Previsione di Sintomatologie Post-Dialisi

Studio e realizzazione di un sistema supervisionato di estrazione regole

Francesco Pontillo (mat. 600119)
Universitá degli Studi di Bari
Dipartimento di Informatica
Via E. Orabona, 4 - 70125 Bari, Italy
francescopontillo@gmail.com

ABSTRACT

Implementazione Prolog dell'algoritmo di Intelligenza Artificiale C4.5, con k-fold test e confronto con altri sistemi pre-esistenti di classificazione.

1. OBIETTIVO

Obiettivo del processo di Data Mining del sistema da sviluppare é di prevedere possibili sintomatologie successive ad una seduta di emodialisi. A partire da specifici dati registrati durante una dialisi, si vuole prevedere quali classi di sintomatologie il paziente potrá riscontrare dal momento in cui la dialisi termina al momento in cui esegue la seduta di dialisi successiva.

In questo modo, il medico puó confermare la possibilitá di occorrenza di una o piú problematiche suggerite, ed eventualmente prescrivere una opportuna terapia per contrastare la sua insorgenza.

2. SELEZIONE DEGLI ATTRIBUTI

I dati a disposizione nella base di dati da analizzare sono numerosi, e devono essere selezionati appropriatamente per evitare l'introduzione di attributi poco rilevanti con lo scopo del sistema.

Ogni seduta di dialisi memorizza (1) una **data** di svolgimento, (2) la **durata** della seduta stessa, (3) un identificativo del **paziente**, (4) altri **parametri** registrati durante la sessione e (5) eventuali **sintomatologie** riscontrate.

2.1 Dati del paziente

Le informazioni relative ai pazienti sono ricavate, anonimizzandole, dalla base di dati originale. Ai fini del processo di estrazione delle regole, é opportuno considerare il **sesso** del paziente e la sua **etá** al momento della seduta di dialisi in analisi[1].

[1]Ció non esclude la possibilitá di considerare altri dati relativi al paziente; l'algoritmo da realizzare potrebbe essere este-

2.2 Parametri della seduta di dialisi

I parametri piú rilevanti di una seduta di dialisi, al fine di prevedere eventuali sintomatologie successive, sono divisi in piú categorie [2] [3].

A. L'**efficienza della rimozione dei prodotti di scarto** é indotta dai valori dei parametri riportati in Tabella 1.

KT/V	indice di efficienza dialitica
QB	flusso di sangue trattato
WS	peso iniziale
WE	peso finale
PWE	peso finale ottimale
PT	durata ottimale
T	durata reale

Table 1: Parametri di efficienza eliminazione scarti

B. L'efficienza dell'eliminazione dell'acqua all'interno del corpo del paziente é indotta dai parametri in Tabella 2.

SPS	pressione sistolica iniziale
SPE	pressione sistolica finale
DPS	pressione diastolica iniziale
DPE	pressione diastolica finale
BV	volume ematico finale

Table 2: Parametri di efficienza eliminazione acqua

C. Altre tipologie di dati che potrebbero risultare utili a fornire previsioni significative sono riportati in Tabella 3.

PBF	flusso sangue teorico
BF	flusso sangue reale
PUF	ultrafiltrazione media teorica
UF	ultrafiltrazione media reale

Table 3: Altri parametri di efficienza dialitica

2.3 Attributi derivati

A partire dalle informazioni disponibili nella base di dati, risulta evidente la presenza di alcuni attributi "nascosti" che possono essere più utili ai fini dell'apprendimento.

In tabella 4 sono elencati gli attributi derivati dalle precedenti tabelle; ad esempio, ΔWL rappresenta la differenza

fra la perdita di peso programmata e quella effettiva, a sua volta calcolata come differenza fra peso iniziale e peso finale.

PWL	perdita peso programmata
RWL	perdita peso reale
ΔWL	differenza perdita peso
ΔT	differenza durata trattamento
SPA	pressione sistolica media
DPA	pressione diastolica media
ΔBF	differenza flusso sangue
ΔUF	differenza UF medio

Table 4: Parametri derivati

2.4 Sintomatologie

Il sistema verrá addestrato con istanze di esempio pre-classificate. La classificazione consiste nell'assegnazione, ad ogni esempio, di una o piú categorie di sintomi, ad esempio: aritmia sintomatica, aritmia asintomatica, astenia, brividi, brividi e dispnea, cefalea, collasso (PA $< 30\%$ inizio), conati di vomito, crampi, depressione, ansia, diarrea, dispnea e molti altri.

Inoltre, é prevista la classe 'asintomatico', che definisce una sintomatologia assente corrispondente ad un esempio negativo dal punto di vista della classificazione.

3. SELEZIONE DEI DATI

Le informazioni sottoposte all'algoritmo di apprendimento sono state selezionate a partire da una base dati molto ricca[2] e sono stati sottoposti ad una serie di passaggi[3].

3.1 Creazione dei valori derivati

Per poter istanziare i valori degli attributi definiti in 2.3, é stata eseguita una query di tipo `SELECT` che preleva informazioni dalla tabella di origine ed effettua semplici calcoli di trasformazione.

In questo modo, alla fine del processo di trasformazione, gli attributi per ogni seduta di dialisi sono:

- `SESSION_ID`, l'ID della seduta di dialisi, utile per identificare la seduta in ogni momento
- `SESSION_DATE`, la data di esecuzione
- `KTV`, il valore di KT/V
- `QB`, il valore di QB
- `PROG_WEIGHT_LOSS`, la perdita peso programmata
- `REAL_WEIGHT_LOSS`, la perdita peso reale
- `DELTA_WEIGHT`, la differenza fra la perdita di peso reale e quella programmata
- `PROG_DURATION`, la durata programmata della dialisi
- `REAL_DURATION`, la durata effettiva della dialisi
- `DELTA_DURATION`, la differenza fra la durata reale e quella programmata
- `SAP_START`, la pressione sistolica arteriosa prima della seduta
- `SAP_END`, la pressione sistolica arteriosa dopo la seduta
- `AVG_SAP`, la pressione sistolica arteriosa media
- `DAP_START`, la pressione diastolica arteriosa prima della seduta
- `DAP_END`, la pressione diastolica arteriosa dopo la seduta
- `AVG_DAP`, la pressione diastolica arteriosa media
- `BLOOD_VOLUME`, il volume di sangue trattato
- `DELTA_BLOOD_FLOW`, la differenza fra flusso di sangue teorico ed effettivo
- `DELTA_UF`, la differenza dell'ultrafiltrazione media reale e teorica

Come si nota, sono stati eliminati alcuni attributi originali: il flusso di sangue teorico e reale e l'ultrafiltrazione media teorica e reale.

