Programmazione Logica Induttiva (Inductive Logic Programming, ILP)

- E' il settore dell'Apprendimento Automatico che impiega la programmazione logica come linguaggio di rappresentazione degli esempi e dei concetti

Definizione del problema di ILP

- Formalmente:
- Dati:

  un insieme *P* di possibili programmi

  un insieme E^+ di esempi positivi

  un insieme E^- di esempi negativi

  un programma logico consistente B, tale che

  $B \not\models e^+$, per almeno un $e^+ \in E^+$
- Trovare:

  un programma logico $P \in P$ tale che

  $\forall\ e^+ \in E^+$, $B,P \models e^+$ (P è **completo**)

  $\forall\ e^- \in E^-$, $B,P \not\models e^-$ (P è **consistente**)

Terminologia

- Terminologia:
- B: **background knowledge**, conoscenza a priori, predicati di cui conosciamo già la definizione.
- E^+ E^- costituiscono il **training set**
- P programma **target**, B,P è il programma ottenuto aggiungendo le clausole di P dopo quelle di B.
- *P* viene detto **hypothesis space**. Definisce lo spazio di ricerca. La descrizione di tale spazio delle ipotesi prende il nome di **language bias**.

- B,P |= e si dice "**P copre e**"

Esempio

- Apprendimento del predicato father.

Dati

P : programmi logici contenenti clausole father(X,Y) :- α con α congiunzione di letterali scelti fra

parent(X,Y),parent(Y,X),male(X),male(Y),

female(X),female(Y)

B= { parent(john,mary),male(john),

parent(david,steve),male(david),

parent(kathy,ellen),female(kathy)}

Esempio

E^+={father(john,mary),father(david,steve)}

E^-={father(kathy,ellen),father(john,steve)}

Trovare:

- un programma per calcolare il predicato father che sia consistente e completo rispetto ad E^+, E^-

Possibile soluzione:

father(X,Y):-parent(X,Y),male(X).

Esempio (2)

- Apprendimento di intersection

Dati:

P : programmi logici contenenti clausole int(X,Y,Z):-α con

$\alpha \supseteq$ {null(X),null(Y),null(Z),cons(X1,X2,X),int(X2,Y,W) int(X2,Y,Z),cons(X1,W,Z),member(X1,Y), notmember(X1,Y)}

E^+={int([4,2,6],[5,2,8],[2])}

E^-={int([4,2,6],[5,2,8],[2,6]), int([4,2,6],[5,2,8],[4,2,6])}

Esempio (2)

B: null([]).

cons(X,Y,[X|Y]).

member(X,[X|Y]).

member(X,[Z|Y]) :- member(X,Y).

notmember(X,[]).

notmember(X,[Z|Y]) :- X ≠ Z,notmember(X,Y).

Trovare:

- un programma per calcolare l'intersezione che sia consistente e completo rispetto ad E^+, E^-

Esempio (2)

Possibile soluzione:

P1:

- int(X,Y,Z) :- null(X), null(Z).
- int(X,Y,Z) :- cons(X1,X2,X),member(X1,Y),int(X2,Y,W), cons(X1,W,Z).
- int(X,Y,Z) :- cons(X1,X2,X),int(X2,Y,Z).

Attenzione: P1 deriva int([1],[1],[]), che è falso.

Non è garantito che il programma sia consistente anche per gli esempi che non sono nel training set!

Aree di Applicazione [Lav94]

- acquisizione di conoscenza per sistemi esperti
- knowledge discovery nei database: scoperta di informazioni potenzialmente utili dai dati memorizzati in un database
- scientific knowledge discovery: generazione di una teoria scientifica a partire da osservazioni + generazione di esperimenti discriminanti + test della teoria
- software engineering: sintesi di programmi logici
- inductive engineering: costruzione di un modello di un dispositivo partendo da esempi del suo comportamento.

