{|T|},\right)$$

dove |A| rappresenta la numerosità del generico insieme A. Dunque è possibile identificare l'entropia dell'insieme $H(T)$ con l'entropia della probabilità della distribuzione nelle sue componenti $H(T_P)$.

La suddivisione dell'insieme T delle osservazioni nei sottoinsieme $T_1,T_2,...,T_n$ sulla base dell'attributo X comporta che l'informazione necessaria per identificare la classe di appartenenza di un elemento di T è pari alla media pesata dell'informazione necessaria ad identificare la classe di un elemento di ciascun sottoinsieme. La notazione matematica relativa al concetto di *informazione* è

$$info_X(T)=\mathrm{H}(X,T)=\sum_{i=1}^{n}\frac{|T_i|}{|T|}\mathrm{H}(T_i) \qquad con\ 0\leqslant H(X,T)\leqslant 1$$

E' intuitivo verificare che

- l'entropia è minima quando tutti gli esempi in T appartengono alla stessa classe C_j: le frequenze T_i saranno sempre pari a 0 tranne che per la classe *j*-esima, la cui probabilità sarà pari ad 1 (poniamo $\log_2 0 = 0$). Nella teoria delle comunicazioni il ricevente non ha bisogno di nessuna informazioni dal momento

che il mittente invierà un solo tipo di messaggio.

- l'entropia è massima quando tutte le classi hanno lo stesso numero di osservazioni.

Dunque l'entropia viene usata nella classificazione perché misura l'impurità di un insieme: è massima (più impuro) quando l'insieme contiene lo stesso numero di osservazioni per ciascuna classe, mentre è minima (più puro) per la presenza di osservazioni appartenenti ad una sola classe. Siccome l'interesse è focalizzato nella ricerca degli attributi che diano maggiore informazione, si introduce il *guadagno di informazione* inteso come numero di bit risparmiati conoscendo il valore di un certo attributo X

$$gain(X,T) = H(T) - H(X,T)$$

Quindi la procedura di selezione dell'attributo su cui effettuare la suddivisione identifica quel valore dell'attributo che comporta il più alto guadagno di informazione rispetto agli altri.

Tuttavia questa soluzione non è definitiva: il criterio del guadagno tende a favorire gli attributi con cardinalità elevata. Il caso limite si ottiene in presenza di un attributo contenente una chiave univoca (ad esempio il codice fiscale): tutte le osservazioni hanno una frequenza pari ad 1 con la conseguenza che l'entropia condizionata $H(X,T)$ sarà massima.

Nel 1986 Quinlan propose l'utilizzo di un indice di guadagno normalizzato, definito come rapporto tra la riduzione di entropia ottenuta mediante il test sull'attributo X e l'entropia propria dello split su X:

$$gainratio(X,T) = \frac{gain(X,T)}{splitinfo(X,T)} = \frac{gain(X,T)}{H(P_{X,T})}$$

Per le variabili continue, definiti $\{V_1, V_2, ..., V_m\}$ gli m valori della variabile continua nell'insieme dei dati ordinati dal più grande al più piccolo, la procedura di selezione del migliore split si comporta

analogamente al caso discreto considerando tutti i possibili m-1 valori soglia V_j che dividono l'insieme in due parti: $\{V_1, ..., V_j\}$ secondo l'affermazione $A \leq V_j$ e $\{V_{j+1}, ..., V_m\}$ per l'alternativa $A > V_j$.

La procedura di creazione dell'albero viene seguita da una particolare fase di potatura detta *pessimistic pruning*: l'algoritmo si basa su una stima del tasso di errore al fine di sostituire un nodo con un sotto-albero o con una foglia di quest'ultimo se la stima dell'errore è più piccola rispetto a quello complessivo del sotto-albero. In particolare il test statistico sul numero di errori utilizza un intervallo di confidenza (25% di default).

Nella fase di creazione dell'albero, i valori mancanti influenzano in maniera particolare il calcolo dell'entropia: il concetto di fondo è che l'informazione aggiuntiva data da un attributo senza valore deve essere nulla. Dunque posto T l'insieme delle osservazioni su cui fare il test e F il sottoinsieme delle osservazioni non nulle sull'attributo A, applicando il test X su A vale $info(T)=info(F)$ e $info_X(T)=info_X(F)$: il guadagno di informazione deve essere rettificato perché l'insieme delle osservazioni con valore nullo sull'attributo A ($T \setminus F$) non portano alcuna informazione aggiuntiva. Questo equivale a considerare un sottoinsieme aggiuntivo nel calcolo dell'informazione

$$splitinfo(X, T) = -\sum_{i=1}^{n+1} \frac{|T_i|}{|T|} \log_2 \frac{|T_i|}{|T|} \qquad \text{con } T_{n+1} = T \setminus F$$

Nella fase di previsione ciascuna osservazione ha un peso associato pari ad 1: qualora l'osservazione abbia valore nullo su un attributo oggetto di test, si devono esplorare tutti i K nodi sottostanti aggiornando il peso con il rapporto tra la somma dei pesi delle foglie accessibili dal j-esimo nodo figlio (per ogni j=1,...,K) e il totale dei pesi delle foglie accessibili da tutti i K nodi figli. Il risultato del procedimento consiste nell'accesso a più foglie, ossia una distribuzione di classi invece della singola classe: la probabilità di ciascuna classe è dato dal peso finale dell'osservazione moltiplicato per la probabilità

della classe nella foglia. Ovviamente verrà scelta la classe con maggiore probabilità, con l'accortezza di sommare le probabilità delle foglie con la stessa classe.

1.2.7. Reti neurali

Le reti neurali artificiali (o semplicemente reti neurali) sono nate negli anni '40 nell'ambito nella neurobiologia come strumento per lo studio e comprensione dei meccanismi che governano il sistema nervoso animale (*reti neurali biologiche*). Questi studi furono ripresi negli anni '50-'60 dalla ricerca sull'intelligenza artificiale: l'obiettivo consisteva nell'estenderne i risultati all'ambito informatico, al fine di consentire anche al computer l'apprendimento automatico. L'esito del progetto fu molto limitato a causa dell'eccessiva complessità computazionale e la limitatezza teorica dei primi modelli elaborati. Negli anni '80 tali ricerche raccolsero i primi importanti risultati grazie al progresso tecnologico e all'ideazione di nuovi algoritmi più efficienti.

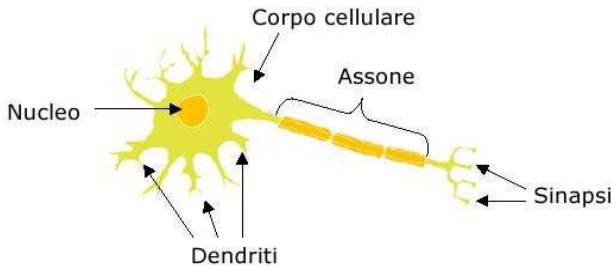

Figura 1.5: Neurone biologico

In biologia il neurone è composto da:

- un corpo cellulare, detto nucleo.
- uno o più dendriti, il cui compito consiste nella ricezione dei segnali inviati da altri neuroni.
- un assone che a sua volta veicola i segnali generati dallo stesso neurone agli altri neuroni connessi.

Ogni neurone è collegato a sua volta ad uno o più neuroni attraverso la

sinapsi.

L'attività del neurone consiste nell'alternanza di invio del segnale (*stato attivo*) e riposo/ricezione dei segnali provenienti dagli altri neuroni (*stato non attivo*). Il passaggio da una fase all'altra è causata dalle sollecitazioni esterne rappresentate dai segnali che vengono captati dai dendriti: ciascun segnale ha un effetto eccitatorio o inibitorio, concettualmente rappresentabile da un certo peso associato allo stimolo. Il neurone in stato inattivo cumula tutti i segnali ricevuti finché non viene raggiunta una certa soglia (detta *soglia di attivazione*) che determina il passaggio allo stato attivo: in tal modo il neurone invia un proprio segnale. Il neurone biologico è la componente fondamentale di un sistema nervoso, la cui capacità e flessibilità è proporzionale al numero di neuroni e alle connessioni tra gli stessi neuroni.

Figura 1.6: Neurone artificiale

Analogamente a quello biologico, il neurone artificiale è composto da:

- una o più connessioni di entrata, con il compito di raccogliere i segnali (numerici) inviati da altri neuroni: a ciascuna connessione è assegnato un peso che verrà utilizzato per considerare correttamente ogni segnale inviato;
- una o più connessioni di uscita che veicolano il segnale destinato agli altri neuroni;
- funzione di attivazione: determina il valore numerico del segnale in uscita, sulla base dei segnali ricevuti dalle

connessioni di entrata con altri neuroni e opportunamente raccolti, dai pesi associati a ciascun segnale captato e alla soglia di attivazione del neurone stesso.

In particolare la funzione di attivazione può essere a sua volta scomposta in:

- *Potenziale netto*, ossia la funzione che combina opportunamente tutti i segnali in ingresso considerando il peso associato a ciascun segnale. Il valore di tale funzione (generalmente lineare) viene confrontato con il livello di attivazione: a seconda dell'esito del confronto viene inviato o meno il segnale d'uscita. Per l'j-esimo neurone il potenziale netto è $P_j=f(w_h, x_h; h)$, ossia una funzione che dipende dal peso w_h e segnale x_h rispetto ai neuroni P a cui è connesso, con h=1,2,...,P; definita θ_j la soglia di attivazione, il neurone invia il segnale quando $P_j - \theta_j > 0$.

- *Funzione di trasferimento*, è la funzione che genera il valore numerico del segnale di uscita mediante l'opportuna trasformazione del potenziale netto: $z_j = f(P_j)$.

  Generalmente tali funzioni sono di tipo lineare, a gradini, a rampa o sigmoidali.

L'insieme dei neuroni artificiali e le loro connessioni formano la rete neurale artificiale, la cui struttura in termini di organizzazione è detta topologia o architettura di rete.

Convenzionalmente si identificano tre gruppi di neuroni, detti *strati*: lo strato di input identifica i neuroni che ricevono i segnali dall'esterno della rete e hanno la particolarità di trasmettere il messaggio senza modificarne il valore; lo strato di output che invia i segnali all'esterno della rete e lo stato nascosto o latente caratterizzato da neuroni che comunicano sia in input che in output con altri neuroni, senza aver nessun contatto con l'esterno della rete.

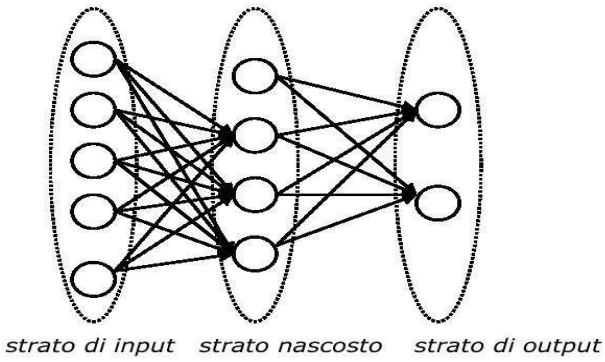

Figura 1.7: Rete neurale artificiale

Tale definizione lascia notevole libertà sulle possibili tipologie di rete e connessioni tra neuroni: in particolare una rete è detta *autoassociativa* se lo strato di input e lo strato di output coincidono; è *eteroassociativa* se lo strato di input è diverso dallo strato di output. Inoltre collegamenti tra neuroni appartenenti allo stesso strato o a strati differenti sono detti rispettivamente *connessioni intrastrato* e *interstrato*. Infine una rete in cui ogni neurone è collegato (tramite connessioni intrastrato e interstrato) a tutti gli altri neuroni è detta *totalmente connessa*, mentre una rete in cui ogni neurone è collegato solo a tutti i neuroni dello strato precedente e dello strato successivo è detta *totalmente interconnessa*: sono ammesse anche reti *parzialmente connesse* e *parzialmente interconnesse*.

Per completare la topologia di una rete è necessario stabilire anche la direzione di transito del flusso di segnali (direzione delle connessioni): una rete totalmente (o parzialmente) interconnessa è detta di tipo *feed-forward* se le connessioni sono unidirezionali (i segnali si muovono in un'unica direzione dallo strato di input a quello di output); se le connessioni sono bidirezionali, la rete è detta di tipo *feed-back*.

La rete neurale rappresenta un modello previsivo: ogni segnale in ingresso è una variabile esplicativa mentre il segnale in uscita rappresenta la previsione. Il neurone rappresenta una trasformazione lineare o non lineare delle variabili esplicative in cui i parametri sono rappresentati dai pesi (associati ai segnali in input); la topologia della

rete rappresenta la forma funzionale.

La fase di apprendimento della rete neurale corrisponde alla stima dei parametri attraverso cui il modello è in grado di fornire la previsione a fronte di una certa combinazione delle variabili esplicative. Tuttavia è necessario definire inizialmente la tipologia di rete desiderata, il numero di unità presenti nello strato latente, la funzione di attivazione e quella di trasferimento.

L'approccio di stima dei parametri consiste come sempre nella minimizzazione dell'errore di stima: a parte la complessità che cresce proporzionalmente al numero di unità dello strato latente, le derivate parziali rispetto ai pesi risultano essere spesso non lineari; inoltre anche con topologie semplici possono esistere più punti di minimo locale. In definitiva è assai raro che esista una soluzione analitica al problema di stima.

Nel seguito della relazione verrà trattato solo il caso del percettrone multistrato che rappresenta l'implementazione più semplice e più utilizzata nell'ambito del data mining.

Percettrone multistrato (Multi-Layer Perceptron)

Una rete percettrone multistrato è una rete neuronale artificiale eteroassociativa, con uno strato nascosto, totalmente interconnessa e di tipo feed-forward in cui:

- il potenziale netto dei neuroni che compongono lo strato nascosto e lo strato di output viene calcolato usando la funzione somma
- la funzione di trasferimento dei neuroni dello strato nascosto è di tipo sigmoidale (funzione logistica)

Una volta fissato il numero r di unità nello strato latente, si ottiene il vettore dei parametri $\theta=(\alpha_{01}, ..., \alpha_{0r}, \alpha_{11}, ..., \alpha_{1r}, ..., \alpha_{p1}, ..., \alpha_{pr}, \beta_0, ..., \beta_r)$ da stimare utilizzando i valori osservati (sempre in forma vettoriale) x_i,

con $i = 1,2,\ldots, n$. Nella figura seguente è presentato il percettrone multistrato con lo strato latente composto da tre unità e relativo al modello di regressione logistica: in forma matematica

$$f(x) = \frac{\exp(\alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3)}{1 + \exp(\alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3)}$$

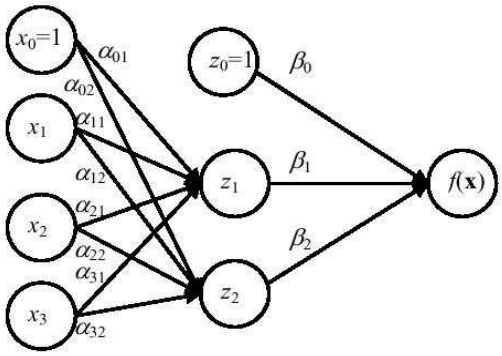

Figura 1.8: Percettrone multistrato (MLP)

Per la stima dei parametri viene utilizzato l'algoritmo di retropropagazione (*backpropagation*), una tecnica iterativa che consente la minimizzazione dell'errore di previsione empirico. L'algoritmo viene inizializzato mediante un vettore di valori $\theta^{(0)}$ e ad ogni iterazione procede secondo i seguenti passi:

1. *forward pass*: a partire dai valori osservati e dai valori dei parametri ottenuti al passo precedente, si calcolano gli *n* segnali in uscita dal percettrone multistrato $f(x_i)$;
2. *backward pass*: ciascun valore del parametro viene modificato sulla base del confronto (tramite la funzione di perdita) tra $f(x_i)$ e y_i in modo da ridurre l'errore di previsione.

Tale algoritmo soffre di alcune limitatezze. Anzitutto non è assicurato il raggiungimento del minimo assoluto: è conveniente creare diversi modelli con differenti valori iniziali dei parametri, in modo da selezionare la soluzione migliore in termini di errore. Inoltre risulta essere lento nella convergenza del valore dei parametri.

1.2.8. Classificazione basata sulle istanze

La classificazione basata sulle istanze (*instances-based learning*) è una tecnica molto particolare perché a differenza delle altre tecniche che mirano alla creazione di un modello dai dati di stima, essa basa la procedura di previsione direttamente sui dati già classificati, mediante il confronto delle caratteristiche delle osservazioni.

Per questo motivo tale tecnica è detta *lazy learning*: è pigra proprio perché nella fase di training non fa assolutamente nulla per comprendere i dati. Risulta dunque in antitesi con l'approccio classico che concentra lo sforzo maggiore nella fase di stima al fine di generare un'astrazione dei dati stessi: in tale contesto è quindi detto *eager learning*.

Dovendo confrontare le nuove osservazioni con esempi già classificati, la tecnica necessita la definizione dei concetti di istanza simile e di classificazione simile: dal punto di vista algoritmico tale configurazione si concretizza nella scelta della funzione che misura la similarità tra osservazioni e della funzione di selezione della classe di appartenenza di una nuova osservazione sulla base degli esempi simili.

K-NN (k-Nearest Neighbor)

E' l'implementazione più semplice di tale tecnica: l'algoritmo *nearest neighbor* (Cover e Hart, 1967) restituisce la classe dell'esempio il più vicino possibile all'osservazione da classificare. Normalmente viene usata la versione modificata, detta *k-nearest neighbor* (k-NN): la procedura classifica la nuova osservazione sulla base dei k esempi più vicini. Dunque la formulazione originale rappresenta il caso particolare 1-NN.

Nella fase di training, l'algoritmo memorizza gli esempi già classificati associando al vettore degli attributi dell'osservazione x_j la funzione $f(x_j)$ che determina il valore della classe, per ogni j-esima

osservazione.