3.2 Associazione con sintomatologie

Nel programma che genera i dati, le sintomatologie vengono comunicate e quindi inserite, dal medico o dall'infermiere, qualche momento prima della dialisi successiva del paziente. Per poter mettere a confronto i dati della seduta di dialisi, di cui sopra, con i dati della sintomatologia rilevata, é stato necessario eseguire una query molto complessa per mettere in correlazione:

- il paziente
- la data di dialisi
- la data di dialisi minore fra quelle successive alla data di riferimento della seduta originaria

3.3 Associazione con dati del paziente

Infine, il dato della sintomatologia singola è stato associato univocamente con il paziente di riferimento, tramite l'apposito identificativo.

Tutte queste operazioni sono state eseguite staticamente, ovvero andando a creare una copia dei record in altre tabelle; ció si é reso necessario in quanto, anche utilizzando macchine potenti, la selezione completa dei record impiegava interi minuti per completare, soprattutto a causa dell'associazione poco ottimizzata con le date (cfr. 3.2).

so andando a considerare anche i dati relativi alle malattie pregresse del paziente ed eventuali comorbiditá registrate.

[2]Circa dal 1999 ai primi mesi del 2014.

[3]Tutte le trasformazioni e selezioni di dati descritte in questa sezione sono codificate nel `scripts/01-sql-server-tables.sql`.

3.4 Migrazione dei dati

Per una gestione piú libera dei dati, si é scelto di migrare le tabelle create da Microsoft SQL Server a MySQL[4], anche in ottica futura (cfr. 9).

4. PULIZIA DEI DATI

Una volta spostati i dati su un database MySQL, si é scelto di eliminare alcuni record e mantenerne altri piú rilevanti[5]. La base dati originaria, infatti, contiene 185476 record.

4.1 Calcolo dello score

Ad ogni riga di rilevazione sintomo é stato associato un punteggio, o *score*, che permetta di capire quanto quella riga é completa (e quindi piú o meno rilevante rispetto alle altre).

Fissato il numero degli attributi (di dialisi) a 15, un record con *score* piú elevato sará selezionato con piú probabilitá per avviare il processo di apprendimento.

4.2 Pazienti rilevanti

Inoltre, lo *score* é stato utilizzato anche per poter eliminare, dai record giá selezionati, tutti quelli che appartengono a pazienti che hanno meno di 5 rilevazioni di sintomi con uno *score* percentuale piú basso dell'80%.

Tutti i dati selezionati fino a questo punto, quindi, appartengono a pazienti che hanno almeno 5 rilevazioni di sintomi ottimali.

4.3 Gestione dei valori nulli

I valori nulli sono stati gestiti "staticamente", ovvero per ogni paziente sono state calcolate le medie dei valori di ogni attributo (ignorando quindi i valori nulli); in un passo successivo, sono stati scansionati tutti i record e, qualora fosse rilevato un valore nullo, é stato inserito il valore medio relativo al paziente associato.

Tutto ció, tuttavia, ha portato comunque a mantenere alcuni valori nulli all'interno della base dati. Ad esempio, poche rilevazioni di sintomatologia contengono valori effettivi di `KTV`, probabilmente perché si tratta di una misura di difficile calcolo da parte dei medici. Tali valori sono stati gestiti diversamente (cfr. 7.5.2).

Alcuni record, inoltre, non contenevano l'ID del sintomo target rilevato, e si é pertanto assunto che l'utente avesse erroneamente cancellato (dall'interfaccia del sistema) la dicitura "asintomatico", aggiungendo comunque una sintomatologia nulla (con ID uguale a 1).

5. APPRENDIMENTO DI REGOLE

L'obiettivo del sistema é l'apprendimento di regole utilizzabili per fare previsioni significative. Avendo a disposizione una moltitudine di dati pre-classificati, risulta immediato pensare ad un approccio guidato che generi un albero di decisione.

[4]Lo script di migrazione é presente in `scripts/02-mysql-migration-script.sql` e viene richiamato in automatico, tramite appositi parametri di connessione, dal file batch `03-mysql-copy-migrated-tables.cmd`.

[5]Gli script rilevanti sono contenuti nel file `scripts/04-mysql-scores.sql`.

5.1 Alberi di decisione

Nei sistemi TDIDT, il concetto é rappresentato in termini di un albero di decisione costruito in modalitá top-down con una tecnica model driven: il sistema seleziona un attributo che caratterizza correttamente tutti gli esempi, quindi procede ricorsivamente fino a raggiungere la copertura totale.

Questi algoritmi distinguono tra uno spazio degli esempi e uno spazio delle ipotesi, riuscendo a ridurre di molto la complessità di ricerca nello spazio delle ipotesi.

Siano dati:

- S_0 insieme di oggetti
- C insieme di classi
- $\mathbb{A}$ insieme di attributi
- $\Lambda_A = \{a_1, \cdots, a_r\}$ valori discreti che un attributo $A \in \mathbb{A}$ può assumere

La costruzione dell'albero di decisione equivale a trovare un albero T che classifichi correttamente tutti gli oggetti in S_0.