Applicazioni di successo

- predizione della relazione struttura - attività nella progettazione di medicine
- predizione della struttura secondaria delle proteine, importante per determinare l'attività di una medicina.
- predizione della mutagenesi dei composti aromatici, al fine di prevedere la carcinogenesi.
- classificazione dei documenti in base alla loro struttura
- diagnosi di guasti in apparati elettromeccanici
- regole temporali per la diagnosi di guasti ai componenti di satelliti

Classificazione dei sistemi

- Esistono diverse dimensioni di classificazione dei sistemi di ILP [Lav94].
- Possono richiedere che tutti gli esempi siano dati all'inizio (**batch learners**) oppure possono accettarli uno ad uno (**incremental learners**).
- Possono apprendere un solo predicato alla volta (**single predicate learners**) oppure più di uno (**multiple predicate learners**).
- Durante l'apprendimento possono fare domande all'utente per chiedere la validità delle generalizzazioni e/o per classificare esempi generati dal sistema (**interactive learners**) oppure non avere interazione con l'esterno (**non-interactive learners**).

Classificazione dei sistemi

- Infine, un sistema può imparare una teoria da zero oppure partendo da una teoria preesistente incompleta e/o inconsistente (**theory revisors**).
- Sebbene si tratti di dimensioni indipendenti, i sistemi esistenti appartengono a due gruppi opposti.
- Da una parte ci sono i sistemi batch, non interattivi che apprendono da zero un concetto alla volta (**empirical systems**), dall'altro ci sono i sistemi incrementali, interattivi che fanno theory revision e imparano più predicati alla volta (**interactive or incremental systems**)

Relazione di generalità

- I sistemi apprendono una teoria compiendo una ricerca nello spazio delle clausole ordinato secondo la relazione di generalità.
- Nel caso della programmazione logica induttiva la relazione di generalità e' basata su quella di θ-sussunzione [Plo69]:
- La clausola C1 **θ-sussume** C2 se esiste una sostituzione θ tale che $C1\theta \subseteq C2$.
- Si dice che una clausola C1 e' almeno tanto generale quanto una clausola C2 (e si scrive $C1 \geq C2$) se C1 θ-sussume C2
- C1 e' piu' generale di C2 ($C1 > C2$) se $C1 \geq C2$ ma non $C2 \geq C1$

Esempi di relazione di θ-sussunzione

C1=grandfather(X,Y)←father(X,Z).

C2=grandfather(X,Y)←father(X,Z),parent(Z,Y).

C3=grandfather(john,steve)←father(john,mary), parent(mary, steve).

C1 θ-sussume C2 con la sostituzione vuota θ=∅

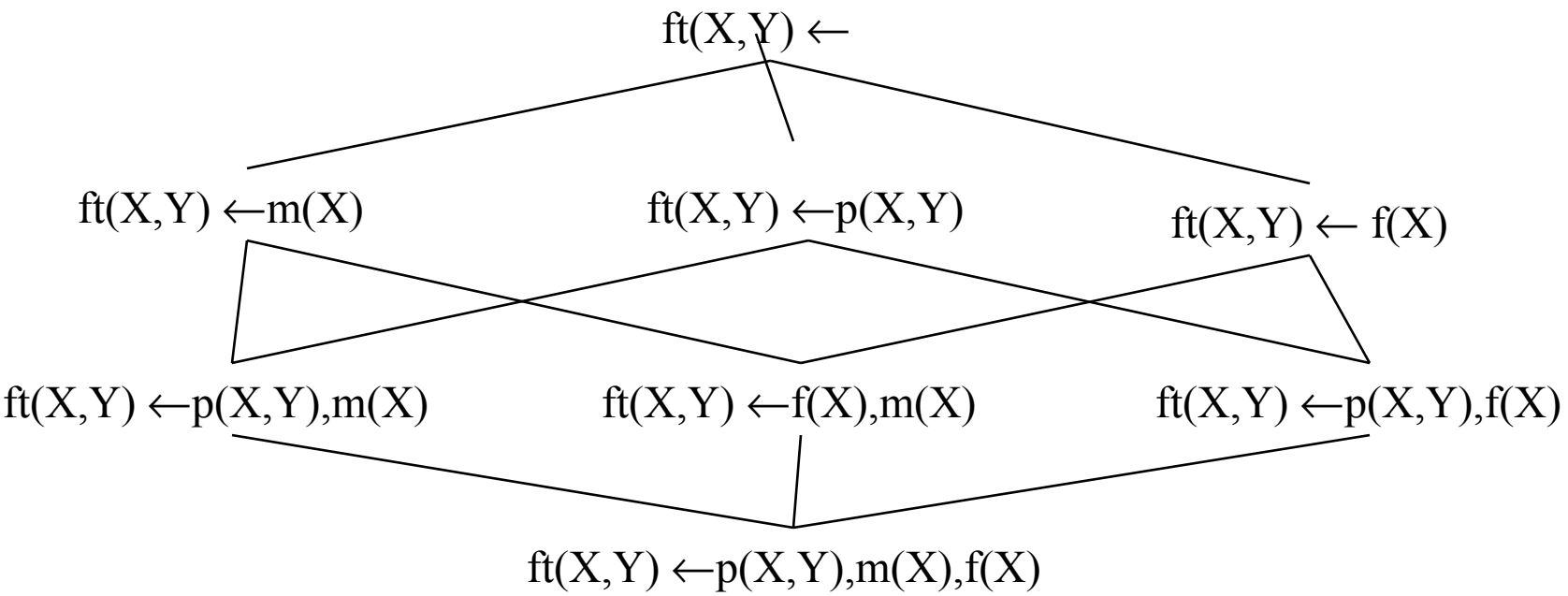
C1 θ-sussume C3 con la sostituzione θ={X/john,Y/steve,Z/mary}.