Nella successiva fase di classificazione della generica osservazione x_q, vengono cercati i k esempi più vicini da cui selezionare la classe più rappresentativa. La vicinanza tra osservazioni viene calcolata tramite la distanza euclidea di tutti gli attributi dell'osservazione: le variabili quantitative vengono normalizzate affinché la diversa scala di misura non incida sulla distanza; le variabili qualitative sono sostituite da variabili indicatrici che assegnano il valore 1 nel caso di uguaglianza del valore dell'attributo e 0 altrimenti.

Nel caso in cui l'attributo sia discreto, viene restituito il valore v che risulta più frequente tra i k esempi selezionati rispetto ai possibili V valori che può assumere la classe:

$$f'(x_q) = \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} I(v, f(x_i))$$

dove f' rappresenta la stima della f per x_q, f è la funzione che ritorna un valore di V rispetto al vettore degli attributi x_j relativa alla j-esima osservazione e I è la funzione indicatrice.

Nel caso della classificazione di variabili continue, viene restituita la media dei valori restituiti dalle funzioni relative ai k esempi selezionati:

$$f'(x_q) = \frac{1}{k} \sum_{i=1}^{k} f(x_i)$$

Una semplice estensione consiste nel considerare il contributo di ciascun dei k esempi più vicini mediante pesi proporzionali alla distanza tra la nuova osservazione e ciascun esempio:

$$w_i = \frac{1}{d(x_q, x_i)^2}$$

Dunque le precedenti stime di f possono essere riscritte come

$$f'(x_q) = \operatorname{argmax}_{v \in V} \sum_{i=1}^{k} w_i I(v, f(x_i))$$

$$f'(x_q) = \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

IB1, IB2, IB3, IB4, IB5

Aka, Kibler e Albert (1991) hanno descritto tre algoritmi basati sulle istanze in ordine crescente di complessità.

IB1 è un'implementazione del nearest neighbor per variabili continue: i valori vengono normalizzati e i valori mancanti vengono sostituiti al fine di essere il più lontani possibile da ciascuna osservazione.

IB2 contiene delle ottimizzazioni al fine di ridurre la quantità di memoria utilizzata per contenere gli esempi di training; introduce anche la novità di memorizzare le osservazioni che sono state classificate erroneamente.

IB3 è un'ulteriore estensione al fine di essere robusta rispetto a valori anomali (*noisy data*): le istanze classificate erroneamente vengono eliminate mentre le istanze corrette vengono a loro volta utilizzate per la classificazione.

Aka (1992) ha descritto IB4 e IB5 aggiungendo la possibilità di gestire attributi irrilevanti e nuovi rispetto a quelli che contraddistinguono i dati di stima.

1.2.9. Regole di associazione

La tecnica delle regole associative è nata al fine di individuare attributi fortemente correlati sulla base delle osservazioni disponibili, la cui numerosità è per definizione molto elevata. Questo particolare scenario è tipico dell'analisi del paniere della spesa: si è interessati alla ricerca di prodotti con forte relazione di associazione, come ad

esempio il panettone e il torrone nel caso degli acquisti in periodo natalizio. L'approccio è di tipo descrittivo, dal momento che il risultato finale consiste in un elenco di cosiddette *regole*, ossia relazioni tra attributi del tipo "se è presente l'attributo *X*, allora l'osservazione conterrà anche l'attributo *Y*". Graficamente una regola è rappresentata come segue

$$X \Rightarrow Y$$

dove *X* è detto antecedente o LIIS (*left-hand side*) e *Y* conseguente o RIIS (*right-hand side*). Dal punto di vista teorico, l'antecedente può contenere più attributi mentre il conseguente deve essere unico.

A seconda del tipo di contesto, la ricerca delle regole di associazione è detta *association discovery* e *sequence discovery*: in particolare quest'ultima tipologia tende ad individuare un'importante campo di applicazione il cui scopo consiste nell'individuare relazioni di tipo sequenziale/temporale tra gli attributi.

E' facile intuire la possibilità di generare un numero molto elevato di regole anche in presenza di poche decine di attributi: infatti tale tecnica deve necessariamente considerare tutte le possibili combinazioni degli attributi tra loro. Per limitare il numero di regole e soprattutto di individuare solo le regole statisticamente più interessanti, sono stati definiti i seguenti indici.

- *Supporto*: corrisponde alla frazione di osservazioni per le quali si verifica la presenza dell'attributo *X*, ovvero il totale delle osservazioni con l'attributo *X* rispetto alla numerosità campionaria. La notazione matematica più semplice è quella relativa al calcolo delle probabilità, ossia

  $$supp(X) = \mathrm{P}(X)$$

  Nel caso di una regola associativa tra i due attributi *X* e *Y*, il supporto è definito come

$$supp(X \Rightarrow Y) = \mathrm{P}(X \cap Y) \quad \text{con } X \cap Y \neq \emptyset$$

- *Confidenza*: corrisponde alla frazione di osservazioni per le quali l'attributo X implica la presenza dell'attributo Y, ossia

$$conf(X \Rightarrow Y) = \mathrm{P}(Y|X) = \frac{supp(X \cup Y)}{supp(X)}$$

- *Lift*: corrisponde alla devianza tra il supporto della regola tra X e Y rispetto al supporto che si ottiene sotto l'ipotesi di indipendenza tra i due attributi, ossia

$$lift(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)\,supp(Y)}$$

Gli algoritmi per la ricerca di regole associative mirano ad individuare tutte le regole con un valore di supporto e confidenza tale da rispettare una certa soglia prefissata.

Apriori

L'algoritmo Apriori è senza dubbio l'implementazione più importante di tale tecnica. Sviluppato da Rakesh Agrawal nella divisione di ricerca IBM, tale algoritmo parte dal presupposto che ciascuna osservazione rappresenti una particolare transazione, i cui attributi corrispondono alla presenza di un determinato prodotto, detto *item*. Ciascuna transazione è memorizzata in un record del database avente un identificativo univoco, detto TID. In tale contesto, detto $I = \{i_1, i_2, ..., i_m\}$ un insieme di m item, la regola di associazione $X \Rightarrow Y$ è tale che $X \subset I$, $Y \subset I$, $X \cap Y = \emptyset$.

L'intuizione alla base dell'algoritmo consiste nella ricerca di un insieme di prodotti detti *itemset* molto grandi (*large*) con una frequenza tale da garantire la soglia di minimo supporto (*minsupp*). Una volta isolati tali itemset, le regole sono generate considerando tutti i vari sottoinsiemi di item tali per cui viene soddisfatta la soglia di minima confidenza: in dettaglio, detto L_k l'insieme di tutti gli itemset

di k elementi e l un generico elemento di tale insieme, verrà creata la regola relativa ad un sottoinsieme a di l

$$a \Rightarrow (l-a) \text{ se e solo se } \frac{supp(l)}{supp(a)} > minconf$$

Questo approccio nella generazione delle regole è giustificato dalla cosiddetta regola *a priori*, secondo la quale se un itemset è frequente, allora tutti i suoi sottoinsiemi sono frequenti. Ad esempio dato l'itemset {1 2 3} frequente, risulta che anche {1 2},{1 3} e {2 3} sono frequenti; viceversa anche se solo {1 2} non fosse frequente, non lo sarebbe nemmeno {1 2 3}.

L'algoritmo Apriori procede all'identificazione degli itemset più frequenti mediante la costruzione successiva di itemset più grandi: detto L_{k-1} l'insieme di tutti gli itemset più frequenti di k-1 elementi, la creazione del successivo insieme L_k avviene secondo i seguenti passi.

- Passo di combinazione (*join*): vengono combinati tutti gli elementi di L_{k-1} al fine di creare l'insieme candidato di itemset con k elementi C_k con k-2 elementi comuni, ossia

  $$C_k = \{A \cup B \mid A, B \in L_{k-1}, |A \cap B| = k-2\}$$

  Ad esempio posto L_3 = {{1 2 3},{1 2 4},{1 3 4},{1 3 5},{2 3 4}}, verrà creato l'insieme candidato C_4= {{1 2 3 4},{1 2 3 5},{1 3 4 5}}.

- Passo di taglio (*prune*): vengono eliminati dall'insieme candidato C_k tutti gli itemset i cui sottoinsiemi di numerosità k-1 non appartengono a L_{k-1}, dal momento che a priori si può concludere che essi non possano avere confidenza minima. Tornando all'esempio del punto precedente, tale passo conserverà solo l'elemento {1 2 3 4}.

- Verifica degli itemset candidati: vengono conteggiate tutte le transazioni in cui compaiono gli itemset candidati in C_k. L'insieme finale L_k conterrà solo gli elementi di C_k che hanno

una frequenza che rispetti il valore soglia, ossia

$$L_k = \{c \in C_k | supp(c) \geqslant minsupp\}$$

La procedura inizia con L_1 e termina quando l'insieme degli itemset L_k risulta vuoto: successivamente si passa alla creazione delle regole di associazione.

1.2.10. Metodi di raggruppamento

I metodi di raggruppamento (*cluster analysis*) costituiscono una tecnica da applicare qualora vi sia la necessità di riunire le osservazioni in gruppi, senza però fornire al sistema alcun esempio di riferimento (tipico della classificazione) a causa dell'assenza di ipotesi sulla struttura dei dati stessi. Questo approccio lascia all'algoritmo il compito di isolare gruppi di dati (detti *cluster*) ritenuti interessanti sulla base di caratteristiche simili valutate con una certa metrica: la misura di vicinanza o equivalentemente di lontananza è detta rispettivamente *similarità* e *dissimilarità*. In generale si utilizza definire la dissimilarità tra due osservazioni i e i' come $d(i,i')$: tale funzione deve essere applicata a tutti i p attributi che caratterizzano un'osservazione, ossia $d_j(x_{ij},x_{i'j})$ con j=1,2,...,p.

Intuitivamente il concetto di dissimilarità deve rispettare le seguenti proprietà:

1. $d_j(x, x) = 0$
2. $d_j(x, x') \geq 0$
3. $d_j(x, x') = d_j(x',x)$

Da notare che l'ultima proprietà di simmetria non è un requisito necessario, ma di fatto è utilizzata dalla gran parte degli algoritmi.

Per le variabili quantitative la funzione più naturale impiegata per misurare la dissimilarità è la distanza euclidea: $d_j(x, x') = (x - x')^2$. Tuttavia possono essere usate altre misure, come la distanza di

Mahalanobis, di Minkowsky o di Manhattan.

Per le variabili qualitative sconnesse solitamente si utilizza $d_j(x, x') = 1\text{-}I(x=x')$, con $I(x=x')$ funzione indicatrice che vale 1 se x e x' coincidono, altrimenti 0.

Le variabili qualitative ordinali, in cui è implicito un ordinamento dei valori (ad esempio il titolo di studio), sono spesso trattate come variabili quantitative mediante l'assegnazione di un punteggio crescente ad ogni valore possibile della variabile.

Una volta selezionate le metriche da usare per ciascun tipo di variabile, si procede alla definizione della dissimilarità tra osservazioni i e i' che intuitivamente sarà costruita come somma delle dissimilarità per ciascun attributo dell'osservazione: tuttavia è importante introdurre un peso soggettivo ad ogni tipo di variabile al fine di confrontare in maniera calibrata il diverso contributo. Perciò detti P_{QT}, P_{QLO}, P_{QLS} rispettivamente l'insieme degli attributi quantitativi, qualitativi ordinali e qualitativi sconnessi, w_{QT}, w_{QLO}, w_{QLS} i rispettivi pesi e d_j^{QT}, d_j^{QLO}, d_j^{QLS} le rispettive dissimilarità, la dissimilarità tra osservazioni i e i' è data da

$$d(i,i') = \frac{\sum_{j \in P_{QT}} w_{QT} d_j^{QT}(i,i') + \sum_{j \in P_{QLO}} w_{QLO} d_j^{QLO}(i,i') + \sum_{j \in P_{QLS}} w_{QLS} d_j^{QLS}(i,i')}{w_{QT} w_{QLO} w_{QLS}}$$

Un'ulteriore concetto utilizzato dalla maggior parte degli algoritmi è la cosiddetta *matrice di dissimilarità D*, i cui elementi sono i risultati dell'applicazione della funzione di dissimilarità tra le n osservazioni. Tale matrice, diagonale e di dimensione n x n, è inoltre caratterizzata da elementi nulli nella diagonale ed elementi non negativi altrove qualora la funzione di dissimilarità soddisfi la proprietà simmetrica.

Sebbene il concetto di clustering sia estremamente semplice, esistono molti approcci diversi: dal punto di vista algoritmico, la difficoltà maggiore consiste nell'ottimizzazione della ricerca e confronto tra le n osservazioni e la descrizione di topologie non regolari dei cluster. Di

seguito vengono descritti i metodi più comuni.

Il metodo non gerarchico identifica un modus operandi in cui le osservazioni vengono attribuite ad un solo cluster sulla base della similarità delle osservazioni: a sua volta ogni cluster deve essere il più possibile “lontano” rispetto agli altri.

Il metodo gerarchico punta alla decomposizione dell'insieme delle osservazioni al fine di ottenere una struttura gerarchica rappresentabile da un albero binario le cui foglie contengono le singole osservazioni mentre i nodi costituiscono i sottoinsiemi rispetto all'insieme dei dati di partenza. Tipicamente i procedimenti con cui arrivare a questo risultato sono due:

- metodi agglomerativi: si fondano sulla metodologia *bottom-up*. Inizialmente ogni oggetto dell'insieme forma un cluster differente, poi i cluster vengono fusi (*merged*) fino alla soddisfazione di una condizione di arresto o alla formazione di un unico grande cluster.
- metodi divisivi: procedono al contrario, ovvero secondo l'approccio *top-down*. Vi è un unico cluster iniziale che viene decomposto (*splitted*) lungo un albero di cluster fino al raggiungimento di una condizione di arresto, come il numero di cluster desiderati o il raggiungimento di un limite inferiore per la distanza dei due cluster più vicini.

Il grosso limite che accomuna gli approcci finora presentati, detti di partizionamento, è l'irreversibilità delle scelte effettuate (fusione o decomposizione): se un'operazione non è effettuata oculatamente, non esiste la possibilità di correggere l’errore compromettendo così il risultato finale. Inoltre la costruzione di cluster basati sulla misurazione della distanza a coppie di osservazioni, limita la rappresentazione di cluster approssimativamente sferici e risulta del tutto inefficiente con altre topologie di raggruppamenti.

Il metodo basato sul concetto di di densità permette la soluzione al problema della topologia dei gruppi: una volta trovato un cluster, si continua ad accrescerlo aggiungendovi osservazioni se la densità nelle "vicinanze" (*neighbourhood*) supera una certa soglia. Come risultato si ottiene la capacità di identificare cluster di qualsiasi contorno e di isolare allo stesso tempo le osservazioni outliers.

K-medie

L'algoritmo delle K-medie si basa sul metodo non gerarchico e interessa solo variabili quantitative: fissato il numero K di cluster, viene misurata la similarità delle osservazioni rispetto al valore medio di ogni gruppo. La procedura è iterativa: selezionati casualmente K punti di aggregazione (detti *centroidi*) attorno ai quali formare i cluster, vengono effettuati i seguenti passi:

1. la i-esima osservazione x_i è assegnata al cluster per cui è minimizzata la distanza euclidea tra l'osservazione e la media del cluster m_k, per ogni i=1,2,...,n
2. una volta riassegnate le osservazioni ai cluster, viene ricalcolata la media m_k per ogni k=1,2,...,K.

Il processo continua finché i centroidi m_1, m_2,...,m_k si stabilizzano.

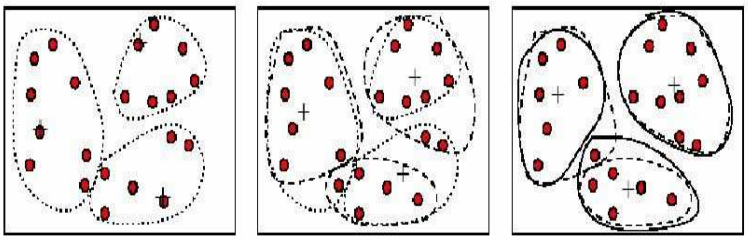

Figura 1.9: Composizione del cluster nell'evolvere dell'algoritmo K-medie

Dunque l'algoritmo delle K-medie cerca di minimizzare la funzione obiettivo $e^2=\sum_{j=1}^{K}\sum_{i\in C_j} d(x_i, c_j)$ dove c_j rappresenta il centroide del

generico cluster *j*-esimo, C_j.

Da notare che sia le osservazioni che i centroidi sono vettori, dal momento che ogni osservazione è composta da più attributi.

In figura 1.9 sono rappresentati a titolo esemplificativo tre tipi di raggruppamento all'evolvere dell'algoritmo: nel primo si nota la scelta di un oggetto come centroide del gruppo, negli altri si notano gli spostamenti delle medie (denotate con +) e i passaggi di alcuni oggetti da un gruppo all'altro determinando i cambiamenti di forma. La complessità dell'algoritmo è $O(nKt)$, dove t è il numero di iterazioni.

L'algoritmo soffre di due importanti limitazioni: il numero K di cluster deve essere determinato dall'utente prima dell'esecuzione e quindi può rappresentare un problema nei casi in cui non vi è nessuna informazione a priori sui possibili gruppi in cui possono dividersi le osservazioni; inoltre è sensibile agli outliers, che possono deviare gravemente il valore medio dei cluster.

In letteratura sono presenti numerose varianti dell'algoritmo, quali le versioni di Lloyd (1957), Forgy (1965), MacQueen (1967), Hartigan e Wong (1979): tuttavia le personalizzazioni sono relative alla selezione dei centroidi iniziali e alle condizioni di fine dell'algoritmo.

PAM

L'algoritmo PAM (*Partitioning Around Medoids*) rappresenta un'estensione dell'algoritmo delle K-medie. Al posto di utilizzare i centroidi come punti di aggregazione, viene introdotto il concetto di *medoide* che rappresenta l'osservazione selezionata come rappresentante del gruppo e tale da minimizzare la dissimilarità dentro il cluster: in tal modo gli attributi possono essere anche discreti e la procedura risulta essere robusta rispetto alla presenza di outliers.

Più in dettaglio, dopo la scelta iniziale di K osservazioni rappresentative di ciascun gruppo, l'algoritmo prosegue iterativamente

secondo i seguenti passi:

1. ciascun medoide raggruppa le osservazioni con maggiore similarità;
2. si valuta ogni possibile coppia di osservazioni all'interno di ciascun cluster per trovare medioidi più rappresentativi.