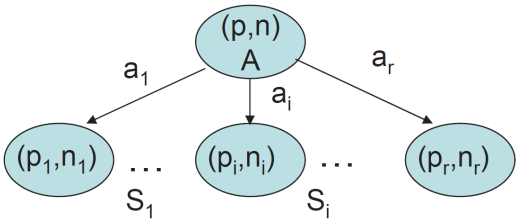

Figure 1: Esemplificazione di un albero di decisione

Con riferimento alla figura 1, si nota che:

- $S_i \cap S_j = \emptyset, \forall i \neq j$, cioè ogni sotto-albero S_i è disgiunto dagli altri S_j adiacenti (nello stesso livello)
- $S_i = \{e \in S | A(e) = a_i\}$, per ogni esempio e appartenente ad un sotto-albero S_i, l'attributo $A(e)$ assume sempre lo stesso valore a_i (l'attributo selezionato per caratterizzare il sotto-albero S_i ha valore costante in tutti i sotto-alberi figli)

5.2 Costruzione dell'albero di decisione

A partire da esempi pre-classificati in fase di training, il sistema genera un albero di decisione in cui S_0 è la radice. Ad ogni nodo σ viene selezionato il miglior attributo A^*, in accordo a qualche criterio, per effettuare il test a quel nodo. Infine, si assegna il nome di una classe ad ogni nodo foglia.

L'albero può anche essere validato su un insieme di testing: si segue il cammino dalla radice ad una foglia testando ad ogni nodo il valore dell'attributo selezionato, e procedendo il cammino sul valore (o range di valori) dell'osservazione da classificare.

5.3 Scelta dell'attributo più discriminante

Esistono diversi criteri euristici per scegliere l'attributo più discriminante ad ogni livello, ognuno dei quali utilizza una misura specifica per massimizzare l'effetto discriminante:

- Massimizzazione dell'**informazione**
  - entropia minima
  - rapporto di guadagno
  - guadagno di informazione normalizzato
  - riduzione della lunghezza della descrizione
- Minimizzazione dell'**errore**
  - riduzione dell'errore nel training set
  - dissimilarità
  - indice di diversità di Gini
- Massimizzazione della **significatività**, basato su statistiche varie (χ^2, G, $\cdots$)

5.3.1 *Entropia*

L'entropia definisce la *impurity* di un insieme arbitrario di dati S, contenente esempi positivi (in proporzione p_+) e negativi (in proporzione p_-):

$$entropia(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

In questo modo il valore dell'entropia agli estremi è:

- 0 se c'è minima entropia, ovvero se tutti gli esempi sono positivi
- 1 se c'è massima entropia, ovvero se gli esempi sono positivi e negativi in ugual numero

La funzione entropia relativa ad una classificazione booleana varia coma la proporzione p_+, cioè fra 0 e 1.

Più in generale, se il concetto target può assumere c valori anziché 2, la funzione entropia diventa:

$$entropia(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

dove p_i è la proporzione di esempi in S appartenenti alla classe i; In questo senso, la categorizzazione booleana è una sua specializzazione a due classi.

5.3.2 *Information Gain*

L'information gain di un attributo A relativo ad un insieme di esempi S è così calcolato:

$$Gain(S, A) = entropia(S) - \left[\sum_{a \in \Lambda_A} \left(\frac{|S_a|}{|S|} \times entropia(S_a) \right) \right]$$

dove S_a è il sottoinsieme di S per il quale l'attributo A ha valore a.

$Gain(S, A)$ rappresenta la **riduzione attesa in entropia** (disordine), causata dal conoscere il valore dell'attributo A. Maggiore è la riduzione dell'entropia, maggiore è il guadagno che si ottiene selezionando A come attributo discriminante.

5.4 Considerazioni sui sistemi TDIDT

É possibile passare da alberi di decisione a regole di decisione, semplicemente analizzando ogni cammino dai nodi foglia al nodo radice e generando, per ognuno, una regola in cui la scelta di uno specifico arco introduce un nuovo congiunto.

I sistemi TDIDT sono molto efficienti, e gli alberi di decisione possono trattare sia attributi a valori categorici, gestendo più archi uscenti da ogni nodo, che continui, tramite definizione di intervalli che vanno a configurarsi come categorie.

Essi sono inoltre non parametrici e non incrementali, in quanto l'introduzione di un nuovo esempio potrebbe rovinare la rigida struttura ad albero, e quindi il suo potere inferenziale.
Tuttavia, nel caso di dati rumorosi, potrebbero essere generati alberi di dimensioni enormi, che andrebbero quindi gestiti con opportuni algoritmi di pruning.

Altri metodi, infine, rimpiazzano i nodi foglia con distribuzioni di probabilità della classe inferita, soprattutto con numerosi esempi di apprendimento.

5.5 C4.5

Per apprendere regole utili a classificare appositamente una seduta di dialisi si é scelto di implementare il funzionamento base dell'algoritmo C4.5 di Ross Quinlan [5] [6], estensione del precedente ID3[4]. In questo modo risulta possibile:

- utilizzare la sintomatologia come attributo target (da classificare)
- poter sfruttare i numerosi dati disponibili
- generare un albero di decisione
- convertire l'albero in un insieme di regole Prolog

L'implementazione del C4.5 é stata descritta, concettualmente e nella sua implementazione, in 7.5.2.

6. TEST DELL'ALGORITMO

Gli esempi di rilevazioni di sintomatologie a disposizione sono, purtroppo, mal distribuiti: alcune sintomatologie sono molto presenti, mentre altre sono molto rare. Per questo motivo, e per valutare la stabilitá del sistema, si é ritenuto opportuno validare le regole generate.

6.1 Test su algoritmi di IA

Il processo di apprendimento per gli schemi più comuni opera in più fasi:

1. Costruzione della struttura base a partire dai **training data**
2. Ottimizzazione delle impostazioni sui parametri, eseguita con i **validation data**
3. Test del sistema ottenuto, tramite i **test data**

Ovviamente training, validation e test data sono insiemi di dati completamente indipendenti.

A valutazione completa, infine, tutti i dati rilevati devono essere utilizzati per ricostruire un nuovo classificatore per l'utilizzo effettivo.

Non potendo disporre di un dataset molto stabile[6], tuttavia, si é dovuto ricorrere ad altre tecniche per realizzare un classificatore che possa essere ritenuto valido.