C2 θ-sussume C3 con la sostituzione θ={X/john,Y/steve,Z/mary}.

Proprieta' della θ-sussunzione

- La θ-sussunzione ha l'importante proprietà che se C1 θ-sussume C2 allora C1 |= C2.
- Non e' pero' vero viceversa.
- Per questa ragione la θ-sussunzione viene utilizzata per rappresentare la relazione di generalità: essa approssima la relazione di generalità intuitiva che è basata sulla relazione di conseguenza logica.
- La θ-sussunzione ha un'altra importante proprietà: introduce un reticolo nell'insieme delle clausole. Questo significa che ciascuna coppia di clausole ha un least upper bound (lub) e un greatest lower bound (glb).

Reticolo di θ-sussunzione



Algoritmi

- Gli algoritmi per ILP si dividono in **bottom-up** e **top-down** [Ber96] a seconda che utilizzino operatori di generalizzazione o specializzazione.
- Gli algoritmi bottom-up partono dagli esempi (specifici o bottom) e li generalizzano ottenendo una teoria che li copre.
  - Vengono utilizzati degli operatori di generalizzazione ottenuti invertendo le regole deduttive di unificazione, risoluzione e implicazione.

Algoritmi top down

- Gli algoritmi top-down imparano programmi generando le clausole in maniera sequenziale (una dopo l'altra). La generazione di una clausola avviene partendo da una clausola generale (con il body vuoto) e specializzandola (applicando un operatore di specializzazione o raffinamento) fino a che non copre più nessun esempio negativo.

Algoritmi top-down

- Operatore di raffinamento:

$\rho(C)=\{C' | C' \in L, C > C'\}$

dove L e' lo spazio delle possibili clausole.

Tipicamente un operatore di raffinamento genera solo le clausole che sono specializzazioni minime (ovvero le più generali) della clausola.

Un operatore di raffinamento tipico applica due operazioni sintattiche ad una clausola:

- applica una sostituzione alla clausola
- aggiunge un letterale al body

Algoritmo top-down di base

Algoritmo ImparaTopDown(E,B;H): dato il training set E e la background B restituisce H

H :=̃

E_{ar} :=E

repeat /* ciclo di covering */

GeneraClausola(Ecur,B;C)

Aggiungi la clausola alla teoria H:=H∪{C}

E_{ar} :=E_{ar} -covers(B,H,E_{ar}) Rimuovi da E_{ar} gli esempi positivi coperti da H'

until il criterio di Sufficienza e' soddisfatto

Ciclo di specializzazione

GeneraClausola(E,B;C): dato il training set E e la background B restituisce la clausola migliore C

Scegli un predicato p da imparare

C := P(X) :- true.

repeat /* ciclo di specializzazione */

Trova il raffinamento $C_{best} \in \rho(C)$ utilizzando una funzione euristica

$C:=C_{best}$

until il criterio di Necessita' e' soddisfatto

return C

Criteri di terminazione

- Gli algoritmo top-down differiscono tra di loro a seconda del criterio di necessita', di sufficienza e dell'euristica che utilizzano per la valutazione della clausola
- In genere, il criterio di Sufficienza richiede che tutti gli esempi positivi siano coperti
- In genere, il criterio di Necessità richiede che nessun esempio negativo sia coperto dalla clausola. Nel caso di dati rumorosi, questo criterio può essere rilassato e si può ammettere la copertura di un certo numero di esempi negativi.