Il costo di una singola iterazione è $O(k(n-k)^2)$: dunque sebbene più robusto, il costo computazionale è maggiore rispetto alle K-medie; inoltre necessita sempre del valore K come parametro di configurazione dell'algoritmo stesso.

Coweb e Classit

Con tali algoritmi viene ora introdotta una nuova modalità di analisi, detta *clustering incrementale*. La procedura provvede alla creazione di un albero che viene aggiornato ad ogni osservazione del dataset: tale fase di aggiornamento consiste nella creazione di una nuova foglia o nella ristrutturazione dell'intero albero, a seconda della qualità della partizione, misurata dalla cosiddetta category utility. La *category utility* è una funzione definita come

$$CU(C_1, C_2, \ldots, C_k) = \frac{1}{k} \sum_{l=1}^{k} \Pr(C_l) \sum_{i=1}^{p} \sum_{j=1}^{n} \left(\Pr(a_i = v_{ij} | C_l)^2 - \Pr(a_i = v_{ij})^2 \right)$$

dove $C_1, C_2, \ldots, C_k$ sono i k cluster, a_i è il generico attributo con $i=1,2,\ldots,p$ e v_{ij} il valore dell'attributo i-esimo dell'osservazione j-esima con $j=1,2,\ldots,n$. Dalla notazione matematica della categority utility è evidente come l'obiettivo di tale procedura consista nell'ottenere una miglior stima dei valori degli attributi, ossia il fatto di conoscere il valore dell'attributo condizionato all'appartenenza di un determinato cluster $\Pr(a_i = v_{ij} \mid C_l)$ deve essere preferibile rispetto all'assenza di tale conoscenza, data da $\Pr(a_i = v_{ij})$.

Nel caso di variabile numeriche, si assume che esse abbiano distribuzione normale con media e varianza calcolate nel dataset: la

category utility può essere dunque scritta come

$$CU(C_1 C_2, \ldots, C_k) = \frac{1}{k} \sum_{l=1}^{k} \Pr(C_l) \frac{1}{2\sqrt{\pi}} \sum_{i=1}^{p} \left(\frac{1}{\sigma_{il}} - \frac{1}{\sigma_i} \right)$$

Per evitare che la deviazione standard sia nulla (nodo con una sola istanza), viene utilizzato il parametro *acuity*, che rappresenta il valore minimo di σ_i, con $i = 1, 2, \ldots, p$.

Un altro parametro importante è il cosiddetto *cutoff* che impedisce il proliferare di foglie: esso forza il raggruppamento di osservazioni sufficientemente simili ad altre senza dover aggiungere nuove foglie.

Coweb e Classit rappresentano algoritmi specializzati nel clustering incrementale per osservazioni con attributi rispettivamente nominali e numerici.

EM (Expectation-Maximization)

L'algoritmo *Expectation-Maximization* è una tipologia di clustering basato sulle probabilità: esso si basa sull'idea che i k cluster siano ottenuti da altrettante distribuzioni di probabilità, ciascuna delle quali determina la presenza dei propri membri nel dataset. Tale metodo di clustering è basato sulle seguenti ipotesi:

- ad ogni cluster è associata una sola distribuzione di probabilità;
- ogni cluster ha una distribuzione diversa rispetto agli altri cluster;
- ogni osservazione appartiene ad un solo cluster, sebbene si ignori quale esso sia;
- una distribuzione fornisce la probabilità che un'osservazione assuma certi valori sui suoi attributi, supponendo che sia noto il cluster di appartenenza.

Il cuore dell'algoritmo consiste nella procedura di stima dei parametri delle distribuzioni: a tal fine viene utilizzato il metodo della massima

verosimiglianza applicato alle stesse distribuzioni. In particolare esistono due fasi iterative:

- Stima (*Expectation*): si calcola la probabilità di appartenenza ad un cluster utilizzando i parametri stimati nell'iterazione precedente. In particolare alla *g*-esima iterazione tale passo calcola la funzione

$$Q(\theta, \theta^{(g-1)}) = E\left[\log P(Y|\theta)|X, \theta^{(g-1)}\right]$$

- Massimizzazione (*Maximization*): si stimano nuovamente i parametri affinché venga massimizzata la funzione Q, ossia

$$\theta^{(g)} = arg\max_{\theta} Q(\theta, \theta^{(g-1)})$$

L'algoritmo si arresta quando la stima dei parametri converge.

DBScan

L'algoritmo DBSCAN (*Density Based Spatial Clustering of Application with Noise*) è un algoritmo basato sulla densità, sviluppato da Ester, Kriegel, Sander e Xu (1996): esso definisce un cluster come un insieme di osservazioni o punti connessi in densità (*density connected*).

Un oggetto è detto nucleo (*core–object*) se all'interno della sua ε-vicinanza (ε*-neighbourhood*), definita da un raggio ε, si trovano un numero sufficiente di punti; tale quantità è definita dal parametro *MinPts*.

Un oggetto *p* è direttamente raggiungibile in densità dall'oggetto *q* rispetto al raggio ε ed a un numero minimo di punti *MinPts* in un insieme *D* se *p* si trova all'interno della ε–vicinanza di *q* che contiene almeno *MinPts* punti.

Un oggetto *p* è raggiungibile in densità dall'oggetto *q* rispetto al raggio ε ed a un numero minimo di punti *MinPts* in un insieme *D* se esiste una successione di punti $p_1, p_2, \ldots, p_n$ con $p_1 = q$, $p_n = p$ e p_i

appartente a D e tale che p_{i+1} è raggiungibile in densità da p_i rispetto a ε e *MinPts* per ogni i=1,2,...,n.

Un oggetto p è connesso in densità ad un oggetto q rispetto a ε e *MinPts* in D se esiste un oggetto o appartenente a D che p e q sono raggiungibili in densità da o rispetto a ε e *MinPts*.

Un cluster basato sulla densità è un insieme di oggetti connessi in densità che è massimo rispetto alla raggiungibilità: ogni elemento estraneo al cluster è rumore. L'algoritmo verifica la ε–vicinanza di ogni punto dell'insieme dati: se viene raggiunto un numero *MinPts* di punti appartenenti alla ε–vicinanza di un punto p, viene creato un nuovo cluster con nucleo p; quindi procede all'aggiunta dei punti raggiungibili direttamente dal cluster, con la possibilità di portare alla fusione di regioni raggiungibili. La procedura termina quando non è più possibile aggiungere punti ai cluster trovati.

1.2.11. Algoritmi genetici

Sono particolari algoritmi studiati per guidare il processo di machine learning nella ricerca dei modelli migliori in un certo ambito di possibili soluzioni. Tale categoria si riferisce al campo dell'evoluzione biologica in cui una generazione (di modelli) passa le proprie caratteristiche alla successiva al fine di selezionare le migliori.

Le reti neurali possono utilizzare questo tipo di algoritmi per la gestione dello strato nascosto.

1.2.12. Combinazione di classificatori

La combinazione di classificatori si è rivelata essere una tecnica molto valida nei contesti in cui vengono scelti metodi di classificazione instabili, come nel caso di alberi o reti neurali: i loro risultati sono strettamente legati all'insieme di stima, la cui variazione anche marginale può comportare una diversa costruzione del modello, pur

mantenendo lo stesso tasso di errore di previsione. Tuttavia il classificatore costruito con tale tecnica risulta molto competitivo rispetto a tecniche più sofisticate, come attestano molti test presenti in letteratura.

Bagging

Sviluppato da Breiman (1996), tale algoritmo si basa sull'idea di combinare diversi modelli stimati su insieme di dati differenti: la previsione di un'osservazione risulta essere la classe più frequentemente selezionata dai vari modelli. Tale criterio è detto voto di maggioranza: ogni modello viene visto come un elettore che assegna il suo voto ad una delle possibili classi.

Più in dettaglio, dall'insieme di stima Z vengono creati *m* nuovi campioni mediante la procedura di *bootstrap*: ciascun campione Z_j* è ottenuto mediante un campionamento casuale semplice con reinserimento tra tutte le possibili osservazioni. Da ciascun insieme viene costruito un nuovo classificatore C_j*: il modo più semplice per combinare i risultati di classificazione è utilizzare la media aritmetica, ossia

$$C_{bag}(x)=\frac{1}{m}\sum_{j=0}^{m} C_j*(x)$$

Qualora esistano due possibili classi di appartenenza, il classificatore associa l'osservazione *x* (vettore dei attributi) alla classe 0 oppure 1 a seconda che il valore del classificatore sia inferiore o superiore al valore soglia ½.

Questa procedura può essere utilizzata anche per la previsione di variabili continue: in tal caso non è necessario ricorrere al voto di maggioranza, ma semplicemente utilizzare il valore restituito dalla media aritmetica.

Bumping

Analogamente al bagging, tale metodo utilizza la procedura di bootstrap per ottenere dall'insieme di stima Z un numero *m* di nuovi campioni di dati Z_j* su cui modellare i classificatori C_j*, con j = 1,2,...,*m*. La particolarità del metodo consiste nel selezionare il classificatore che risulta avere il minimo errore in termini di previsione rispetto all'insieme di stima Z originario.

Boosting

Il boosting è una procedura sviluppata da Freund e Shapire (1997) al fine di combinare i risultati di *m* classificatori C_j* costruiti da insiemi di osservazioni Z_j* opportunamente selezionati dall'insieme di stima Z. La particolarità di tale procedura consiste nella modalità con cui viene creato il generico insieme Z_j*. Ad ogni osservazioni x_i dell'insieme di stima Z viene associato un peso w_i: inizialmente valorizzata con 1, la quantità w_i rappresenta la probabilità di entrare nel campione. Una volta costruito l'insieme Z_j* e il classificatore C_j*, vengono aggiornati i pesi di ciascuna osservazione aumentando il valore nel caso di classificazione errata o diminuendo nel caso in cui la classificazione sia corretta. In tal modo le osservazioni classificate in maniera errata hanno una maggior probabilità di entrare nell'insieme di stima del prossimo classificatore.

Foreste casuali

Le foreste casuali sono un algoritmo per il miglioramento degli alberi decisionali: la strategia consiste nella combinazione di più alberi differenti nelle modalità di crescita. L'idea è di far crescere interamente ciascun albero limitando la creazione di un nuovo nodo con l'analisi di un numero prefissato di attributi scelti a caso tra quelli disponibili: l'insieme degli alberi così creati (detta appunto foresta) sarà combinata al fine di selezionare i nodi migliori, riducendo così in

maniera più semplice il numero di attributi importanti per la previsione.

2. WEKA

Weka (*Waikato Environment for Knowledge Analysis*) è un software sviluppato dall'Università di Waikato nella Nuova Zelanda: il progetto è nato sotto l'influenza governativa al fine di essere il punto di riferimento delle tecniche di machine learning per l'economia nazionale. La propaganda del progetto dichiara

> *The programme aims to build a state-of-art facility for developing techniques of machine learning and investigating their application in key areas of the New Zealand economy. Specifically we will create a workbench for machine learning, determine the factor that contribute towards its successful application in the agricultural indiustries, and develop new methods of machine learning and ways of assessing their effectiveness.*

Questo aspetto marcatamente nazionalistico è intuibile dal nome stesso: weka è il nome di un tipico uccello neozelandese che rappresenta la mascotte del progetto.

Fondato da Ian Witten, il progetto iniziò alla fine del 1992; si decise di adottare il linguaggio TCL/TK, sebbene molti algoritmi fosse scritti in linguaggio C. La prima versione fu pubblicata nel 1994: rimase però confinata all'interno dell'università dal momento che il software risultò altamente instabile. Nell'ottobre del 1996 fu rilasciato Weka 2.1, seguito nel luglio del 1997 dalla versione 2.2: il prodotto si dimostrò molto più maturo e arricchito di nuovi algoritmi. Tuttavia agli inizi del 1997 venne presa la decisione di rendere il software multipiattaforma e si iniziò il porting in linguaggio Java, denominando il nuovo branch di sviluppo JAWS (*JAva Weka System*). Nel maggio del 1998 fu rilasciata l'ultima versione in TCL/TK, Weka 2.3: l'anno seguente fu pubblicato Weka 3.1, interamente scritto in Java.

Senza dubbio la riscrittura del codice sorgente in linguaggio Java ha portato notevoli vantaggi:

- **Portabilità**: Java è un linguaggio interpretato, nel senso che il codice generato dal compilatore non viene eseguito direttamente dal sistema operativo quanto da uno strato intermedio, detto macchina virtuale, che traduce le operazioni in linguaggio macchina. Dunque per poter eseguire il programma Weka è necessario che sia installato nel computer l'ambiente di esecuzione Java (JRE, *Java Runtime Environment*) di cui la *Java Virtual Machine* ne è la componente principale. Questa peculiarità permette ai programmi di essere multipiattaforma: infatti l'unica componente a cambiare è la JRE installata nel sistema operativo, a differenza dei programmi che restano invariati.

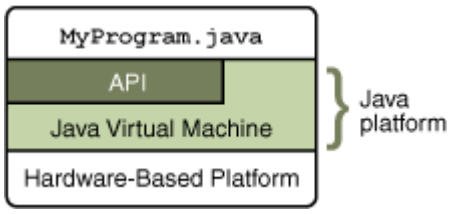

Figura 2.1. La JVM come strato intermedio tra sistema operativo e programma

- **Gestione della memoria**: dal momento che un programma Java viene eseguito all'interno di una macchina virtuale, la gestione della memoria è controllata e gestita da un opportuno modulo, detto *Garbage Collection*: tale componente gioca un ruolo fondamentale per la robustezza delle applicazioni, visto che impedisce spiacevoli inconvenienti derivati dall'errata allocazione o liberazione della memoria.
- **Linguaggio object-oriented**: essendo un linguaggio di ultima generazione, Java è concepito per rispondere ad una programmazione orientata agli oggetti. A differenza della programmazione procedurale in cui un sorgente viene visto

come una sequenza di operazioni, l'approccio orientato agli oggetti cerca di descrivere la realtà a componenti ben definiti, dotati di proprie caratteristiche (*attributi*) e funzionalità (*metodi*). In Java si utilizza il termine *classe* al posto di oggetto: nel proseguo della relazione i due termini verranno utilizzati in maniera differente a seconda che si intenda l'astrazione della realtà (oggetto) o l'implementazione in Java (classe).

- **Documentazione**: essendo piuttosto recente, la definizione del linguaggio Java implementa delle caratteristiche che spesso sono state lacunose in altri contesti, come nel caso della documentazione. A questo scopo è disponibile nel pacchetto destinato ai programmatori (JDK, *Java Development Kit*) il comando *javadoc* che genera automaticamente la documentazione in HTML a partire dagli stessi sorgenti oggetto di compilazione: questo meccanismo facilita notevolmente la stesura e l'aggiornamento dei manuali operativi.
- **Community**: grazie alla semplicità del linguaggio, alla portabilità e ad una marcata aspirazione open source, col passare degli anni sempre più persone, organizzazioni e società hanno attivamente operato al fine di offrire più o meno gratuitamente soluzioni scritte in linguaggio Java.
- **Standard**: proprio per coordinare gli sforzi dell'intera community, è nata l'esigenza di adottare degli standard per tracciare le linee guida con cui implementare personali soluzione a problemi comuni. Ad esempio, per quanto riguarda l'accesso ai dati contenuti in un database è stato creato lo standard JDBC (*Java DataBase Connector*): esso definisce un'interfaccia comune di astrazione delle operazioni di accesso ai dati rispetto alle diverse implementazioni di ciascun database.

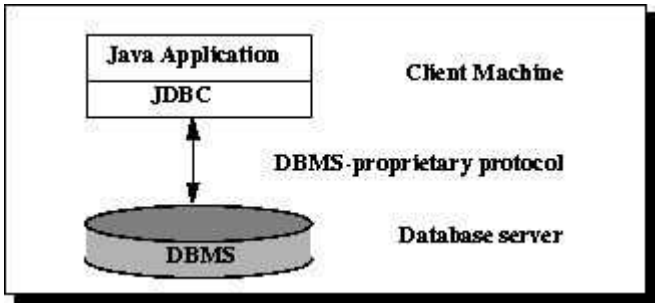

Figura 2.2. Architettura JDBC

Analogamente, il codice di Weka è stato scritto rispettando le indicazioni descritte nello standard JSR-000073, detto più semplicemente *Java Data Mining API Specification Request*: tale documento [2.01] fissa la struttura che devono avere le applicazioni di data mining nella piattaforma J2EE (*Java 2 Enterprise Edition*) al fine di garantire una facile ed efficace integrazione tra una generica applicazione e i diversi motori di data mining.

2.1. Architettura

L'architettura di Weka 3 è facilmente comprensibile grazie alle peculiarità del linguaggio Java: infatti la possibilità di produrre documentazione a partire dai sorgenti del programma permette di conoscere in maniera semplice e dettagliata la struttura logica su cui è sviluppato il software. Dunque il mezzo conoscitivo più immediato consiste proprio nella consultazione della documentazione in linea (HTML) offerta insieme al software: gli sviluppatori di Weka consigliano di familiarizzare con tale strumento per poter apprendere autonomamente eventuali future modifiche o novità future.

Prima di procedere alla presentazione dell'architettura di Weka, è necessario introdurre due importanti concetti tipici del linguaggio Java: la suddivisione dei sorgenti in package e il meccanismo dell'ereditarietà.

In Java ciascun oggetto appartiene ad un pacchetto, detto *package*, che rappresenta un'unità logica ben precisa: nel caso di Weka, la radice è il package *weka* mentre i figli rappresentano tutte le varie componenti, eventualmente suddivise in successive diramazioni specializzate (ad esempio gli algoritmi di classificazione ad albero sono contenuti nel package *weka.classifiers.trees*). Questa caratteristica facilita una strutturazione ordinata degli oggetti che si ripercuote anche in una semplicità nella gestione dei sorgenti: infatti la relazione tra package e classi viene mantenuta nel file system con la creazione di cartelle contenenti i file sorgenti di ciascuna classe.