6.2 K-Fold Cross-Validation

La **cross-validation** é una metodologia di test che divide l'insieme dei dati in:

- **training set**, di solito $\frac{9}{10}$ dei dati originali
- **testing set**, di solito $\frac{1}{10}$ dei dati originali

Operando in questa maniera, l'insieme dei dati di test é sempre differente dall'insieme dei dati di *training*. In generale, il processo:

1. Divide i dati in k sotto-insiemi di dimensioni uguali.
2. Utilizza, a turno, ogni sotto-insieme per il testing, ed i rimanenti per il training.
3. Esegue, alla fine dei k passi, una media delle stime di errore per ottenere una **stima di errore finale**.

Poiché la cross-validation partiziona l'insieme degli esempi in k sotto-insiemi, è chiamata anche **k-fold cross-validation**.

Il metodo standard per la valutazione è il **10-fold cross-validation**, che divide lo spazio degli esempi in 10 sotto-insiemi, in quanto sperimentazioni estensive (ma anche dimostrazioni teoriche) hanno rilevato che 10 è la scelta migliore di partizionamento per poter fornire una stima accurata.

7. IMPLEMENTAZIONE IN PROLOG

Il programma Prolog é diviso in 6 moduli, ognuno dei quali si occupa di parti differenti del programma[7].

Si é scelto di utilizzare SWI-Prolog[1] come ambiente Prolog.

7.1 Utility

Sono stati realizzati 2 moduli che realizzano funzioni di utilitá.

7.1.1 `util.pl`

`util.pl` contiene brevi regole che implementano:

- timer, con avvio, lettura e stop (`timer_start`, `timer_get`, `timer_stop`, tra gli altri)
- formattazione di millisecondi (`format_ms`) e secondi (`format_s`) nel formato intellegibile `{M}m {S}s {MS}ms`
- stampa a video di generiche liste di elementi o di un elemento singolo, con ritorno a capo (`println`)
- generici *helper* per liste, che realizzano funzioni di minimo e massimo (`list_min`, `list_max`), ricerca dell'elemento piú comune (`list_most_common`) e dell'indice di un elemento specifico (`index_of`), oltre che funzionalitá di aggiunta e rimozione di elementi
- concatenazione di elementi di una lista in una stringa
- realizzazione del logaritmo in base 2 (`log2`), utilizzato successivamente (cfr. 7.5).

7.2 Avvio del programma

Il programma principale é definito nel modulo `main.pl`, che si occupa del caricamento in memoria di tutti i file Prolog necessari e di definire il metodo `main(Config, Symptom)`:

- `Config` dichiara al programma qual é il file di configurazione con il quale si vuole accedere al database.[8]. Vedi 7.3 per il l'accesso al database. Si é optato per un file di configurazione che contenesse tutti i parametri di connessione poiché la scrittura degli stessi ad ogni avvio del programma sarebbe risultata troppo verbosa.
- `Symptom` definisce l'ID del sintomo che si vuole utilizzare come esempio positivo per l'attributo target.

Entrambi i parametri supportano i meta-valori `default` e `ask`: `default` esegue un *fallback* dei parametri sul file di configurazione `prolog/config/database.properties` e sul sintomo con ID 2, mentre `ask` imposta il programma in modo da chiedere all'utente, in maniera interattiva e quando necessario, gli stessi parametri. La figura 2 mostra l'avvio dell'algoritmo con il file di configurazione di default e per il sintomo 8.

Sono anche presenti funzioni di avvio rapido: `main_def/0` (che avvia il programma con parametri di default), `main/0` (che avvia in modalitá `ask`) e `make_doc/0` (che genera la documentazione HTML).

Il sistema di logging implementato é, di default, molto verboso. Se non si vuole avere in output il dettaglio completo di ció che sta avvenendo nel programma, é possibile utilizzare le regole messe a disposizione dal modulo di logging (vedi 7.1) per settare un livello di logging meno verboso:

```
?- log_level(info).
```

Per uscire dal programma e chiudere in maniera pulita la connessione, basta chiamare la regola `out`.

[6] Al momento é in corso un procedimento per raccogliere dati da piú centri di dialisi in tutta Italia, in modo tale da disporre di un insieme di esempi piú ricco e fare meno ricorso a filling-in di dati nulli. Sarebbe inoltre utile rilevare dati per razze diverse da quella caucasica, che copre l'intero *set* di esempi, rendendo l'analisi di tale attributo praticamente inutile.

[7] Ogni regola definita nei diversi moduli é stata documentata. La documentazione é consultabile aprendo in un browser il file `doc/index.html`. Per un problema nel modulo di generazione della documentazione, i link diretti dalla `index.html` alle regole documentate non funzionano; utilizzare, invece, i collegamenti ai moduli, dai quali si puó comunque accedere alla documentazione delle regole.

[8] Alcuni file di configurazione di esempio sono presenti nella

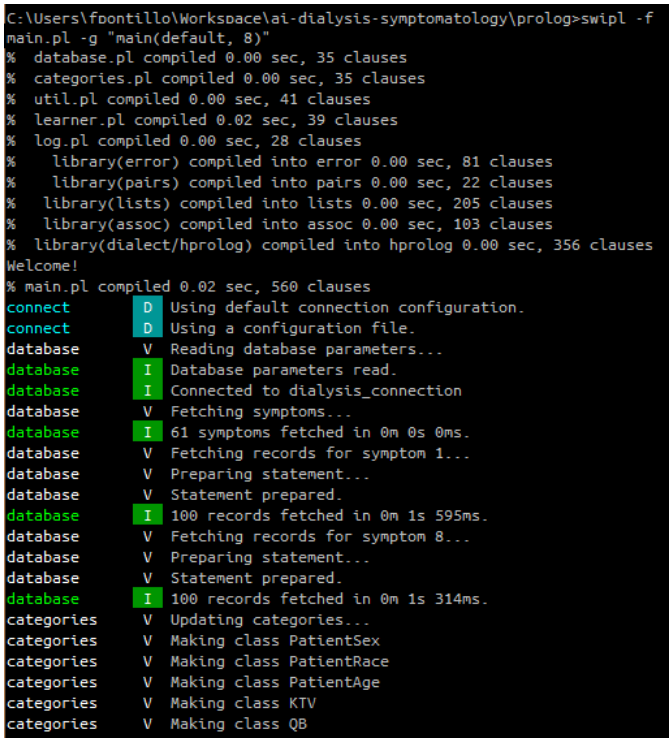

Figure 2: Avvio del programma per il sintomo 8

7.3 Lettura dal database

Una volta letti i parametri impostati dal `main` vengono eseguiti i processi di connessione e lettura dei dati utili alla generazione dell'albero di decisione.