Euristiche

- L'euristica più semplice e' l'accuratezza attesa della clausola

$$A(C)= n^{+}(C)/(n^{+}(C)+n^{-}(C))$$

- dove $n^{+}(C)$ e $n^{-}(C)$, sono, rispettivamente, gli esempi positivi e negativi coperti dalla clausola C
- Un'altra euristica e' l'informativita', definita come

$$I(C)=-\log_2(n^{+}(C)/(n^{+}(C)+n^{-}(C)))$$

- Altre euristiche sono:
  - il guadagno di accuratezza GA(C',C)=A(C')-A(C)
  - il guadagno di informazione GI(C',C)=I(C')-I(C)

Esempio

Esempio

P : father(X,Y) :- α con

$\alpha \supseteq$ {parent(X,Y),parent(Y,X),
male(X),male(Y),female(X),female(Y)}

B= { parent(john,mary),male(john),
parent(david,steve),male(david),
parent(kathy,ellen),female(kathy)}

E^+={father(john,mary),father(david,steve)}

E^-={father(kathy,ellen),father(john,steve)}

Esempio

1° passo di specializzazione

father(X,Y) :- true.

copre tutti gli e^+ ma anche gli e^-

2° passo

father(X,Y) :- parent(X,Y).

copre tutti gli e^+ ma anche l'e^- father(kathy,ellen).

3° passo

father(X,Y) :- parent(X,Y), male(X).

copre tutti gli e^+ e nessun e^-

Gli esempi positivi coperti vengono rimossi, E^+ diventa vuoto e l'algoritmo termina generando la teoria:

father(X,Y) :- parent(X,Y), male(X).

Strategie di ricerca

- L'algoritmo top-down mostrato utilizza una strategia di ricerca di tipo depth-first.
- Alcuni algoritmi invece utilizzano strategie di tipo beam search ovvero mantengono un insieme di clausole di dimensione fissa d

Beam:={C := P(X) :- true}.

repeat /* ciclo di specializzazione */

prendi la prima clausola M dal beam,

aggiungi al beam tutti i raffinamenti ρ(M)

ordina il beam secondo l'euristica e rimuovi le ultime clausole che eccedono la dimensione d

until il criterio di necessita' e' soddisfatto

Strategie di ricerca

- In questo caso il criterio di necessita' richiede che la migliore clausola del beam sia consistente

Osservazioni sugli algoritmi top-down

- **Ricerca non esaustiva**: non viene esplorato lo spazio dei possibili programmi ma solo quello delle possibili clausole. Ogni volta che viene trovata una clausola consistente viene aggiunta alla teoria e non viene più ritratta, non viene fatto il backtracking sulle clausole.
  - Problemi nel caso di predicati ricorsivi o nel caso di predicati multipli
- Problema: non è detto che venga trovata una soluzione anche se questa esiste nello spazio delle ipotesi.
- Compromesso necessario per contenere il tempo di calcolo.

Osservazioni sugli algoritmi top-down

- **Copertura degli esempi**: alcuni sistemi utilizzano un differente criterio per la copertura degli esempi in cui per verificare la copertura di un esempio si usa una sola clausola appresa, la background knowledge e gli esempi (**extensional coverage**). Si parla di sistemi **estensionali** per distinguerli dagli **intensionali**.
- **P copre estensionalmente un esempio e** solo se esiste una clausola $C = l{:-}l_1,l_2,...,l_n$ e una sostituzione θ tale che $e=[l]\theta$ e
- $B,E^+ \models [l_1,l_2,...,l_n]\theta$

FOIL

- FOIL [Qui90,Qui93a,Cam94,Qui91] è tra i piu' diffusi sistemi di ILP. Il codice sorgente C è liberamente scaricabile dalla rete.
  - http://www.cse.unsw.edu.au/~quinlan/foil6.sh
- E' dotato di una efficiente implementazione che lo rende in grado di affrontare problemi reali di apprendimento.
- FOIL è un sistema empirico top-down ed estensionale.
- La background knowledge è fornita a FOIL in forma estensionale, ovvero per ciascun predicato viene fornito l'insieme di tutti i fatti ground veri per quel predicato.