Il concetto di ereditarietà è tipico dei linguaggi *object-oriented*: l'idea è che oggetti simili possano essere rappresentati in modo tale da condividere quelle particolarità che risultano essere comuni. Gli sviluppatori di Weka hanno usato in maniera massiccia tale tecnica, semplificando notevolmente il codice. Ad esempio tutti gli algoritmi di classificazione estendono l'oggetto *weka.classifiers.Classifier* che contiene due metodi basilari: *buildClassifier* per la creazione del modello a partire da un insieme di dati e *classifyInstance* per generare la previsione associata ad un'osservazione. Siccome le tecniche di valutazione dei risultati di previsione, quali ad esempio la matrice di confusione, sono indipendenti dal tipo di algoritmo usato, è sufficiente invocare il metodo *classifyInstance* di un generico oggetto *weka.classifiers.Classifier* per ottenere la previsione da verificare. E' evidente come quest'approccio ottimizzi la scrittura del codice: l'implementazione di un nuovo algoritmo di classificazione non partirà da zero, ma sfrutterà l'oggetto *weka.classifiers.Classifier* in modo da integrarsi perfettamente con gli strumenti già esistenti.

2.1.1. Oggetti di base e utilità

In questo paragrafo vengono identificati i pacchetti che contengono le classi fondamentali dell'intero software: in particolare

- **package *weka.core***: come si intuisce dal nome, il package contiene tutte le classi che costituiscono il cuore: ad esempio la definizione del dataset rappresentato dalla classe *weka.Instances* e definito come insieme di istanze della classe *weka.Instance* (la singola osservazione), le distribuzioni di probabilità definite nella classe *weka.core.Statistics* e l'insieme di tutti gli oggetti per la lettura e salvataggio dei dataset da diverse fonti raggruppati nel package *weka.core.converters*.
- **package *weka.estimators*:** contiene l'insieme di classi che rappresentano l'implementazione di diversi stimatori di probabilità (Poisson, Normale, Mahalanobis, ecc...)

2.1.2. Pre-processing

In questo paragrafo vengono presentati i pacchetti in cui trovare una vasta serie di strumenti per la preparazione e trasformazione dei dati in vista della successiva analisi: vista la marginale importanza nel contesto della relazione, si rimanda alla documentazione in linea per una lista dettagliata.

- **package *weka.filters***: contiene tutte le classi di filtro, inteso come manipolazione dei dati contenuti nel dataset: ogni oggetto rappresenta uno strumento da applicare in maniera automatica (*unsupervised*) oppure interattiva (*supervised*). A loro volta queste due categorie di approccio ai dati sono suddivise a seconda che l'applicazione avvenga su un singolo attributo (*attribute*, dal nome della classe *weka.core.Attribute* che rappresenta una variabile associata ad un'osservazione) o sull'intero insieme degli attributi della singola osservazione (*instance*, dal nome dell'oggetto *weka.core.Instance* che rappresenta l'insieme dei valori degli attributi).

  Ad esempio per aggiungere una nuova variabile con la

trasformazione logaritmica della variabile "peso", si dovrà applicare all'intero dataset la classe *weka.filters.unsupervised.attribute.AddExpression* opportunamente configurato; per creare un dataset più piccolo con osservazioni casualmente selezionate rispetto al dataset originale si userà la classe *weka.filters.unsupervised.instance.Resample*.

- **package *weka.attributeSelection***: in questo pacchetto sono raggruppati tutti gli algoritmi per la selezione o valutazione delle relazioni presenti tra gli attributi del dataset, in modo da supportare l'utente nella scelta degli attributi più importanti su cui basare l'analisi.

  Ad esempio la classe *weka.attributeSelection.PrincipalComponents* implementa l'analisi delle componenti principali, tecnica largamente utilizzata in statistica.

- **package *weka.datagenerators***: contiene classi per la costruzione di dataset con dati casualmente generati da utilizzare nella fase di test degli algoritmi: ad esempio la classe *weka.datagenerators.RDG1* per i test sull'induzione di regole e *weka.datagenerators.BIRCHCluster* per la verifica degli algoritmi di clustering.

2.1.3. Algoritmi di machine learning

In Weka tutte le tecniche e gli algoritmi sono catalogati come machine learning, sebbene molti strumenti siano stati sviluppati in ambito statistico: sono contenuti in package che riflettono la suddivisione canonica, ossia

- **package *weka.associations***: contiene tutte le classi che risolvono il problema dell'associazione;

- **package *weka.classifiers***: contiene tutte le classi che implementano algoritmi di classificazione; secondo gli autori questa parte è stata sviluppata con maggior cura perché ritenuta più interessante;
- **package *weka.clusterers***: raggruppa tutte le classi che affrontano il clustering.

2.1.4. Intefacce grafiche

Weka è stato concepito per essere facilmente usabile: lo sforzo di sviluppare interfacce semplici ed intuitive è davvero apprezzabile, sebbene talvolta sia limitante rispetto alle potenzialità del prodotto. E' possibile scegliere diversi tipi di GUI (*Graphics User Interface*) attraverso cui sfruttare diverse caratteristiche di Weka: un comodo menù di selezione è proposto all'avvio del programma nel caso sia eseguito direttamente dal JAR di rilascio.

- **Simple CLI**: permette l'esecuzione degli algoritmi da linea di comando, da cui appunto CLI (*Command Line Inteface*). Tale modalità simula l'esecuzione dell'algoritmo prescelto come fosse un programma a parte: l'utilizzo necessita di una buona conoscenza dell'implementazione del particolare dell'algoritmo su Weka in modo da inserire correttamente gli eventuali valori di configurazione opzionali.

  Sebbene tale modalità sia nata per esigenze di test in fase di sviluppo, è certamente utile avere la possibilità di eseguire un algoritmo da linea di comando in modo da non dipendere necessariamente da interfacce grafiche che costringano l'iterazione con l'utente. Ad esempio è possibile delegare l'esecuzione dell'algoritmo a programmi di schedulazione esterna o, più semplicemente, utilizzando degli script di batch.
- **Explorer**: rappresenta l'interfaccia grafica di riferimento per la

fase di analisi. E' possibile accedere a tutte le potenzialità tramite un semplice click del mouse. Si presenta all'utente come un insieme di pagine sovrapposte, ciascuna delle quali offre le varie fasi di analisi: la preparazione dei dati, la classificazione, il clustering, l'associazione, la selezione automatica degli attributi e la visualizzazione grafica dei dati.

Nella prima pagina *Preprocess*, l'utente è guidato nella selezione dei dati da utilizzare per l'analisi: le fonti supportate sono file locali e remoti (nei formati ARFF, C45 e CSV) oppure tramite l'esecuzione di una query di selezione (SELECT) su un database accessibile via JDBC. Una volta importati i dati, il dataset è pronto per essere eventualmente modificato in modo manuale o automatico mediante l'applicazione dei filtri. La pagina è composta da tanti pannelli con le informazioni salienti del dataset di analisi: la numerosità, la dimensionalità, il numero di frequenze e la distribuzione di ciascun attributo.

La pagina *Visualize* si propone come strumento per l'analisi grafica: grafici bidimensionali ottenuti mediante tutte le possibili combinazioni tra le variabili con la possibilità di effettuare zoom e selezionare singole unità presenti in ciascun grafico..

Le restanti pagine aiutano l'utente nell'applicazione delle varie tecniche di data mining al dataset. La struttura di ciascuna pagina è essenzialmente la stessa: i possibili algoritmi tra cui scegliere, le opzioni di configurazione dell'algoritmo, i pulsanti per attivare/terminare l'elaborazione dell'algoritmo prescelto ed infine la finestra con l'output finale del modello costruito accompagnato dalle valutazioni sulla bontà dei risultati ottenuti.

- **Experimenter**: questa interfaccia grafica unita al pacchetto *weka.experiment* si propone come soluzione di rapida

configurazione al problema dell'esecuzione sincrona (utilizzo di più thread) di più esperimenti: l'obiettivo è quello di applicare contemporaneamente diversi algoritmi di data mining su uno stesso dataset di ingresso e di poter analizzare successivamente i risultati ottenuti. L'idea è quella di offrire all'utente una componente con cui poter schedulare verifiche di modelli sviluppati precedentemente su dataset di grandi dimensioni: la differenza in termini di numerosità dei dati e carico computazionale richiedono un'attività in background e una gestione centralizzata dei risultati dell'elaborazione. A conferma di questo, gli sviluppatori stanno ancora lavorando sulla possibilità di distribuire i test da eseguire su più computer contemporaneamente in modo da ottimizzare al meglio i tempi di attesa dell'elaborazione.

- **KnownledgeFlow**: è uno strumento grafico per la modellazione del cosiddetto *data-flow*, ossia la sequenza di operazioni successive di cui si compone l'analisi. L'obiettivo consiste nel favorire un approccio sistematico al problema: l'utente compone uno schema mediante *drag-and-drop* di icone che individuano una certa operazione (sostanzialmente gli oggetti di cui si compone Weka, come ad esempio la lettura dei dati dal database) e la definizione della successione delle operazioni da effettuare tramite collegamenti grafici tra icone. I vantaggi di un tale approccio sono molteplici: da un parte il linguaggio visivo possiede una maggiore forza comunicativa, dall'altra permette di descrivere in maniera semplice e chiara l'approccio la procedura di analisi; operativamente l'utente ha la possibilità di modificare e salvare diverse soluzioni.

  Tale approccio segue quello offerto da tutti i più importanti prodotti commerciali, quali *Enterprise Miner* di SAS, *Insightful Miner* di Mathsoft/S-Plus, *Clementine* di SPSS, *Darwin* di

Oracle, *KnowledgeSTUDIO* di Angoss e *Intelligent Miner* di IBM.

- **ArffViewer**: è uno strumento per la visualizzare e la modifica dei dati provenienti da un file di tipo ARFF, il formato nativo di Weka con cui salvare i dati. L'*Attribute Relation File Format* è semplicemente un file CSV (*Comma Separated Values*) preceduta da un'intestazione contenente delle informazioni sui dati.

```
% Dati provenienti da ftp://ftp.ics.uci.edu/pub/machine-learning-databases/autos
@relation 'Caratteristiche di 206 modelli di auto importati negli USA nel 1985'

@attribute 'marca' {alfa-romeo,audi,bmw,chevrolet,dodge,honda,isuzu,jaguar,mazda,mercedes-benz}
@attribute 'alimentazione' {benz,diesel}
@attribute 'aspirazione' {std,turbo}
@attribute 'carrozzeria' {convertible,hatchback,sedan,wagon,hardtop}
@attribute 'trazione' {rwd,fwd,4wd}
@attribute 'posiz.motore' {front,rear}
@attribute 'dist.assi' REAL
@attribute 'lunghezza' REAL
@attribute 'larghezza' REAL
@attribute 'altezza' REAL
@attribute 'peso' REAL
@attribute 'cilindrata' REAL
@attribute 'compressione' REAL
@attribute 'HP' REAL
@attribute 'giri.max' REAL
@attribute 'percorr.urbana' REAL
@attribute 'percorr.strada' REAL
@attribute 'N.cilindri' REAL

@data
alfa-romeo,benz,std,convertible,rwd,front,225,428.8,162.8,124,1155.8,2.1303,9,111,5000,8.928,11.479,4
alfa-romeo,benz,std,convertible,rwd,front,225,428.8,162.8,124,1155.8,2.1303,9,111,5000,8.928,11.479,4
alfa-romeo,benz,std,hatchback,rwd,front,240,434.8,166.4,133.1,1280.5,2.4908,9,154,5000,8.078,11.053,6
audi,benz,std,sedan,fwd,front,253.5,448.6,168.1,137.9,1060.1,1.7862,10,102,5500,10.203,12.754,4
audi,benz,std,sedan,4wd,front,252.5,448.6,168.7,137.9,1281,2.2286,8,115,5500,7.652,9.353,5
audi,benz,std,sedan,fwd,front,253.5,450.3,168.4,134.9,1137.2,2.2286,8.5,110,5500,8.078,10.628,5
audi,benz,std,sedan,fwd,front,268.7,489.5,181.4,141.5,1290,2.2286,8.5,110,5500,8.078,10.628,5
audi,benz,std,wagon,fwd,front,268.7,489.5,181.4,141.5,1339.9,2.2286,8.5,110,5500,8.078,10.628,5
audi,benz,turbo,sedan,fwd,front,268.7,489.5,181.4,142,1399.8,2.1467,8.3,140,5500,7.227,8.503,5
audi,benz,turbo,hatchback,4wd,front,252.7,452.6,172.5,132.1,1384.8,2.1467,7,160,5500,6.802,9.353,5
bmw,benz,std,sedan,rwd,front,257,449.1,164.6,137.9,1086.4,1.7698,8.8,101,5800,9.778,12.329,4
```

Figura 2.3: Esempio di file in formato ARFF

Il file inizia con una riga contenente il tag @relation, che indica il nome o la descrizione del dataset. Seguono tante righe precedute dal tag @attribute quanti sono gli attributi di ciascuna osservazione: per ogni attributo è specificato il nome e il tipo come elenco dei possibili valori se categoriale, real se numerico, string per testo libero e date se data (seguito

dall'eventuale formato se diverso da quello ISO). La sezione dedicata alle osservazioni è segnalata dalla riga con il tag @data. In qualsiasi punto del file è possibile inserire delle righe di commento avendo cura di inserire il carattere % all'inizio di ciascuna riga. Infine i valori mancanti sono identificati dal carattere ?.

Esiste piccola variante del formato ARFF, detta *sparse ARFF*, in cui i valori a zero dei dati possono essere omessi. Ogni osservazione è racchiusa tra parentesi graffe ed ogni valore dell'attributo deve essere preceduto dal progressivo dell'attributo stesso: gli attributi senza valore vengono automaticamente valorizzati con zero. Questa particolarità permette una maggiore semplicità nella descrizione dei dataset relativi all'analisi di regole associative: se un'osservazione rappresenta una transazione e gli attributi individuano il nome di un prodotto di un supermercato (possono essere migliaia!), risulta molto più semplice e intuibile esprimere solo le quantità dei prodotti acquistati in una particolare transazione, ignorando tutti gli altri prodotti non presenti nel carrello della spesa.

2.2. Algoritmi di data mining disponibili

Weka implementa numerosi algoritmi di machine learning: per un elenco esaustivo si faccia ancora una volta riferimento alla documentazione in linea. Di seguito è proposta una tabella riepilogativa che mette in relazione ciascun algoritmo con la tecnica implementata e il nome della classe Java che lo rappresenta.

Tecnica	Algoritmo	Package e nome della classe
Associazione		
Rule Induction	APriori	weka.association.APriori

Tecnica	Algoritmo	Package e nome della classe
Classificazione		
Analisi discriminante	Semplice	weka.classification.bayes.naiveBayes weka.classification.bayes.naiveBayesSimple weka.classification.bayes.naiveBayesMultinomial
	Rete Bayesiana	weka.classification.bayes.BayesNet
Regressione	Lineare	weka.classification.function.LinearRegression
	Logistica	weka.classification.function.Logistic
	Locale	weka.classification.function.LWR
Reti neurali	Percettrone multi strato	weka.classifiers.functions.MultilayerPerceptron
Instance-based / Lazy Learning	k-NN	weka.classification.lazy.Ibk
	K*	weka.classification.lazy.KStar
	Lazy semi-naïve Bayes learning	weka.classification.lazy.LBR
Alberi	Decision stump	weka.classification.trees.DecisionStump
	ID3	weka.classification.trees.ID3
	C4.5	weka.classification.trees.J48
	M5	weka.classification.trees.M5Prime
	Logistic Model Tree	weka.classification.LMT
Combinazione di classificatori	Bagging	weka.classification.Bagging
	Boosting	weka.classification.meta.AdaBoostAB weka.classification.meta.LogitBoost weka.classification.meta.MultiLogitAB
	Random Forest	weka.classification.trees.RandomForest
Clustering		
Instance-based / Lazy Learning	Coweb	weka.clusteres.Cobweb
	EM	weka.clusteres.EM
	Density Based Clustering	weka.clusteres.DBScan
	K-medie	weka.clusteres.SimpleKMeans

3. R

R ebbe origine da un progetto di Ross Iheka e Robert Gentlmen dell'Università di Auckland: l'idea era lo sviluppo di una piattaforma per l'analisi statistica liberamente utilizzabile. Ben presto acquistò il nomignolo “GNU S” perché si poneva in netta contrapposizione con il più celebre linguaggio statistico commerciale S, sviluppato da AT&T Bell Laboratories. R si propone nel panorama open source come un potente ambiente di analisi computazionale e grafica, dotato di un completo linguaggio con cui l'utente interagisce con il motore interno. Il progetto garantisce una buona compatibilità con il linguaggio S (e la sua versione commerciale S-Plus): questo aspetto ha comportato senza dubbio gran parte del successo di R che, grazie alla licenza d'uso gratuita, ha attirato specialisti dell'area scientifica provenienti da ambienti di ricerca in cui S risultava lo strumento di analisi per antonomasia. Oggigiorno R gode di una vasta comunità di specialisti (non solo del settore statistico) che utilizzano e spesso contribuiscono alla sua crescita mediante la condivisione di proprie soluzioni, disponibili sotto forma di *package*: a supporto di questa realtà è stato costituito il CRAN (*Comprehensive R Archive Network*), con l'obiettivo di organizzare al meglio tutte le risorse disponibili (sorgenti, documentazione, manuali, package) e di renderle facilmente accessibili in Internet.

Sebbene il linguaggio sia intuitivo e gli strumenti di analisi siano numerosi e molto potenti, la condizione necessaria per utilizzare al meglio R è che l'utente abbia una buona conoscenza sia delle potenzialità del software che delle modalità di analisi.

3.1. Architettura

R è scritto in C ed è disponibile per tutti i principali sistemi operativi: Microsoft Windows, Macintosh OS X e UNIX. Il suo sviluppo è

gestito dal cosiddetto *R Development Core Team*, costituito da programmatori ed esperti scientifici. La struttura principale di R può essere scomposta nei seguenti moduli:

- **motore**: è lo strato più basso del software in cui viene gestito l'aspetto computazionale (precisione di calcolo) e l'interpretazione dei comandi impartiti dal linguaggio con cui l'utente interagisce con il sistema. Gli sviluppatori hanno curato la possibilità di poter integrare il motore R in altri applicativi mediante una cosiddetta versione *embedded*, costituita da una particolare libreria dinamica: si rimanda all'appendice per un maggiore dettaglio tecnico.