La lettura dal database é possibile grazie alla libreria `odbc` di SWI-Prolog [9].

7.3.1 Connessione

La connessione al database avviene tramite la regola `connect` (nelle varianti) e i parametri nel file di configurazione impostato in precedenza: driver ODBC, indirizzo e porta, username, password e database. Se si é in modalitá `ask`, il file di configurazione verrá chiesto all'utente, e nel caso in cui non esista viene eseguito un *fallback* sul file di default.

La lettura del file `.properties` é eseguita da

```
read_database_params(Path, Driver, Server,
 Port, Database, User, Password)
```

che utilizza il costrutto `open_table` di SWI-Prolog per leggere i campi del file di proprietá e unificare con le variabili passate in input.

7.3.2 Lettura sintomi

Il passo successivo consiste nella lettura di tutte le possibili sintomatologie che potrebbero verificarsi nel corso di una seduta di dialisi. La regola `get_symptoms/0` esegue una semplice `SELECT` sulla base di dati, ottenendo, salvando in memoria e stampando a video i sintomi.

7.3.3 Lettura degli esempi

La regola `update_records/0` si occupa di ottenere e salvare in memoria tutti gli esempi che devono essere utilizzati per avviare il processo di apprendimento.

Poiché é necessaria la selezione sia degli esempi positivi che di quelli negativi, `update_records/0` esegue lo stesso *statement* di `SELECT`, opportunamente creato e preparato, andando a modificare l'ID della sintomatologia target da ottenere[10]. Per lo scopo del progetto é stato imposto un limite di 100 esempi positivi e 100 esempi negativi, in modo tale da velocizzare il processo di generazione delle regole.

Il salvataggio dei record avviene andando ad asserire, nella memoria del programma Prolog, strutture del tipo:

```
positive(ID, Attribute, Value)
negative(ID, Attribute, Value)
```

In questo modo é sempre possibile accedere a qualsiasi coppia attributo-valore di un esempio con uno specifico ID, sia esso positivo o negativo.

É stata realizzata anche la regola

```
example(Type, ID, Attribute, Value)
```

dove `Type` puó essere `positive` o `negative`. Tramite questa modalitá é possibile accedere a tutti gli esempi prelevati dalla base di dati. Sono presenti, inoltre, regole di conteggio e di verifica di esistenza (cfr. documentazione HTML).

7.4 Suddivisione attributi in range

Gli attributi dei dati di esempio possono essere numerici (a virgola mobile) o categorici, ma in ogni caso sono identificati da un numero. Essi sono stati dichiarati esplicitamente tramite la clausola `data_type(Attribute, Type)`, che definisce quindi la tipologia per ogni attributo.

Per ogni attributo, quindi, si é avviato un processo di suddivisione (`update_categories/0` e `make_class/2`) in piú *range*, definite dal predicato `class/2`:

- gli attributi di tipo `category` sono giá automaticamente partizionati, per cui una classe di tipo categorico avrá i range corrispondenti a tutti i valori di categoria
- ogni attributo di tipo `number` é stato suddiviso in 10 range di dimensioni uguali[11]

Il modulo `categories` contiene, nelle sue due varianti, il pre-

cartella `prolog/config`.

[9]Per questa ragione, é necessario che sulla macchina client siano installati i driver di connessione ODBC al database target. Poiché i parametri sono impostati dal programma Prolog, non é necessario creare nessuna connessione sulla macchina.

[10]La regola `get_records/2` prepara lo statement all'esecuzione.

[11]L'approccio utilizzato é semplicistico, quindi passibile di

dicato `is_in_range`, che risulta soddisfatto solo se il valore (numerico o categorico che sia) rientra nel *range* specificato.

7.5 Apprendimento e test

Dopo aver generato le categorie, é possibile avviare il processo (iterativo e ricorsivo) di apprendimento e test tramite la regola `learn_please/0`.

Lo scopo di questa fase é di generare due tipi di regole:

- `is_positive(ID, LearningStep` determina se, secondo le regole generate ad un certo passo `LearningStep`, un esempio con un determinato `ID` é ritenuto un positivo.
- `test_error_rate(LearningStep, ErrorRate)` restituisce l'*error rate* calcolato dal processo di test ad uno specifico passo di apprendimento.

L'iterazione principale del processo di apprendimento é definita dal *k-fold cross-validation*:

1. L'insieme degli esempi (positivi e negativi) viene diviso in k sottoinsiemi. Per lo scopo del progetto, si é impostato un k fisso a 10.
2. Ad ogni iterazione, un sottoinsieme viene messo da parte per la fase di test, mentre gli altri 9 concorrono all'apprendimento vero e proprio.
3. L'apprendimento viene avviato, generando regole di tipo `is_positive/2`.
4. Viene eseguito il test con l'insieme di esempi selezionato, generando regole di tipo `test_error_rate/2`.
5. Si itera dal punto 2 fino ad esaurimento dei k *fold* creati.

L'apprendimento, per ogni fase, é avviato da `learn(Step)`.

7.5.1 Suddivisione in fold

La suddivisione in k fold distinti viene eseguita, ad ogni passo, dal predicato `split_examples/2` (vedi figura 3 per un esempio), che asserisce in memoria alcune liste di `ID` di esempi. Tali liste sono poi utilizzate nel filtraggio, quando richiesto, degli esempi in *training* e *testing*.

Nei primi test del programma, si é notato un bassissimo livello di distribuzione degli esempi nei diversi fold di ogni esecuzione, per cui i primi *run* avevano un bassissimo livello di *error rate* (spesso 0), che cresceva con l'aumentare dell'indice del passo.
Ció era dovuto non alla generazione di regole ottimali, ma al fatto che gli esempi positivi venivano selezionati, da Prolog, sempre dopo quelli negativi, comportando un insieme di test composto, ai primi passi, da molti esempi positivi (mentre erano pochi o nulli alla fine) e da pochi negativi (mentre erano molti alla fine).

miglioramento, vedi 9.

Si é pertanto reso necessario "forzare" una distribuzione piú o meno uniforme andando a dividere i *set* di positivi e negativi in *fold* separati, che sono poi stati comunque accorpati in due liste complessive, visionabili a fine step con:

```
?- train_examples(TrainExamples).
?- test_examples(TestExamples).
```

Il filtraggio degli esempi viene poi eseguito da `test_example/4` e `train_example/4` e da altre regole derivate.

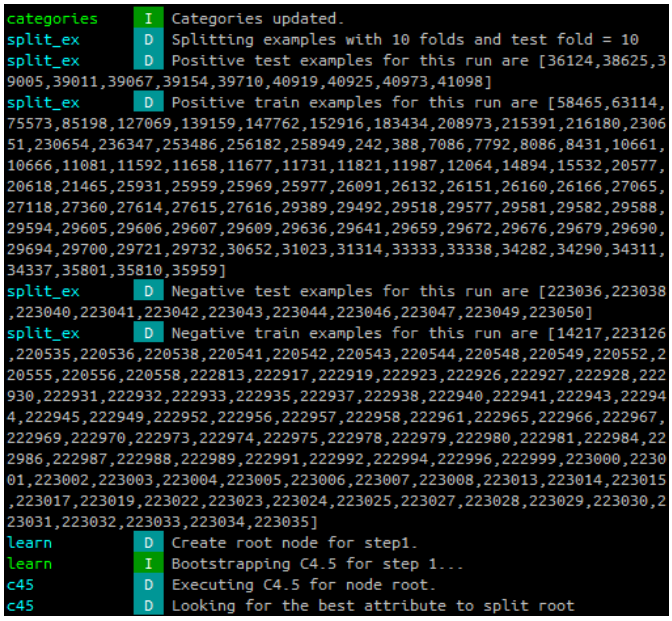

Figure 3: Suddivisione in 10 fold

7.5.2 Apprendimento

L'implementazione dell'algoritmo C4.5 é abbastanza diretta, e si compone di passi ricorsivi che iniziano dalla generazione di un nodo radice.

In generale, tutti i nodi dell'albero di decisione sono cosí composti:

```
node(Name, Parent, SplitAttribute, SplitRange)
```

dove:

- `Name` corrisponde ad un generico nome, che non é detto sia univoco, per il nodo. Genericamente, il nome del nodo é composto dall'attributo su cui il nodo genitore é stato suddiviso e una descrizione testuale del *range* che il nodo rappresenta.
- `Parent` rappresenta il `node/4` genitore; per questa ragione, ogni nodo puó essere identificato univocamente dalla coppia `(Name, Parent)`.
- `SplitAttribute` é l'attributo su cui é stato eseguito lo *split* per il livello corrente dell'albero.

- `SplitRange` é il range dell'attributo di *split* che il nodo corrente espande.

Il nodo radice, non potendo derivare nessuno dei precedenti parametri, ha come valori l'atomo `root`, per semplicitá.

I nodi foglia dell'albero sono identificati dalla regola:

```
node_label(Node, Value)
```

che specifica che un nodo dell'albero é terminale e identifica tutti gli esempi rimanenti come appartenenti a uno specifico range della classe target.

Passi base. Il processo ricorsivo dell'algoritmo C4.5 inizia, quindi, dal nodo `root`. Ogni chiamata alla regola `c45/3` richiede che siano istanziate le seguenti variabili:

- `Node` deve essere un `node/4`, e deve contenere le informazioni sul nodo che si vuole suddividere in piú *split*.
- `Examples` é una lista di `ID` di esempi, ovvero quei dati che devono essere processati al passo corrente.
- `Attributes` è la lista di nomi di attributi che non sono ancora stati selezionati per lo *split*.

Ovviamente il primo passo vedrá `Examples` contenere tutti gli esempi e `Attribute` tutti gli attributi.

Il primo controllo che viene eseguito da `c45/3` verifica se tutti gli `Examples` appartengono giá ad un'unica classe target, ovvero se tutti gli esempi hanno una stessa sintomatologia. In caso affermativo:

1. Si asserisce in memoria un `node_label/2` che dichiara che il nodo corrente ha come unica classe target quella rilevata.
2. Il nodo corrente non viene piú espanso.
3. Viene restituito il controllo al chiamante, ovvero il livello (nodo) superiore.

Se tale controllo non é soddisfatto, se ne fa un altro relativamente alla lista di attributi rimanenti; se la lista é vuota, infatti, si segue lo stesso procedimento per il controllo precedente, con l'unica differenza che il range associato a `node_label` é quello relativo al valore piú comune fra gli esempi rimanenti.

Se nessuna delle verifiche precedenti porta a risultati, il nodo corrente dovrá essere nuovamente diviso in *split*. Il processo, allora, prosegue con:

1. Selezione del migliore attributo.
2. Split sui range dell'attributo selezionato.
3. Chiamata ricorsiva a `c45/3`.

Attributo migliore. Per poter decidere quale, fra gli attributi rimanenti, costituisce il migliore ai fini del processo di apprendimento, si é deciso di utilizzare la misura dell' *information gain* (vedi 5.3.2), che a sua volta si basa fortemente sull'entropia (vedi 5.3.1). Un esempio di esecuzione é visibile in figura 4, dove viene scelto `KTV` poiché possiede il massimo *information gain* di 1.