- **linguaggio**: R utilizza un linguaggio molto semplice, adopera le notazioni standard dell'algebra (ad esempio `1+1`, `3^4`), non ha bisogno di file esterni da includere, le dichiarazioni sono implicite, non ci sono puntatori e le stringhe possono essere trattate e manipolate direttamente: in poche parole, non serve essere dei programmatori per poterlo utilizzare.

  In R tutte le componenti del linguaggio sono oggetti: spesso l'output di un comando è un oggetto il cui contenuto può essere anche molto complesso: l'utente può accedere alle informazioni contenute richiamando gli opportuni attributi. La gestione della memoria associata agli oggetti è garantita da un *garbage collection*, analogamente al linguaggio Java.

  Dal punto di vista implementativo, R è molto simile al Lisp (*List Processor*) che rappresenta uno dei primi linguaggi di programmazione, ideato nel 1958 da John McCarthy come linguaggio formale per studiare le equazioni di ricorsione in un modello computazionale. La popolarità del linguaggio fu subito immediata per il grande interesse suscitato nell'ambito dell'Intelligenza Artificiale: senza dubbio il successo è dovuto

alla semplicità di un linguaggio con poche regole di sintassi e alla possibilità di generare e compilare al volo codice Lisp. Tuttavia questo successo portò alla definizione di numerosi dialetti come InterLisp, Maclisp, ZetaLisp, and Franz Lisp: solo nel 1994, ANSI pubblicò la versione standard di Lisp, chiamata *ANSI X3.226-1994 Information Technology Programming Language Common Lisp* o semplicemente Common Lisp. Sebbene oggi si preferisca utilizzare linguaggi di più alto livello come Java o Python, esistono molti programmi come Emacs e i sistemi CAD (ad esempio AutoCAD e intelliCAD) che impiegano Lisp come linguaggio di estensione.

- **grafica**: per R la grafica è uno dei requisiti più importanti. Questo aspetto è molto curato tanto da unire efficacia ed eleganza: l'analisi grafica deve necessariamente supportare l'analisi computazionale e aiutare l'utente a intuire informazioni particolari, come nel caso della distribuzione dei dati. A parte i classici istogrammi e boxplot, è possibile tracciare funzioni, disegnare punti, inserire notazioni di testo o legende: tutto questo con grande precisione e qualità. E' possibile in qualsiasi momento stampare o salvare su file i grafici eventualmente prodotti.

- **estensioni (package)**: il grandissimo pregio di R consiste nel poter ampliare il programma con plug-in che si integrano perfettamente nel sistema e nel linguaggio: tali estensioni sono dette *package*. Questo approccio permette di svincolare il team di sviluppo di R dalla responsabilità di far crescere il progetto: viceversa sono gli specialisti utilizzatori di R che creano estensioni atte ad affrontare il proprio campo di analisi; una volta consolidato il lavoro, è possibile pubblicare il package. Nella rete CRAN sono disponibili decine e decine di estensioni pronte all'uso: l'offerta spazia dai pacchetti di utilità (ad

esempio connessione ai database), a soluzioni trasversali (ad esempio il package `tree` per gli alberi decisionali e di regressione) o verticali (pacchetti per specifiche aree della biologia). Tuttavia possono presentarsi dei problemi nell'installazione dovuti o alla propria versione di R non compatibile con l'estensione o una dipendenza tra estensioni non soddisfatta.

3.2. Algoritmi di data mining disponibili

Grazie al meccanismo dei package e la fiorente attività della rete CRAN, R può vantare di molte soluzioni provenienti da differenti gruppi di lavoro. Per tale motivo possono esserci implementazioni differenti di una stessa tecnica ad opera di persone diverse.

Tecnica	Algoritmo	Package R
Associazione		
Rule Induction	APriori	arule
Classificazione		
Classificatori Bayesiani	Semplice	MASS
	Reti Bayesiane	sna
Regresione	Lineare	stats
	GLM	stats
	Locale	sm sn
Spline		splines mgcv KernSmooth modreg
GAM		gam
Reti neurali		nnet neural AMORE
Instance-based / Lazy Learning	k-NN	class ipred

Tecnica	Algoritmo	Package R
Alberi	CART	tree rpart party
Combinazione di classificatori	Bagging	adabag
	Boosting	adabag ipred
	RandomForest	party randomForest
Clustering		
Instance-based / Lazy Learning	K-medie	stats cluster

Infine si segnala la presenza del package `RWeka` per poter eseguire gli algoritmi di Weka nell'ambiente R: sebbene l'integrazione sia abbastanza spartana, è possibile configurare gli algoritmi di Weka come se fossero eseguiti da linea di comando.
L'installazione del package è strettamente legata dalla disponibilità del package `rJava`, in modo da eseguire all'interno di R la Java Virtual Machine.

4. Confronto tra R e Weka

Impostare in maniera corretta un confronto tra due applicazioni così differenti è senza dubbio un compito molto delicato: da una parte Weka è stato concepito per affrontare esclusivamente problemi di data mining, dall'altra R si propone come un ambiente completo e facilmente estendibile per problemi di analisi statistica di qualsiasi genere. Usando una terminologia informatica, Weka è definito prodotto verticale (specifico per una precisa area di applicazione) mentre R è un prodotto orizzontale (valido per tanti tipi di problemi).

Al fine di rendere la trattazione il più coerente possibile con l'intento di individuare un prodotto per data mining, tale capitolo è suddiviso in due parti: la prima propone un paragone teorico sulla base dei requisiti ritenuti nei capitoli precedenti indispensabili per un prodotto di data mining; la seconda parte presenta un confronto delle prestazioni in termini di velocità e costo di elaborazione dei due prodotti, misurate durante la creazione e la verifica di particolari modelli.

4.1. Analisi dei requisiti

In questo primo paragrafo viene analizzato come Weka e R rispondono ad una serie di requisiti ritenuti basilari per un prodotto di data mining: la loro scelta è stata effettuata sulla definizione stessa di data mining presentata in questa relazione, e in particolare sulle varie fasi di cui si compone.

- **Accesso ai dati**: ovviamente il primo requisito di un prodotto di analisi dei dati è la flessibilità con cui è possibile importare i dati. Come già affermato in precedenza, non è compito del software di data mining creare e gestire data wearehouse oppure disporre di strumenti per la sintesi di dati provenienti da database di produzione: l'insieme dei dati di analisi devono

essere già pronti e disponibili in un database o un file opportunamente strutturato.

Weka può importare dati disponibili in file di formato CSV, ARFF e C4.5. Per quanto riguarda l'accesso ai dati contenuti in un database, l'implementazione di Weka beneficia dello standard JDBC proprio del linguaggio Java: tuttavia, dal momento che le API di programmazione JDBC offrono una generalizzazione delle comuni operazioni su base dati, è necessario includere nell'ambiente runtime l'opportuno driver con l'implementazione nativa di comunicazione con il particolare database. Ad esempio nel caso in cui i dati siano contenuti in un database MySQL, è necessario inserire nella classpath il file JAR con il driver relativo a tale database (ad esempio `mysql-connector-java-3.0.9-stable-bin.jar`).

Grazie alla potenzialità del linguaggio, R può importare dati contenuti in file CSV in maniera più flessibile rispetto a Weka: è possibile ad esempio definire il tipo di separatore e la stringa che identifica i valori mancanti. Sebbene il formato ARFF non sia supportato nella versione base di R, sono disponibili nella rete CRAN e nel sito di Weka opportune funzioni che permettono la lettura e scrittura di dataset in tale formato.

Per quanto riguardo l'accesso al database, R dispone di molti pacchetti con differenti soluzioni. Il package `RODBC` utilizza lo standard *Open Database Connectivity* (ODBC) del sistema operativo Microsoft Windows: in tal caso l'ambiente R utilizza una caratteristica particolare che però è assente in altri sistemi operativi. Il package `DBI` fornisce un'interfaccia comune per l'accesso ai database sullo stile JDBC: tuttavia soffre della medesima limitazione di dover disporre del driver relativo al particolare database, con l'aggravante che tale standard è

riconosciuto solo in ambito dei software R e S-plus. Infine si segnala la presenza del package `foreign` che permette la lettura di file creati da altri applicativi di analisi statistica come SAS, Minitab, SPSS e Systat.

La fase di importazione dei dati corrisponde sia per Weka che per R alla copia dell'intero insieme di dati nella memoria del computer: ovviamente nel caso di dataset di grosse dimensioni tale operazione può rivelarsi impossibile in termini di disponibilità di risorse fisiche del computer stesso. A tal riguardo, nel corso dei test effettuati su Weka è stata individuata una leggera divergenza sulla modalità con cui vengono gestiti i dati eventualmente utilizzati durante la fase di verifica del modello: un controllo del codice sorgente ha confermato che solo nel caso di utilizzo degli algoritmi da linea di comando l'insieme di verifica non viene memorizzato interamente, ma letto gradualmente copiando e mantenendo in memoria un sola riga alla volta. Questo comportamento è facilmente spiegabile con il fatto che la modalità da linea di comando accetta solo dataset contenuti in file locali, sui quali è meno rischiosa una lettura progressiva delle informazioni rispetto a risorse remote.

- **Analisi esplorativa dei dati**: come già precedentemente affermato, l'analisi esplorativa dei dati risulta essere la fase più delicata dell'intero processo di data mining. Esistono moltissimi modi di rappresentare graficamente i dati al fine di evidenziare un particolare aspetto di distribuzione di una variabile piuttosto che un'eventuale dipendenza con altre variabili: a tal proposito è possibile ricorrere anche a strumenti come OLAP (*OnLine Analytical Processing*) e ipercubi. Tale premessa vuole dunque sottolineare il fatto che un software di data mining non deve necessariamente coprire tutte le possibili modalità di analisi

grafica.

R risulta essere il prodotto migliore sia per la vasta gamma di modalità con cui generare rappresentazioni grafiche sia per la qualità delle immagini ottenute: le potenzialità del linguaggio permettono inoltre l'interazione con l'analista al fine di evidenziare o aggiungere elementi grafici, ottenendo risultati di gran pregio. Infine la possibilità di estendere l'ambiente mediante l'importazione di pacchetti permette la cooperazione con altri strumenti grafici esterni, come nel caso della grafica tridimensionale di GGobi (http://www.ggobi.org).

Gli strumenti di analisi esplorativa dei dati offerta da Weka risultano essere piuttosto spartani: sono disponibili solo le macro informazioni sulle variabili (come il numero di valori nulli e gli indici di localizzazione) e i tipici grafici di analisi della distribuzione dei dati rispetto alle variabili (istogrammi, diagrammi di dispersione,...).

- **Pre-processing**: la preparazione dei dati da utilizzare nelle procedure di creazione dei modelli corrisponde ad una fase molto limitata in quanto si presuppone che i dati importati dall'applicativo siano già strutturati in maniera coerente con l'analisi da effettuare: la manipolazione dei dati dovrebbe essere giustificata solo dalla necessità di escludere o creare nuove variabili in modo da adattare i dati stessi alle esigenze delle diverse tecniche ed algoritmi di data mining.

  Ancora una volta R si presenta senza dubbio come lo strumento migliore grazie alla flessibilità del linguaggio con cui è agevole effettuare piccole o grandi modifiche ai dati.

  Weka risulta essere penalizzato a causa dell'impiego di filtri prestabiliti: sebbene essi coprano la maggior parte delle operazioni usualmente applicate ai dati, non sempre risultano

essere di immediata configurazione, soprattutto se si utilizza l'interfaccia grafica *Explorer*. Questo aspetto risulta limitante nella prima fase di analisi in cui l'analista "gioca" con i dati al fine di conoscere meglio il fenomeno oggetto dello studio.

- **Sviluppo del modello**: lo sviluppo del modello corrisponde ad una fase molto laboriosa in cui l'analista confronta i risultati ottenuti da più modelli generati dal dataset di stima mediante l'applicazione di diverse tecniche e algoritmi. Lo scopo consiste nell'individuare il modello maggiormente interessante: come già affermato nel primo capitolo, un buon modello non corrisponde necessariamente al "vero" modello che ha generato i dati!

  Weka e R permettono l'applicazione di più algoritmi al medesimo dataset di stima durante la stessa sessione di lavoro: se da una parte Weka dispone della possibilità di salvare il modello creato, dall'altra R dà la possibilità di memorizzare l'intera sessione di lavoro (*workspace*) in modo da poter riprendere l'analisi anche in tempi successivi senza perdere alcuna informazione.

  Per una completa analisi di tale requisito si rimanda al successivo confronto pratico in cui verranno misurati i due applicativi in termini di velocità computazionale e occupazione di memoria a parità di dati ed algoritmo utilizzato.

- **Valutazione del modello**: la valutazione della bontà di un modello si basa essenzialmente sugli errori tra la previsione effettuata dal modello e il relativo valore reale presente nell'insieme di verifica. A partire da questa informazione è possibile utilizzare molti indici o test di verifica di ipotesi sulla confidenza dei risultati ottenuti.

  Weka propone come indici di bontà del modello la somma dei

quadrati degli scarti per le variabili continue e la tabella di errata classificazione con relativa curva di ROC per le variabili discrete.

Anche R dispone delle medesime informazioni di sintesi, sebbene non vi sia una funzione che calcoli automaticamente la curva di ROC. Tuttavia le potenzialità del linguaggio e gli strumenti statistici di R possono fornire molti più elementi di valutazione: le cosiddette diagnostiche grafiche (ad esempio il *diagramma di Anscombe* e il *diagramma quantile-quantile*) forniscono preziose informazioni relative ai residui di stima; l'analisi della varianza e la costruzione di intervalli di confidenza sono altri strumenti che possono rivelarsi fondamentali nella valutazione dell'importanza delle variabili nel modello.

Oltre ai precedenti requisiti prettamente connessi all'analisi mediante data mining, si è voluto verificare altre caratteristiche legate alla particolarità di aver scelto prodotti open source. Sebbene non sia trattato in tale relazione, la natura di software open source implica una serie di conseguenze molto importanti: l'aspetto che più interessa e che verrà affrontato nei seguenti requisiti è misurare quanto e come sia possibile sfruttare le potenzialità e il valore aggiunto di un software di terze parti.

- **Licenza**: ogni software è coperto da una licenza che ne descrive la libertà di utilizzo, copia, modifica e distribuzione. La violazione della licenza espone l'utilizzatore ai provvedimenti giuridici e penali conseguenti alla mancata osservanza dei diritti sul copyright. Attualmente vi è molta confusione sul fenomeno open source, comunemente intenso come qualcosa di gratuito. In realtà all'interno del mondo open source esistono diversi tipi di licenze che salvaguardano in maniera diversa il software, sebbene senza dubbio garantiscano

una maggiore libertà rispetto ai software commerciali.

Sia Weka che R sono distribuiti sotto la licenza *GNU General Public License*, o semplicemente GNU GPL. E' la storica licenza del progetto GNU, la prima iniziativa per la distribuzione di software libero. Essa assicura piena autonomia nell'utilizzo del software a chiunque ne riceva una copia: coloro che accettano le sue condizioni hanno la possibilità di modificare, copiare e distribuire il software con o senza personali modifiche, sia gratuitamente che a pagamento. Tuttavia tale distribuzione del software, anche se eventualmente modificato, deve garantire il rispetto del cosiddetto copyleft, ossia deve essere reso disponibile gratuitamente il codice sorgente ad ogni acquirente, incluse tutte le eventualmente personalizzazioni effettuate.

- **Possibili estensioni**: un requisito fondamentale per un software open source è la possibilità di modificare il codice sorgente al fine di personalizzare o estendere le funzionalità rispetto alla versione originale. Il progetto GNU ha sostenuto con molto vigore l'interazione tra diversi gruppi di lavoro al fine di ottenere software più curato e meno affetto da errori (*bugs*).

  Weka è stato concepito per essere esteso nel modo più efficiente possibile: la scrittura di nuovi algoritmi risulta davvero semplificata grazie alle numerose classi di utilità offerte dall'architettura di Weka. Tuttavia l'integrazione con le interfacce grafiche non è altrettanto immediato: infatti è necessario intervenire manualmente ad aggiungere il riferimento del nuovo algoritmo all'interno delle stesse GUI.

  R è certamente dotato del meccanismo più semplice per aggiungere nuove funzionalità all'ambiente di lavoro: l'installazione dei *package* rappresenta la soluzione vincente

per l'integrazione di nuove estensioni rispetto alle interfacce preesistenti. Inoltre la possibilità di creare dei pacchetti in linguaggio C, C++ e Fortran permette la cooperazione tra algoritmi preesistenti e l'ambiente R. Ad esempio il recente pacchetto `arule` rappresenta l'infrastruttura necessaria per lo scambio delle informazioni di input e output tra l'ambiente R e l'implementazione originaria degli algoritmi Apriori e Eclat di Borget (2003) in linguaggio C. Questo emblematico esempio evidenzia in modo chiaro la differente filosofia su cui poggiano i due progetti: se da una parte R è stato concepito per essere flessibile all'integrazione di altre soluzioni preesistenti, Weka è stato sviluppato per essere piuttosto monolitico proprio perché lo scopo del progetto prevede la riscrittura degli algoritmi in linguaggio Java, sebbene siano disponibili in altri linguaggi o in librerie già compilate di terze parti.

- **Integrazione in altri programmi**: una particolarità molto interessante della filosofia open source consiste nella possibilità di integrare le potenzialità offerte da un altro software in un proprio progetto: in tal modo si evita di sprecare tempo e risorse per la soluzione di un certo tipo di problema o di funzionalità che a sua volta è già stato realizzato da un altro progetto di lavoro. Tuttavia la procedura di integrazione rappresenta un aspetto molto delicato: sebbene esista l'indubbio vantaggio di aver a disposizione una determinata soluzione già pronta, si potrebbe facilmente incappare nel pericolo di dover investire eccessivamente nell'individualizzazione ed implementazione dei punti di comunicazione tra la parte software del proprio progetto e quella esterna.