L'entropia viene calcolata dalla regola `entropy/2`, che utilizza i `train_example/4`; l'*information gain*, invece, viene calcolato da `info_gain/3`, che si basa sul calcolo dell'entropia e che somma, per ogni range dell'attributo A, il valore restituito da `partial_info_gain/4`:

$$\frac{|S_a|}{|S|} \times entropia(S_a)$$

```
learn       D  Create root node for step1.
learn       I  Bootstrapping C4.5 for step 1...
c45         D  Executing C4.5 for node root.
c45         D  Looking for the best attribute to split root
info_gain   V  Calculating info gain for PatientSex...
info_gain   V  Info gain for PatientSex is 0.07515755532773815
info_gain   V  Calculating info gain for PatientRace...
info_gain   V  Info gain for PatientRace is 0.0
info_gain   V  Calculating info gain for PatientAge...
info_gain   V  Info gain for PatientAge is 0.7032793449801468
info_gain   V  Calculating info gain for KTV...
info_gain   V  Info gain for KTV is 1.0
info_gain   V  Calculating info gain for QB...
info_gain   V  Info gain for QB is 0.1434201518017706
info_gain   V  Calculating info gain for ProgWeightLoss...
info_gain   V  Info gain for ProgWeightLoss is 0.062292921756616604
info_gain   V  Calculating info gain for RealWeightLoss...
info_gain   V  Info gain for RealWeightLoss is 0.0734317467007154
info_gain   V  Calculating info gain for DeltaWeight...
info_gain   V  Info gain for DeltaWeight is 0.026053752261298135
info_gain   V  Calculating info gain for ProgDuration...
info_gain   V  Info gain for ProgDuration is 0.32998618521769096
info_gain   V  Calculating info gain for RealDuration...
info_gain   V  Info gain for RealDuration is 0.18064858000071204
info_gain   V  Calculating info gain for DeltaDuration...
info_gain   V  Info gain for DeltaDuration is 0.11052656535893546
info_gain   V  Calculating info gain for SAPStart...
info_gain   V  Info gain for SAPStart is 0.09333995695131014
info_gain   V  Calculating info gain for SAPEnd...
info_gain   V  Info gain for SAPEnd is 0.05037892920413467
info_gain   V  Calculating info gain for SAPAverage...
info_gain   V  Info gain for SAPAverage is 0.0909627176178609
info_gain   V  Calculating info gain for DAPStart...
info_gain   V  Info gain for DAPStart is 0.18190123868502994
info_gain   V  Calculating info gain for DAPEnd...
info_gain   V  Info gain for DAPEnd is 0.04742966114110625
info_gain   V  Calculating info gain for DAPAverage...
info_gain   V  Info gain for DAPAverage is 0.15659893130789448
info_gain   V  Calculating info gain for DeltaBloodFlow...
info_gain   V  Info gain for DeltaBloodFlow is 0.4003172022627709
info_gain   V  Calculating info gain for DeltaUF...
info_gain   V  Info gain for DeltaUF is 0.028464787900896793
best_attr   D  Best info gain is achieved with attribute KTV with a value of 1.0
best_attr   D  Best attribute calculus took 0m 1s 329ms.
```

Figure 4: Scelta dell'attributo migliore secondo l'*information gain*

Prima di ogni calcolo di entropia o di information gain, l'insieme degli esempi da analizzare S per un certo attributo A viene ripulito da tutti i dati che hanno un valore nullo in corrispondenza dell'attributo A.

Split sui range. Dopo aver calcolato l'*information gain* per tutti gli attributi, si seleziona quello con la misura piú elevata e si ottengono tutti i suoi *range*.

Creazione nodi figli. Per ogni range dell'attributo si crea un nuovo nodo avente come nome il template seguente:

```
[ Attribute : range(Inizio, Fine) ]
```

Si chiama, infine, la regola `c45` con parametri:

- `Node` il nodo appena creato
- `ParentNode` il nodo correntemente in analisi e in ingresso al passo corrente
- `Attributes` l'insieme di attributi in ingresso al passo corrente, eccetto l'attributo su cui é stato eseguito lo *split*

Terminazione locale. Quando non ci sono piú esempi da analizzare, o gli attributi su cui eseguire lo *split* sono terminati, l'algoritmo termina, avendo prodotto un albero di decisione completo[12].

7.5.3 Generazione di regole

Una volta appreso l'albero di decisione ad uno specifico passo i, il programma genera le regole `is_positive/2` andando a risalire l'albero di decisione, da ogni nodo foglia positivo `node_label/2`, e aggiungendo in lista una condizione di appartenenza dell'attributo su cui il nodo é stato "splittato" al range del nodo stesso[13].

La regola principale che si occupa della generazione delle regole é `gen_all_the_rulez/1`, che chiama ricorsivamente `gen_rule/2` per ogni nodo foglia.

7.5.4 Test

Dopo la generazione delle regole al passo i, si esegue un processo di test, che asserisce in memoria un *error rate* per il passo in analisi, corrispondente alla frazione di falsi positivi e falsi negativi sul numero totale di esempi di test:

$$Error\_Rate_i = \frac{False\_P_i + False\_N_i}{Test\_Examples_i}$$

7.5.5 Terminazione dell'apprendimento

Una volta che tutti i k cicli di apprendimento-test sono terminati, viene stampato un report (da `print_report/0`, vedi figura 5) che stampa l'*error rate* calcolato per ogni esecuzione e, infine, l'*error rate* totale, calcolato come:

$$Error\_Rate = \frac{\sum_{i=1}^{k} Error\_Rate_i}{k}$$

Al termine di ogni processo di apprendimento viene creato un file, `runs/log_{ID}.csv`[14] contenente, per ogni esecuzione: il sintomo, il passo, il numero di regole generate e l'error rate; infine, il file riporta il tempo di esecuzione in secondi,

[12] É possibile avviare il programma, controllandone ogni passo, chiamando manualmente le clausole in `main/2`; in questo modo si puó avviare manualmente il processo di apprendimento e chiamare, alla fine del passo di generazione dell'albero, la regola `print_le_tree/0`, che stampa a video una rappresentazione grafica e formattata dell'albero di decisione prodotto, in cui ogni elemento foglia (positivo) avrá un check verde.

[13] Vengono generate regole per i soli nodi foglia positivi.