  Weka risulta essere senza dubbio avvantaggiato grazie all'adozione del linguaggio Java: la programmazione ad oggetti e la facilità di integrare il codice compilato rappresenta senza

dubbio la soluzione più allettante per i progetti che utilizzano Java. Tuttavia in altri scenari l'utilizzo di Weka potrebbe risultare non altrettanto naturale: infatti la creazione dell'ambiente di esecuzione per il codice Java, la cosiddetta Java Virtual Machine, laddove non sia disponibile, risulta essere un'operazione essere molto delicata e critica.

L'integrazione di R risulta sostanzialmente indifferente al tipo di linguaggio usato dal software ospitante. Grazie alla possibilità di compilare il codice sorgente R in modalità *embedded*, tutte le potenzialità di R sono rese pubbliche da una particolare libreria dinamica e facilmente utilizzabili mediante l'invocazione di semplici funzioni (API): si rimanda all'appendice per un approfondimento di carattere tecnico. L'aspetto più delicato è rappresentato dalla componente grafica: siccome non tutti i sistemi operativi gestiscono in maniera nativa l'interfaccia grafica e soprattutto risultano estremamente differenti nell'utilizzano delle componenti grafiche, gli sviluppatori di R consigliano di lasciare al motore grafico di R le ottimizzazioni per il sistema operativo in cui viene eseguito il programma, scoraggiando altresì eventuali soluzioni personali.

4.2. Applicazioni pratiche

Questa seconda parte del confronto riassume i risultati dei test effettuati su Weka e R, applicando agli stessi dati alcuni gli algoritmi utilizzati più di frequente nelle analisi di data mining. La finalità dei test non consiste assolutamente nella valutazione della qualità della previsione dei modelli generati con Weka piuttosto che con R: questo aspetto esula da questa relazione per molti motivi, quali la necessità di confrontare più modelli generati dallo stesso algoritmo su campioni di dati di natura diversa e l'attività di *tuning* dei parametri di configurazione dell'algoritmo stesso. Infatti in letteratura è stato

largamente dimostrato che una certa variante di un algoritmo possa avere risultati del tutto differenti a seconda della tipologia di campione di dati utilizzato, proprio a causa del diverso fenomeno (distribuzione) che ha generato le osservazioni. A tal proposito è ragionevole considerare le implementazioni presenti in Weka e R come varianti di uno stesso algoritmo: possono contenere ottimizzazioni o sotto-procedure diverse per rispondere a particolari criteri di efficienza e qualità del risultato. A rafforzare questa idea è la presenza di parametri di configurazione anche molto diversi tra Weka e R relativamente allo stesso tipo di algoritmo: solo un'approfondita conoscenza dell'implementazione di ciascun algoritmo nei due prodotti permetterebbe un corretto giudizio comparativo delle previsioni ottenute dai modelli generati. Ad esempio la classe Weka per la regressione lineare è configurata in maniera tale da effettuare l'analisi delle componenti principali prima della procedura di stima. Ovviamente il modello, includendo solo le variabili più interessanti, genera previsioni migliori rispetto ad una qualsiasi procedura di stima che utilizzi solo il criterio dei minimi quadrati, così come definito nell'impianto teorico.

Tutte queste considerazioni sono alla base dell'esclusione delle reti neurali dall'insieme degli algoritmi confrontati. Nelle prove eseguite per familiarizzare con le diverse soluzioni presenti nei due prodotti, sono state evidenziate delle profonde differenze di approccio alle rete neurali: da una parte Weka propone classi specializzate nel risolvere una certa problematica (come nel caso del percettrone multistrato), dall'altra tutti i pacchetti R contengono funzioni atte a risolvere i vari aspetti in cui si può scomporre la complessità di una rete neurale. In tale scenario non è possibile effettuare alcun test comparativo senza conoscere in modo chiaro la sequenza di operazioni eseguite dalla particolare implementazione dell'algoritmo in Weka.

L'obiettivo del confronto pratico consiste nell'evidenziare

eventuali differenze di prestazioni nelle fasi di costruzione e verifica di uno stesso modello da parte di algoritmi analoghi di Weka e R: in particolare, durante l'esecuzione dell'algoritmo sono state monitorate le risorse del computer in termini di

- utilizzo CPU, inteso come costo computazionale o di elaborazione dell'algoritmo;
- utilizzo della memoria, inteso come costo dovuto all'allocazione di memoria per l'esecuzione dell'algoritmo;
- durata elaborazione, intesa come tempo impiegato dall'algoritmo per l'esecuzione di una particolare operazione.

Alla fine di ogni test verranno riportati i valori di previsione del modello ottenuto: tuttavia, come già detto, il giudizio sarà sull'eventuale divergenza dei risultati e non sulla qualità del modello stimato in senso assoluto. Più semplicemente, l'interesse consiste nella selezione del prodotto più versatile e performante nell'affrontare i problemi di data mining piuttosto che sul modo in cui li risolve.

Operativamente un test è associato ad un certo tipo di algoritmo che abbia un'implementazione simile in Weka e R: una volta definiti gli insiemi di stima ed eventualmente di verifica come input all'algoritmo, ciascun test è stato suddiviso in tre fasi corrispondenti all'esecuzione dell'algoritmo presente in Weka, in R da console e attraverso la versione *embedded* di R da un programma Java. Non è stata fatta alcuna prova dell'integrazione di Weka in ambiente non Java, visto che è già disponibile con il package `RWeka`.

E' stato scritto un semplice programma in linguaggio Java, utilizzato per l'esecuzione degli algoritmi sia di Weka sia di R in versione *embedded*. Esso garantisce che i passi di analisi impostati nel motore R da un apposito file di comandi siano analoghi a quelli eseguiti dal motore Weka, ossia

1. caricamento dell'insieme di stima in memoria;

2. creazione del modello;
3. caricamento dell'eventuale insieme di verifica in memoria;
4. valutazione del modello sull'insieme di stima o di verifica se disponibile.

Per integrare Weka nel programma di test è bastato inserire il file `weka.jar` (l'archivio in cui sono raccolte tutte le classi di Weka) nella variabile d'ambiente *CLASSPATH* che rappresenta l'insieme dei percorsi in cui la JVM cerca le classi Java da eseguire: il meccanismo è analogo a quello della variabile d'ambiente *PATH* per quanto riguarda i sistemi operativi. Per accedere alle potenzialità di R da Java, il programma di test utilizza una particolare classe con la definizione di un cosiddetto metodo nativo: tramite l'architettura JNI (*Java Native Interface*) l'invocazione di tale metodo richiama un'apposita funzione di una libreria dinamica scritta in C, che costituisce lo strato intermedio tra la parte in Java e il motore R nella versione *embedded*: ulteriori dettagli sono presenti in appedice.

I dati utilizzati nei test sono presi dall'archivio di pubblico dominio *UCI Repository of machine learning databases* dell'Università della California [4.01]. E' stato scelto il dataset *adult* per l'elevata numerosità e dimensionalità campionaria: i due insiemi di stima e verifica contengono i valori (anche nulli) di sei variabili continue e nove discrete, rilevate rispettivamente su 32561 e 16282 individui. In particolare, tre delle variabili discrete supportano un numero di fattori maggiore di 10 e fino ad un massimo di 41 valori diversi: tale aspetto è importante perché si ripercuote in un maggior carico computazione nella creazione dei modelli. La preparazione dei dati è stata effettuata con l'ausilio di R ed è stata articolata in due fasi successive: l'adattamento degli insiemi di stima e verifica al tipo di modello (mediante la conversione di valori o l'eliminazione di variabili non gestite dall'algoritmo) e il salvataggio dei dati in formato CSV.

Successivamente una copia di tali file è stata trasformata in ARFF, mediante l'inserimento della caratteristica testata di tale formato: dunque per ogni algoritmo da testare sono stati creati quattro dataset, rispettivamente per la stima e verifica in R (formato CSV) e in Weka (formato ARFF).

Ciascun test riguarda una particolare tecnica di analisi: sono state scelte la regressione lineare, l'albero decisionale basato sull'informazione, un clustering ottenuto mediante l'algoritmo delle K-medie e le regole associative generate dall'algoritmo Apriori. Nei successivi paragrafi verranno riportati gli esiti di ciascun test effettuato, a sua volta suddiviso nelle tre fasi corrispondenti all'applicazione dell'algoritmo di Weka (con il programma di test), di R (da console) e di R in versione *embedded* invocato da piattaforma Java (con il programma di test). Come riepilogo delle prestazioni di sistema monitorate in ogni fase di test, verranno proposti due grafici che mettono in relazione la durata del test stesso (in ascissa) e rispettivamente il tempo di elaborazione e la memoria utilizzata (in ordinata): in ciascun grafico saranno quindi inserite rette verticali in corrispondenza dell'inizio e fine di ogni passo di analisi in modo da determinare quanti secondi sono stati utilizzati per il caricamento dell'insieme di stima, per la costruzione del modello, per il caricamento dell'insieme di verifica e la valutazione finale del modello. Nel caso dei grafici relativi alla versione R *embedded* da Java, sarà aggiunta un'ulteriore retta verticale per indicare il costo di inizializzazione dell'ambiente R mediante il meccanismo di esecuzione delle librerie dinamiche.

Le prove sono state effettuate su un personal computer dotato di processore *Intel Pentium IV* 3.000 Mhz *dual-core* e 1 GB di RAM: il sistema operativo è Linux *CentOS 4*, la versione libera della distribuzione *Linux RedHat Interprise Edition*, con kernel 2.6.14. Sono state infine utilizzate le versioni 3.4.5 di Weka e 2.3.1 di R.

4.2.1. Test di regressione lineare

Il modello di regressione lineare è stato definito utilizzando come variabile risposta il guadagno di capitale (*capital-gain*) e come variabili esplicative tutti i rimanenti attributi dell'osservazione. La fase di creazione del modello risulta essere volutamente appesantita: infatti si presuppone che le procedure di stima dei parametri non differiscano affatto tra Weka e R per la solida teoria matematica alla base della tecnica utilizzata. Per lo stesso motivo si è scelto di non effettuare alcuna verifica del modello basata su un nuovo insieme di osservazioni: pertanto le operazioni monitorate si limitano al caricamento dei dati e alla costruzione del modello, riducendo a tre il numero delle rette verticali nei grafici riepilogativi.

Per la costruzione del modello in R è stata utilizzata la classica funzione `lm` del pacchetto `stats`: i comandi necessari per il caricamento dell'insieme dei dati e della stima del modello sono stati inseriti in un apposito file. Quest'ultimo è stato prima eseguito dal programma R a console, di cui vengono presentati i grafici relativi al costo di elaborazione in termini di tempo di CPU e utilizzo di memoria.

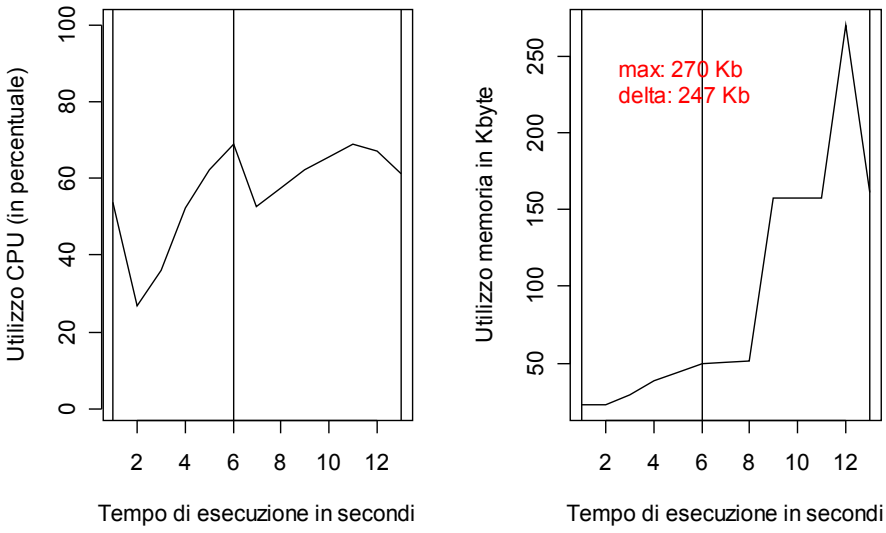

Figura 4.1. Prestazioni computazioni nella creazione del modello lineare con R

Seguono i grafici delle prestazioni misurate durante l'esecuzione dello stesso codice R dal programma di test scritto in Java, utilizzando la versione *embedded* di R .

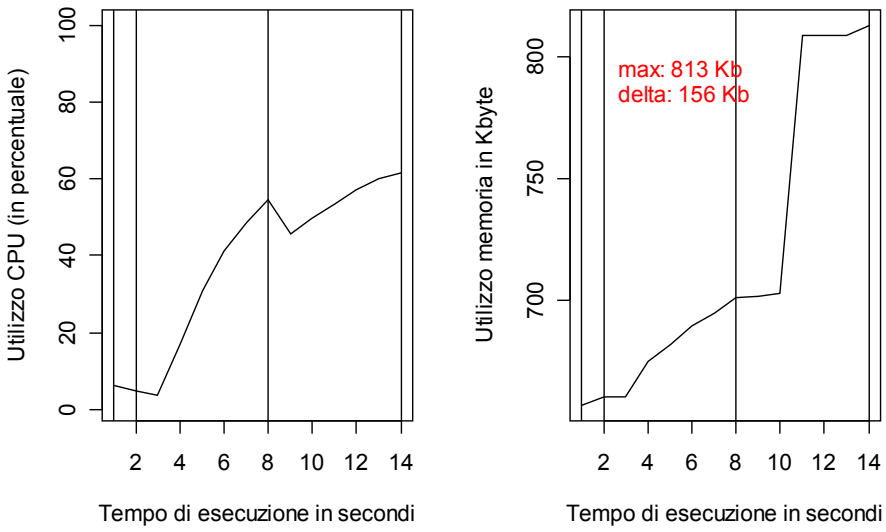

Figura. 4.2. Prestazioni computazioni nella creazione del modello lineare con R da piattaforma Java

La prima considerazione è che non vi è alcuna sostanziale diversità tra R in versione console e *embedded*: l'unica differenza consiste nell'utilizzo della memoria che nel secondo caso è molto superiore, portando il picco massimo da 270 Kbyte a 813 Kbyte. Questo comportamento è dovuto alla presenza della Java Virtual Machine che durante la fase di inizializzazione si appropria di una certa quantità di memoria (657 Kbyte) da utilizzare come *head* del processo Java.

A parte il secondo impiegato dalla versione *embedded* per l'inizializzazione dell'ambiente R all'avvio del processo stesso, la fase di caricamento dei dati e la fase di costruzione del modello hanno impiegato entrambe 5 secondi, utilizzando la maggior parte delle risorse in quest'ultima fase.

La creazione del modello in Weka è stata effettuata mediante la classe `LinearRegression` del package `weka.classifiers.functions`. Al fine di ottenere un

algoritmo del tutto simile a quello proposto da R, è stata adottata una configurazione tale da eliminare tutte le opzioni aggiuntive, quali la selezione automatica delle variabili esplicative da usare nella stima del modello (parametro `-S 1`) e l'eliminazione delle variabili correlate (parametro `-C`).

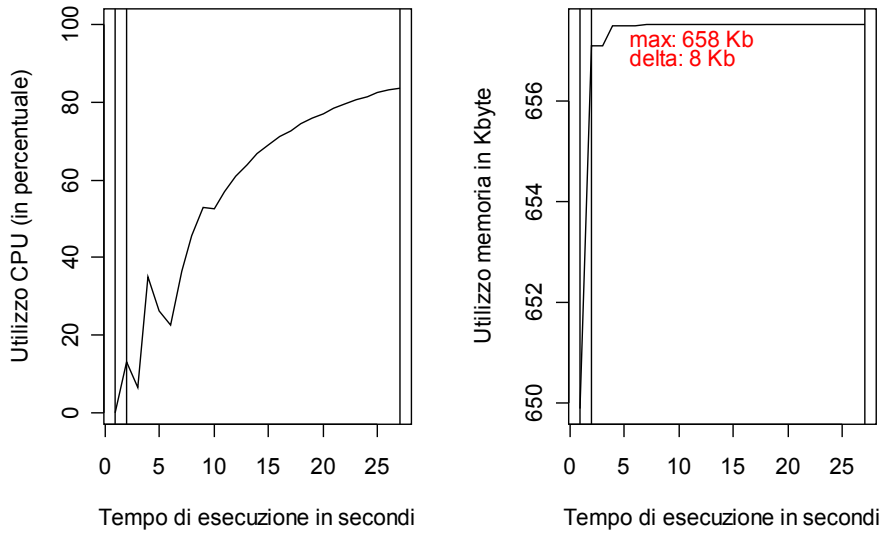

Figura 4.3. Prestazioni computazioni nella creazione del modello lineare con Weka

Si nota anzitutto la maggiore velocità di caricamento dei dati, che è stata eseguita in un solo secondo. Da ulteriori prove effettuate, è emerso che Weka gestisce meglio le variabili discrete con molti fattori rispetto ad R: in dataset di solo attributi numerici, la differenza di prestazioni si annulla.

La fase di costruzione del modello ha impiegato ben 25 secondi. La spiegazione è intuibile in relazione al fatto che la memoria allocata è rimasta pressoché costante: è chiaro che Weka utilizza una procedura ricorsiva sui dati piuttosto di effettuare operazioni su matrici, come avviene per R. Questo approccio è giustificabile secondo una visione conservativa di Weka che, sviluppato appositamente per rispondere a problemi di data mining, preferisce una procedura meno veloce ma più sicura nella stima dei parametri: infatti il modello creato è identico a quello di R in termini di coefficienti ed errori di previsione.

4.2.2. Test con albero decisionale

A causa dell'assenza di possibili alternative di un certo rilievo in Weka, è stata adottata come procedura per la costruzione di un modello ad albero l'algoritmo basato sulla suddivisione degli attributi mediante test sull'informazione. Il modello utilizza l'attributo discreto `class` come variabile risposta e tutti i restanti attributi come variabili esplicative.