[14] Per la creazione dei file di log, é necessario che la cartella `runs` esista giá.

```
report      I  Learning algorithm finished in 9m 13s 0ms.
report      I  Symptom: 8
report      I  Positive examples: 100
report      I  Negative examples: 100
report      I  Total runs: 10
report      I  Runs recap:
report      I    - Run 1 | Rules : 1 | Error Rate : 0
report      I    - Run 2 | Rules : 1 | Error Rate : 0
report      I    - Run 3 | Rules : 1 | Error Rate : 0
report      I    - Run 4 | Rules : 1 | Error Rate : 0
report      I    - Run 5 | Rules : 1 | Error Rate : 0
report      I    - Run 6 | Rules : 1 | Error Rate : 0
report      I    - Run 7 | Rules : 1 | Error Rate : 0
report      I    - Run 8 | Rules : 1 | Error Rate : 0
report      I    - Run 9 | Rules : 1 | Error Rate : 0.2727272727272727
report      I    - Run 10 | Rules : 0 | Error Rate : 0.5
report      I  TOTAL RULES: 9
report      I  TOTAL ERROR: 0.07727272727272727
report      D  Report printed.
save_log    D  Tests written to file.
save_rules  D  Rules written to file.
true.
```

Figure 5: Stampa del report di fine processo

il numero totale di esempi positivi presenti, il numero totale di regole generate e l'errore medio complessivo.

Inoltre, tutte le regole generate durante l'apprendimento vengono salvate nel file `runs/rules_{ID}.pl`[15], ad esempio:

```
is_positive(A, 10) :-
check_condition_list(A,

     [ condition('ProgDuration',
         range(228.0, 231.0)),
       condition('DeltaWeight',
         range(0.0, 0.171028)),
       condition('RealWeightLoss',
         range(1.9050000000000002, 2.438)),
       condition('ProgWeightLoss',
         range(1.854, 2.342)),
       condition('QB',
         range(64.55200160000003,
               72.62100180000003)),
       condition('KTV',
         range(1.39, 9.351)),
       condition('PatientRace',
         range(1, 1)),
       condition('PatientSex',
         range(1, 1)),
       condition('PatientAge',
         range(28108.1, 28617.199999999997))
     ]).
```

8. RISULTATI

Il programma é stato eseguito su 9 sintomi target distinti e con un diverso numero di esempi positivi; la macchina utilizzata per il test ha le seguenti caratteristiche:

- CPU Intel i7-4500U @ 1.80GHz
- 8GB RAM DDR3

[15] Per la verifica delle regole é necessario che sia caricato in memoria Prolog anche la regola `check_condition_list/2`, che serve ad eseguire il *matching* dei range descritti dalla regole generate.

- Samsung SSD EVO 840 250GB (fino a 540 MB/s in lettura, fino a 520 MB/s in scrittura)
- Windows 8.1 x64

I risultati sono stati sintetizzati nelle tabelle 5 e 6 che mostrano, rispettivamente, l'*error rate* medio (assieme al numero massimo di regole generate per ogni passo) e i tempi di esecuzione relativamente ai diversi sintomi analizzati.

Sintomo	Positivi	Error Rate	Regole
2	19	0.26	1
3	56	0.35	1
4	7	0.11	0
5	100	0.18	90
6	34	0.27	0
7	1	0.02	1
8	100	0.08	1
9	5	0.06	93

Table 5: *Error rate* medio su tutti i passi

Sintomo	Positivi	Time
2	19	
3	56	
4	7	
5	100	
6	34	
7	1	
8	100	
9	5	

Table 6: Tempi di esecuzione per sintomo

9. SVILUPPI FUTURI

L'implementazione base del C4.5 realizzata é migliorabile in molti punti.

Un primo miglioramento potrebbe riguardare la generazione dinamica dei range degli attributi: la cardinalitá di ogni range non sarebbe piú fissa, ma varierebbe a seconda della vicinanza dei valori degli esempi o secondo altri criteri.

Inoltre, si potrebbe confrontare l'*error rate* del programma nel caso i valori nulli non vengano riempiti "a monte" del processo (ovvero nel database), e vengano quindi ignorati completamente.

Ad oggi, il programma gestisce gli esempi come strettamente positivi o negativi. Avendo a disposizione circa 60 sintomi, tuttavia, potrebbe risultare utile riadattare alcune parti dell'algoritmo con il fine di classificare, in un unico albero di decisione, tutti i sintomi. Un possibile svantaggio si potrebbe avere relativamente a sintomi che hanno pochi esempi che li soddisfano, e che potrebbero non essere rappresentati da nessuna regola generata. Ci sarebbe, in questo caso, anche da valutare il costo dell'algoritmo, sicuramente piú complesso (si veda il calcolo dell'entropia).

Infine, vista la scarsa distribuzione delle sintomatologie per seduta di dialisi, potrebbe essere utile imporre una certa soglia di tolleranza, ad esempio andando a classificare come nodo foglia anche i nodi che contengono esempi positivi in una certa percentuale (95% anziché 100% come attualmente implementato). In questo modo sarebbe anche possibile definire un ordine di prioritá per le regole generate, da quella "piú certa" a quella con la piú bassa probabilitá.

Miglioramenti implementativi potrebbero essere:

- personalizzazione del numero dei *fold* da utilizzare nella *k-fold cross-validation*
- spostamento dell'intero processo di apprendimento su macchine virtuali nel cloud; é giá in studio una migrazione su Google Compute Engine
- apprendimento multi-threading sui diversi split dei *k-fold*
- apprendimento multi-threading sui diversi *range* dell'attributo migliore ad ogni livello dell'albero
- ordinamento delle regole in base all'*error rate* del passo in cui sono state apprese (diretta proporzionalitá tra ordine regola e ordine *error rate*) e alla loro semplicitá (minor numero di clausole, maggiore rilevanza)

10. REFERENCES

[1] Swi-prolog, Apr. 2014.

[2] R. Bellazzi, C. Larizza, P. Magni, R. Bellazzi, and S. Cetta. Intelligent data analysis techniques for quality assessment of hemodialysis services. In *Proc. of the Workshop on Intelligent Data Analysis and Pharmacology*, 2001.

[3] A. Kusiak, B. Dixon, and S. Shah. Predicting survival time for kidney dialysis patients: a data mining approach. *Comput. Biol. Med.*, 35(4):311–327, May 2005.

[4] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[5] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[6] S. L. Salzberg. Book review: C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. *Mach. Learn.*, 16(3):235–240, Sept. 1994.