Per la costruzione del modello in R è stata scelta la funzione `rpart` presente nell'omonimo pacchetto, configurato esplicitamente ad usare il criterio dell'informazione come test di selezione dell'attributo migliore su cui effettuare la suddivisione. Come nel test precedente, i comandi necessari per svolgere le varie fasi dell'analisi (caricamento dei dati di stima, generazione del modello, caricamento dei dati di verifica e controllo del modello) sono stati inseriti in un apposito file. Quest'ultimo è stato eseguito dal programma R a console, di cui vengono presentati i grafici relativi al costo di elaborazione in termini di tempo di CPU e utilizzo di memoria.

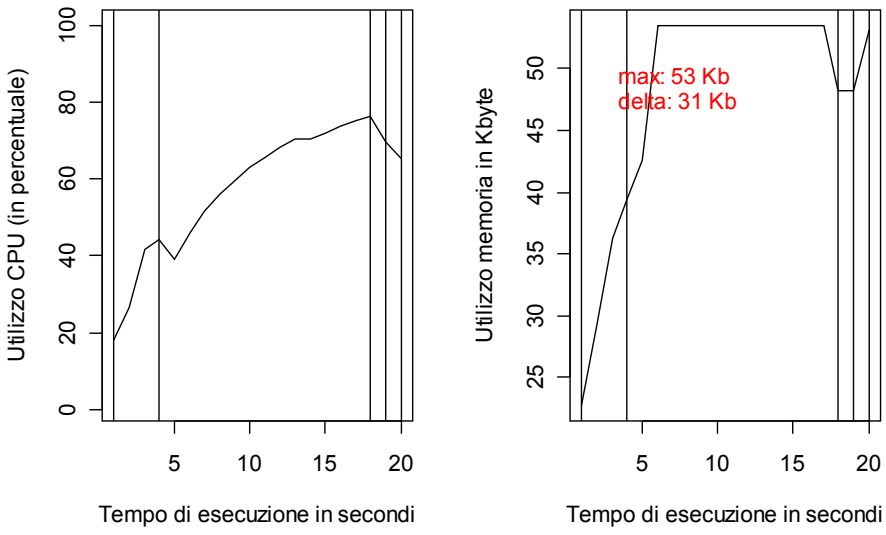

Figura 4.4. Prestazioni durante la creazione dell'albero decisionale con R

Seguono i grafici delle prestazioni misurate durante l'esecuzione dello

stesso codice R dal programma di test in Java utilizzando la versione *embedded* di R .

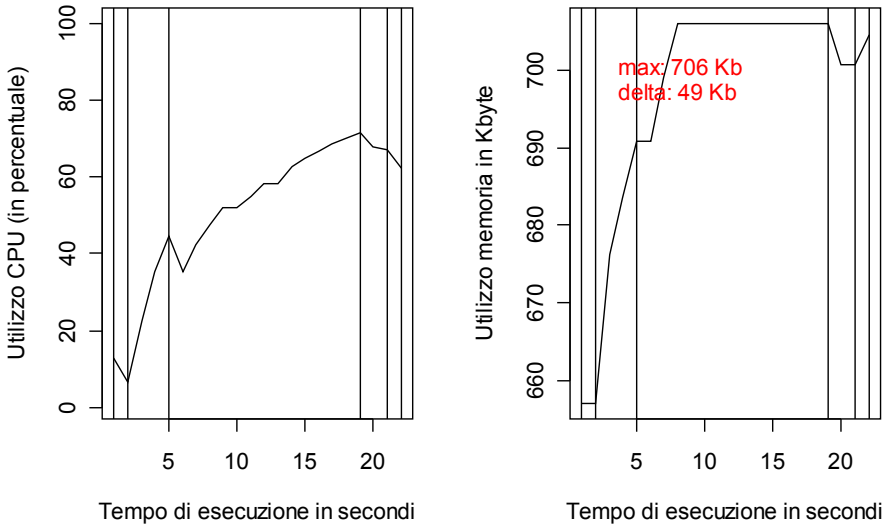

Figura 4.5. Prestazioni durante la creazione dell'albero decisionale con R da piattaforma Java

La fase di caricamento riporta gli stessi valori del test precedente. La creazione del modello ha impiegato tredici secondi con un utilizzo di risorse molto basso: il risultato è un albero di 13 nodi.

Infine la fase di verifica è stata completata in due secondi proporzionalmente utilizzati per la lettura dell'insieme di verifica e la generazione delle previsioni: di seguito viene riportata la tabella di errata classificazione.

Previsione	Risposta effettiva	
	<=50K	>50K
<=50K	11805	1901
>50K	630	1945

Tabella 4.1. Matrice di confusione relativa all'albero costruito con R

La creazione del modello in Weka è stata effettuata mediante la classe `J48` del package `weka.classifiers.trees`: tale classe implementa l'ultima versione dell'algoritmo di Quinlan disponibile

gratuitamente, prima della versione commerciale identificata dalla sigla C5.0. Da notare che non è stata effettuata alcuna modifica alla configurazione proposta come default.

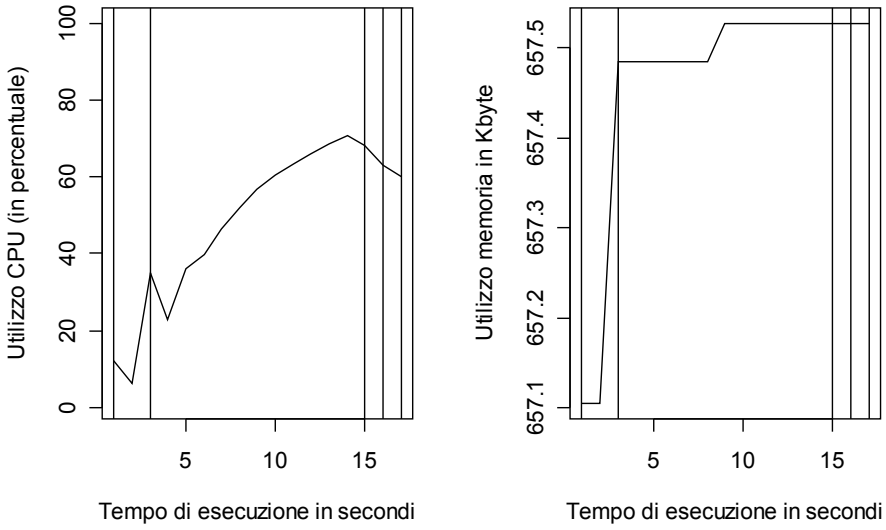

Figura 4.6. Prestazioni durante la creazione dell'albero decisionale con Weka

Ancora una volta, la fase di caricamento riporta gli stessi valori del test precedente. La creazione del modello ha impiegato undici secondi con un utilizzo di risorse molto simile all'esecuzione di R: il risultato tuttavia è un albero di 146 nodi.

Infine la fase di verifica è stata completata in due secondi proporzionalmente utilizzati per la lettura dell'insieme di verifica e il calcolo della previsione: di seguito viene riportata la tabella di errata classificazione.

Previsione	Risposta effettiva	
	<=50K	>50K
<=50K	11556	1425
>50K	879	2421

Tabella 4.2. Matrice di confusione relativa all'albero costruito con R

Un semplice confronto tra i risultati di previsione ottenuti porta a

concludere che il modello costruito con Weka sia il migliore in termini di tempo di elaborazione e attendibilità delle previsioni. Tuttavia se si considera la qualità del modello finale, certamente quello costruito con R risulta molto più semplice, avendo solo 13 rispetto ai 146 nodi proposti da Weka: sebbene il numero di attributi selezionati siano in numero molto inferiore, il tasso di errore relativo ai falsi positivi è pressoché identico (630 per R rispetto ai 879 di Weka).

4.2.3. Test di clustering con l'algoritmo delle K-medie

L'idea di proporre un test sul clustering nasce dalla volontà di valutare la velocità con cui gli algoritmi eseguono il confronto tra le varie osservazioni: la scelta è caduta sull'algoritmo delle K-medie per la semplicità del meccanismo di ricerca che dovrebbe quindi evidenziare meglio le performance di Weka e R. L'insieme di dati utilizzato dall'algoritmo rappresenta una personalizzazione dell'unione tra gli insieme di stima e verifica del dataset *adult*: sono stati infatti eliminati tutti gli attributi discreti in modo da rispettare la formulazione classica dell'algoritmo, ossia la ricerca dei cosiddetti centroidi. Sebbene banale, quest'ultima osservazione serve a sottolineare come Weka, a differenza di R, possiede l'estensione alle variabili discrete nello stesso algoritmo delle K-medie.

Per la costruzione del modello in R, è stata utilizzata la funzione `kmeans` dal pacchetto `stats`: di seguito vengono presentati i grafici ottenuti mediante l'esecuzione dello stesso codice in ambiente R classico e nella versione *embedded* invocata dal programma di test, per la ricerca di due cluster.

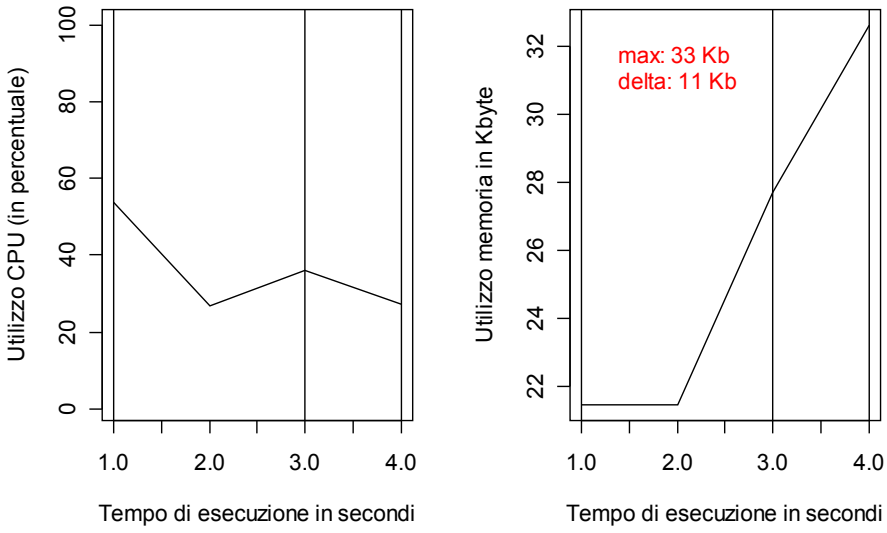

Figura 4.7. Prestazioni dell'algoritmo K-medie in R

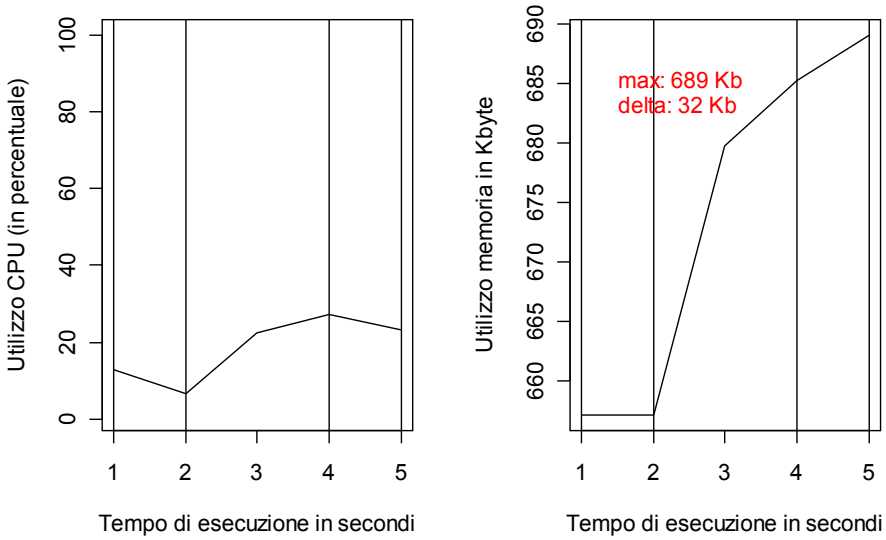

Figura 4.8. Prestazioni dell'algoritmo K-medie in R da piattaforma Java

Il caricamento dell'insieme dei dati (numerosità pari a 48842) costituito da soli attributi numerici è durato due secondi. La fase di stima dei centroidi si è conclusa in un solo secondo, con un utilizzo di CPU e di memoria molto limitato.

La creazione del modello in Weka è stata effettuata mediante la classe `SimpleKMeans` del package `weka.clusterers`, configurato per la ricerca di due cluster.

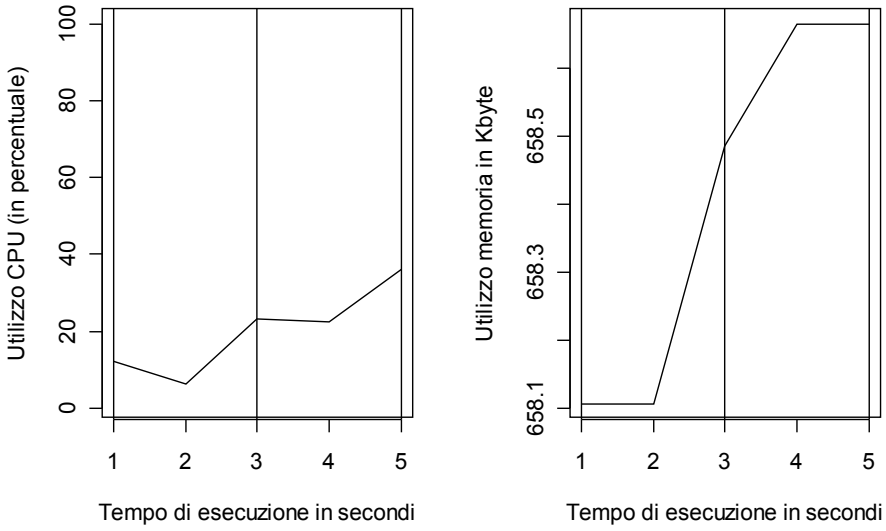

Figura 4.9. Prestazioni dell'algoritmo K-medie in Weka

Il caricamento dell'insieme dei dati è durato due secondi, in analogia al caso di R: questo rappresenta l'ennesima conferma del fatto che R risulta più lento solo nella memorizzazione dei valori relativi ad attributi discreti. La fase di stima dei centroidi è stata completata in due secondi con un leggero aumento del tempo di elaborazione.

Infine, il confronto tra i cluster selezionati da Weka ed R ha dimostrato una sostanziale equivalenza.

4.2.4. Associazione mediante l'algoritmo Apriori

Come nel precedente test, l'insieme di dati utilizzato per il test è stato ottenuto unendo i due insiemi di stima e verifica del dataset *adult*, incrementando così la numerosità campionaria a 48842 osservazioni. Seguendo l'impostazione di Borgelt [4.02], i dati sono stati manipolati in modo da eliminare gli attributi `fnlwgt` (final weigth) e `educational-num`. Successivamente le variabili continue sono state convertite in variabili discrete mediante la creazione di opportune classi di raggruppamento: ad esempio i valori dell'attributo età, originalmente numerici, sono stati sostituiti da un valore discreto appartenente all'insieme {“giovane”, “mezza età”, “anziano”, “vecchio”}.

Per la costruzione del modello in R è stato scelto il pacchetto `arules` contenente l'implementazione originale dell'algoritmo Apriori di Borgelt: la configurazione utilizzata prevede una soglia di supporto e confidenza rispettivamente pari a 10% e al 90%. Di seguito sono presentati i grafici relativi al costo di elaborazione dello stesso file contenente i comandi R necessari all'analisi, eseguito da console e dalla versione *embedded* dal programma di test in Java.

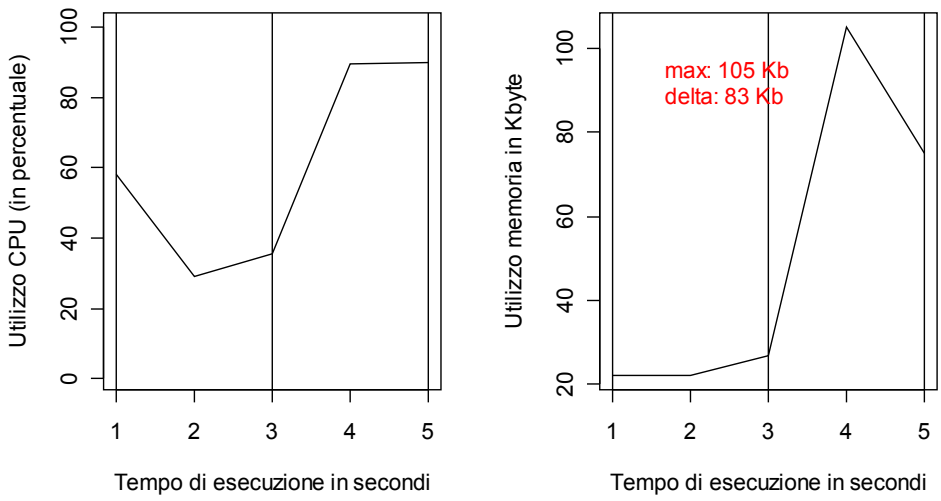

Figura 4.10. Prestazioni nella generazione di regole associative con R

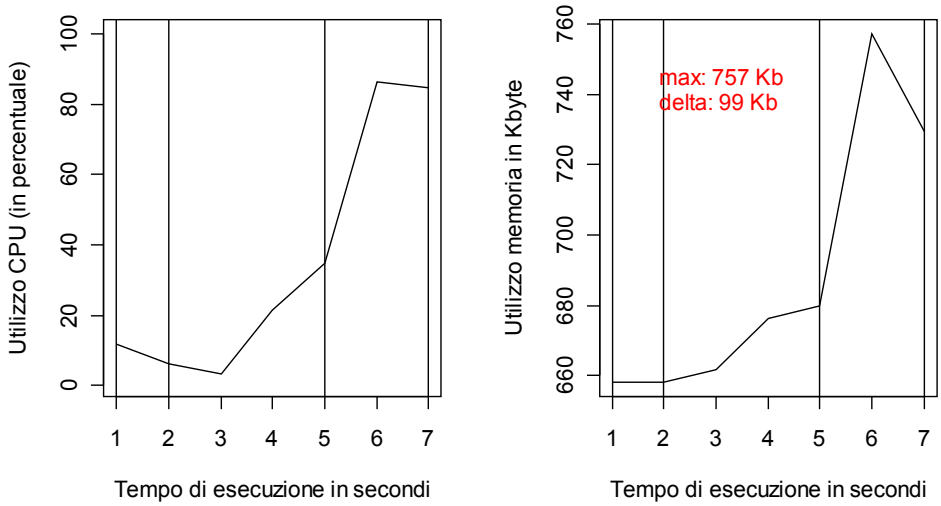

Figura 4.11. Prestazioni nella generazione di regole associative con R da piattaforma Java

La fase di caricamento dei dati è in sintonia con le prestazioni dei test precedenti. La fase di creazione e selezione delle regole di associazione è stata completata in appena due secondi: si sottolinea un certo costo computazionale visto che per metà dell'elaborazione il processore è stato occupato all'89%.

Tuttavia i risultati finali devono tener conto del fatto che il motore R ha impiegato ben 36 secondi per caricare il pacchetto `arule` con le sue dipendenze (`stats4`, `Matrix`, `lattice`), facendo registrare un discreto utilizzo di CPU. Alla luce di queste considerazioni viene proposto un ulteriore grafico, analogo a quello in figura 4.10, che considera anche il costo di caricamento delle estensioni: tale informazione è visualizzata da un'area intermedia tra il caricamento dei dati e la selezione delle regole.

Quest'ultimo grafico proposto (figura 4.12) evidenza come il meccanismo di estensione di R possa talvolta risultare limitante: è infatti possibile spendere più tempo per caricare gli opportuni package di quanto sia necessario all'esecuzione dell'algoritmo stesso. Si precisa però che questo è un costo di inizializzazione, da compiere una volta

sola: l'esecuzione prolungata o ripetitiva dell'algoritmo può quindi ridimensionare pesantemente l'impatto negativo iniziale.

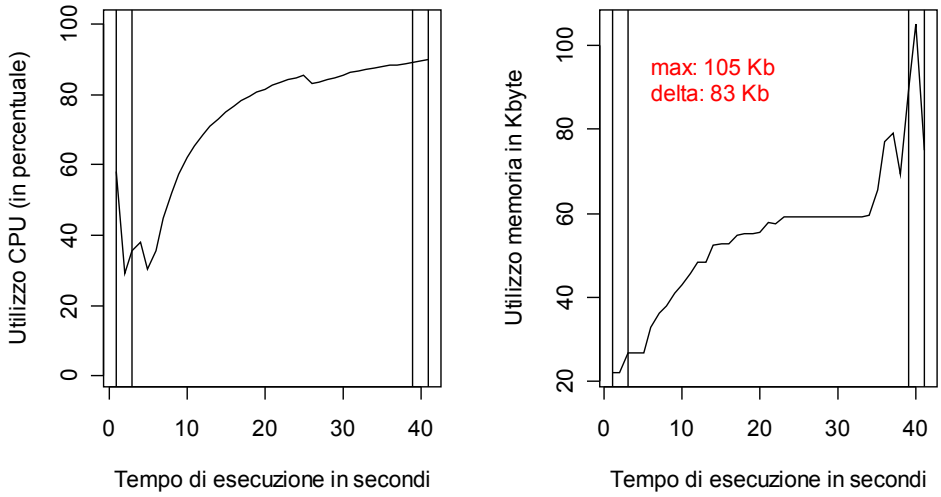

Figura 4.12. Prestazioni nella generazione delle regole associative con R

Per il test su Weka è stato selezionato l'algortimo `Apriori` del package `weka.associations`: sono stati mantenuti tutti i valori di default relativi ai parametri di configurazione, ad eccezione dei livelli di supporto e confidenza impostati come in R.

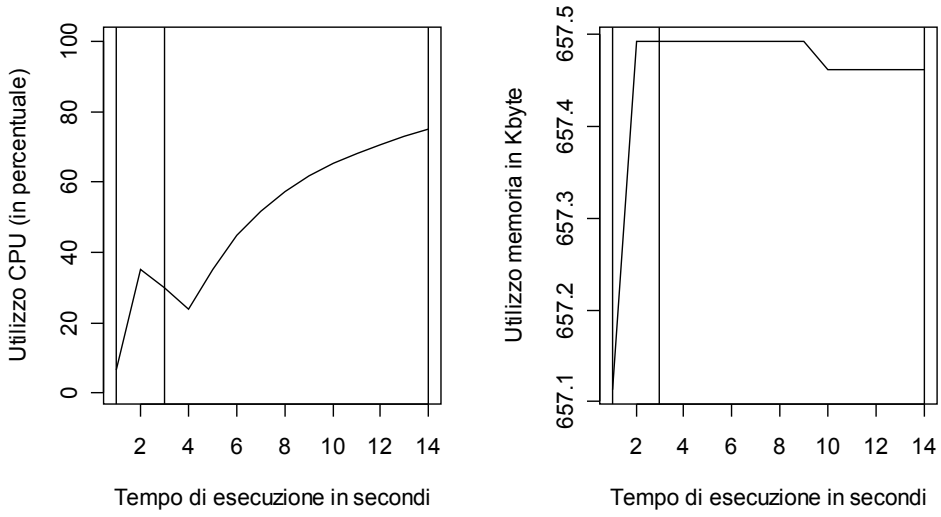

Figura 4.13. Prestazioni nella generazione delle regole associative con Weka

La fase di creazione e selezione delle regole associative ha richiesto ben undici secondi: tuttavia si noti l'inferiore utilizzo di tempo di

processore e la progressiva disallocazione di memoria durante l'elaborazione.

Gli algoritmi utilizzati in R e Weka hanno generato le stesse regole di associazione, con la differenza che in R vengono ignorate quelle aventi confidenza al 100%. Tale comportamento, dovuto alla configurazione di default voluta da Borgelt, è giustificato dal fatto che le regole certe sono generalmente inutili: infatti tra quelle proposte da Weka compaiono

relationship=Husband ⇒ sex=Male
marital.status=Married-civ-spouse, sex=Male ⇒ relationship=Husband

ossia che il ruolo di marito è sempre associato al sesso maschile e che l'uomo civilmente sposato ha sempre il ruolo di marito.

5. Conclusioni

Il data mining rappresenta senza dubbio una nuova frontiera di analisi, destinata a crescere sempre più negli prossimi anni: infatti da una parte continua il frenetico progresso tecnologico, dall'altra hanno sempre maggior diffusione campi di sperimentazione (come la robotica) in cui le tecniche di Machine Learning possono evolversi più rapidamente, grazie soprattutto agli investimenti delle industrie che sperano in un veloce ritorno economico dal settore di business in cui operano.

La scelta dello strumento da utilizzare deve essere necessariamente valutata nel particolare contesto di analisi: infatti le soluzioni offerte da Oracle o IBM sono davvero allettanti in quegli scenari di business in cui il database è una componente fondamentale e insostituibile, tanto da far passare in secondo piano il costo delle relative licenze. Tuttavia è facile prevedere che il fenomeno open source non resterà certamente alla porta: magari con indifferenza e qualche ferita, anche il mondo dell'*Information Tecnology* italiano sta valutando con sempre maggior interesse i prodotti *FOSS*, tanto da contribuire a loro volta con software di qualità come nel caso del progetto *SpagoBI* di Engineering Spa.

A parte proiezioni future sul mondo open source, Weka e R rappresentano già oggi delle soluzioni di alto livello per la maturità del software e per i risultati brillanti che hanno captato l'attenzione di una vasta comunità scientifica. A parte l'aspetto di usabilità in cui Weka si propone in modo più accattivante grazie alle funzionalità a portata di mouse, certamente R si è dimostrato migliore in termini di flessibilità: per quanto ostico possa sembrare, un linguaggio attraverso cui interagire con il motore computazionale e grafico non ha eguali per quanto riguarda comodità e potenzialità. Siccome i test eseguiti non hanno evidenziato grosse differenze se non in termini di occupazione

di memoria e un leggero ritardo nella stima dei modelli per Weka (dovuti essenzialmente alla peculiarità della Java Virtual Machine), non è possibile definire una casistica sulla base della quale effettuare la scelta di Weka piuttosto che R: ancora una volta, sarà il contesto di analisi ad offrire i parametri su cui basare la scelta dello strumento migliore. Tuttavia è indubbio che R possieda attualmente il maggior numero di estensioni rispetto a Weka: anzi, il meccanismo di importazione di nuovi pacchetti garantisce ad R una maggiore facilità e rapidità nell'integrare nuovi algoritmi o librerie precompilate fornite da terze parti.

A. Appendice

In questa sezione vengono presentati con maggior dettaglio i passaggi tecnici ritenuti interessanti per il lettore, relativamente al programma di test scritto in Java con cui sono state effettuate le prove su Weka e R in versione *embedded*. Si precisa che tale programma è stato sviluppato ed eseguito in ambiente Linux: per un porting su Microsoft Windows si rimanda alla documentazione ufficiale presente nella rete CRAN.

A.1. Creazione del motore R in versione embedded

Generalmente l'utilizzo di R avviene tramite linea di comando o interfaccia grafica: per questo motivo i file binari di R vengono distribuiti come programmi statici, inglobando il motore computazionale e grafico nello stesso file eseguibile. Tuttavia in fase di compilazione dei sorgenti è possibile scegliere l'opzione con cui isolare il motore R in una particolare libreria dinamica, detta `libR.a` o `R.dll` a seconda che il sistema operativo sia UNIX o Microsoft Windows: in tale scenario i suddetti file eseguibili sono automaticamente configurati in modo da caricare tale libreria in fase di inizializzazione del processo stesso.

Il vantaggio di questo approccio consiste nella possibilità di accedere direttamente alle potenzialità di R senza dover per forza dipendere dagli eseguibili ufficiali mantenuti dagli sviluppatori di R. A tal proposito nella rete CRAN sono presenti documenti molto dettagliati sul funzionamento e sulle modalità d'uso delle API messe a disposizione dalla stessa libreria di dinamica.

In ambiente UNIX la creazione del motore R in versione *embedded* viene effettuata molto semplicemente mediante l'esecuzione dei seguenti comandi dalla cartella dei sorgenti:

```
cd R-2.3.1
./configure --enable-R-shlib
make
make install
```

A.2. Integrazione di R in ambiente Java

L'integrazione di R in ambiente Java è possibile in due modi diversi: lanciando l'eseguibile R da un programma Java oppure interagendo direttamente con il motore di R mediante l'invocazione delle opportune API messe a disposizione dalla versione *embedded* di R. Ai fini della relazione è stata affrontata la seconda strada che, sebbene più complessa, permette una maggiore flessibilità e cooperazione con altri software. Nel proseguo del paragrafo viene quindi affrontato il modo in cui far interagire un programma Java con la libreria dinamica contenente il motore R.

La Java Virtual Machine possiede un particolare componente, detto JNI (*Java Native Interface*), attraverso cui eseguire codice compilato in linguaggio macchina e fisicamente contenuto in una libreria dinamica presente nel sistema. Da notare che tale soluzione rappresenta un ostacolo alla portabilità, dal momento che è necessario distribuire assieme al programma Java anche la libreria dinamica compilata e (soprattutto) compatibile con il particolare sistema operativo ospitante.

Tecnicamente è necessario scrivere una classe con almeno un metodo dichiarato come nativo (`native`): il compilatore `javac` associa automaticamente ad un metodo siffatto una particolare funzione che si presuppone sia presente in una libreria dinamica. Tale libreria dovrà essere caricata dallo stesso programma Java prima di invocare il metodo nativo, pena l'impossibilità di eseguire l'operazione richiesta. Una volta compilata la classe, deve essere eseguito il comando `javah` (presente nella *Java Development Kit*): esso crea un particolare file in linguaggio C/C++ (cosiddetto *header*) contenente i riferimenti al

metodo nativo della classe stessa. Il passo conclusivo consiste nell'implementare tale funzione in linguaggio C/C++ e compilare il codice sorgente come libreria dinamica.

Segue ora una presentazione pratica dei passaggi salienti: verranno riportate porzioni di codice e opportuni comandi UNIX utilizzati per la realizzazione del programma Java con cui sono state eseguite le prove pratiche del capitolo 4.

Innanzitutto viene riportata una porzione di codice relativa alla classe Java che costituisce il concetto di plugin con R: tecnicamente è modellata sul pattern *singleton* per garantire un controllo centralizzato del caricamento della libreria e dell'inizializzazione del motore R.

```
package dade.util;
public class RPlugin
{
 final static public String libraryName = "Rplugin";
 private static RPlugin instance = null;
 //----------------------------------------
 public static RPlugin getInstance()
 throws Exception
 {
  if( instance==null )
  try {
   // apriamo la libreria dinamica:
   // in Windows ci si aspetta la libreria Rplugin.dll,
   // in UNIX la libreria libRPlugin.so
   System.loadLibrary(libraryName);
   instance = new RPlugin();
  }
  catch(Exception ex){
   System.err.println("Errore libreria dinamica");
   ex.printStackTrace();
  }
  return instance;
 }
 //-----------------------------------------------
 /** Esegue un file con i comandi R
  *  <p>
  *  @param source nome del file con il codice R
```

```
  *  @return true se l'esecuzione di R ha avuto
  *                successo, altrimenti false
  */
 public native boolean execute(String source)
 throws Exception;
}
```

I comandi per la compilazione della classe e la creazione del file *header* sono:

```
javac -classpath $MYCLASSPATH dade/RPlugin.java
javah -jni -classpath $MYCLASSPATH dade.RPlugin
# viene generato il file dade_RPlugin.h
```

Segue un pezzo di codice sorgente in C del file `Rplugin.c`, contenente l'implementazione della funzione dichiarata nel file `dade_RPlugin.h`, a sua volta generato dal comando `javah`: tale funzione verrà eseguita dalla Java Virtual Machine in corrispondenza dell'invocazione del metodo nativo `execute`.

```
#include "dade_RPlugin.h"
/*-------------------------------------------------
 */
JNIEXPORT jboolean JNICALL Java_dade_RPlugin_execute(
   JNIEnv *env, jobject obj, jstring hab)
{
 /* invoco le API R per l'esecuzione del codice R
  * presente nel file contenuto nella variabile "hab",
  * proveniente dall'ambiente Java
  */
}
```

Tale sorgente rappresenta il punto d'intersezione tra la parte Java e R, trasformando le operazioni richieste dai metodi nativi della classe Java in una sequenza di chiamate alle funzioni API presenti nella libreria dinamica con il motore R. Concettualmente la Java Virtual Machine non invoca direttamente R ma una libreria creata appositamente per rispondere al meccanismo di comunicazione imposto dalla componente JNI.

Nel nostro contesto, tale libreria è stata chiamata `libRPlugin.so`: i comandi per la sua compilazione sono

```
gcc -c -I$JAVA_HOME/include -I$JAVA_HOME/include/linux \
       -I$R_HOME/include RPlugin.c
gcc -shared -o libRPlugin.so RPlugin.o -L$R_HOME/lib -lR
```

dove R_HOME e JAVA_HOME sono le variabili contenenti le directory di installazione rispettivamente di R e della JDK (*Java Development Kit*).

Bibliografia

[1.01] Usama Fayyad, Gregory Piatetsky-Shapiro & Padhraic Smyth (1994) "*From Data Mining to Knowledge Discovery in Databases"*. American Association for Artificial Intelligence, Magazine

[1.02] Holsheimer, M. & Siebes (1994) *"Data Mining: The Search for Knowledge in Databases"*. Report CS-R9406. CWI. Amsterdam, The Netherlands

[1.03] Chapman P., Clinton J, Kerber R., Khabaza T., Reinartz T. (august 2000) *"CRISP-DM 1.0"*, The CRISP-DM Consortium

[1.04] I.H. Witten & E. Frank (2005) *"Data Mining: Practical Machine Learning Tools and Techniques"*. Elsevier Inc., San Francisco, second edition

[1.05] Quinlan R. (1993) "*C4.5: Programs for Machine Learning"*. Morgan Kaufmann Publishers, San Mateo, California

[1.06] Breiman L., H. Friedman, J. A. Olshen & C. J. Stone (1984) *"Classification and Regression Trees"*. Wadsworth International Group

[1.07] A. Azzalini & B. Scarpa (2004) *"Analisi dei dati e data mining"*, Springer. Milano

[1.08] Chan, K. Y. & W. Y. Loh (2004) *"LOTUS: An Algorithm for Building Accurate and Comprehensible Logistic Regression Trees"*. Journal of Computational and Graphical Statistics

[1.09] Agrawal R. & Imielienski T. & Swami A. (1993). *"Mining Association Rules between Sets of Items in Large Databases"*. In: Proc. Conf. on Management of Data, 207–216. New York: ACM Press

[1.10] R. Agrawal & R. Srikant (1994) *"Fast Algoritms for Mining Association Rules"*. IBM Almaden Research Center

[1.11] J. Magidson. (1994) *"The Chaid approach to segmentation modeling: Chi- squared automatic interaction detection"*. In R. P. Bagozzi, editor, Advanced Methods of Marketing Research, pagine 118-159. Blackwell Business, Cambridge Massechusetts

[2.01] Mark Hornick & JSR-73 Expert Group (29 luglio 2004) *"JavaTM Specification Request 73: JavaTM Data Mining (JDM)"* [http://jcp.org/aboutJava/communityprocess/mrel/jsr073]

[2.02] Witten, I. H. & E. Frank (2000) "*Data Mining: Practical Machine Learning Tools and Techniques with Java Implemenations"*. San Francisco: Morgan Kaufmann

[3.01] Ihaka, R. & R. Gentleman (1996) *"R: A Language for Data Analysis and Graphics"*. Journal of Computational and Graphical Statistics

[3.02] J.H.Maindonald (2004) "*Using R for Data Analysis and Graphics*". Australian National University

[4.01] Newman, D.J. & Hettich, S. & 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

[4.02] Christian Borgelt & Rudolf Kruse (2002) *"15th Conference on Computational Statistics"* (Compstat 2002, Berlin, Germany) Physica Verlag, Heidelberg, Germany