FOIL

- L'algoritmo di FOIL si differenzia dall'algoritmo top-down nei seguenti punti:
  - usa delle euristiche simili a quelle di C4.5, basate sulla teoria dell'informazione, per la selezione del letterale.
  - ricerca depth-first greedy: una volta che un letterale è stato aggiunto ad una clausola, non vengono considerati in backtracking altri letterali;
  - extensional coverage degli esempi.
  - aggiunta di letterali determinati

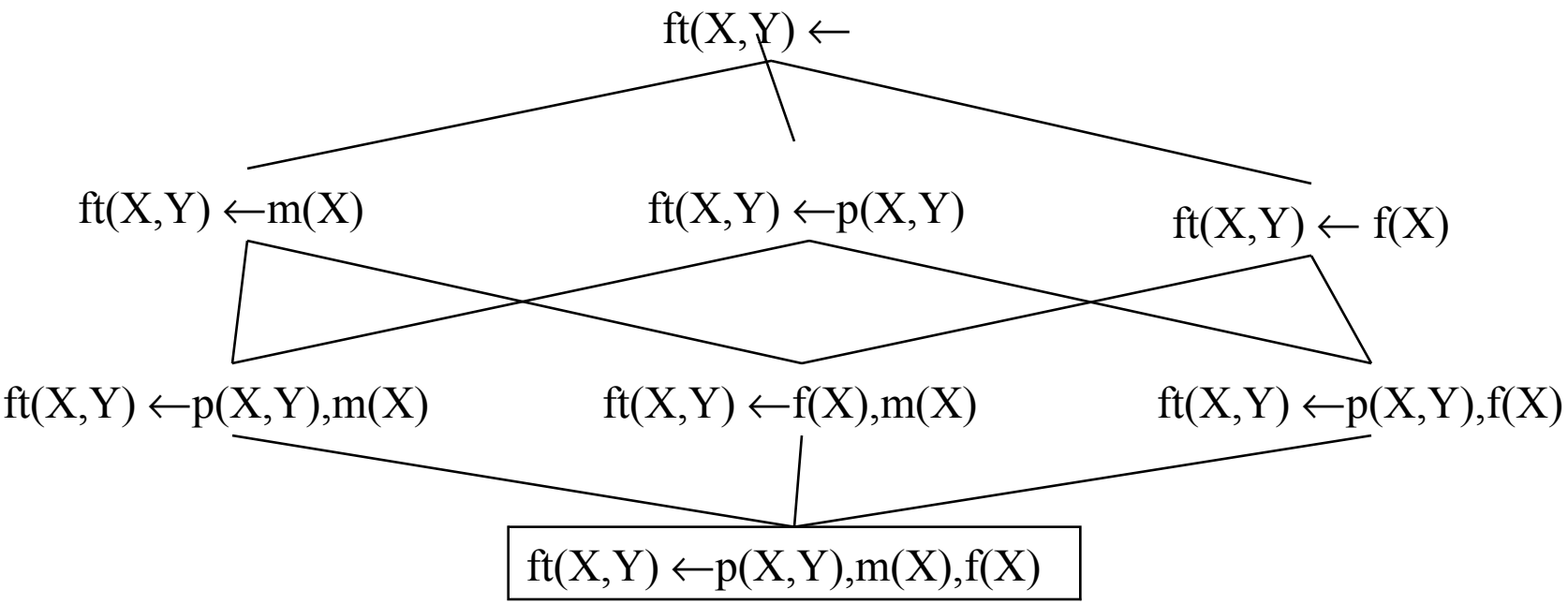
Extensional coverage

- Grazie all'extensional coverage, la generazione di una clausola è completamente indipendente dalle altre e la ricerca nello spazio delle clausole è equivalente a quella nello spazio dei programmi.
- Questo si paga però con il fatto che i programmi appresi possono essere incompleti e inconsistenti:
  - Un programma completo e consistente secondo l'extensional coverage può non esserlo secondo la definizione di ILP (copertura intensionale)
  - Questo si verifica quando si apprendono programmi ricorsivi o contenenti più predicati

Progol

- Progol [Mug95] e' un algoritmo top-down con le seguenti caratteristiche
- utilizza una strategia di tipo beam-search.
- **utilizza una clausola più specifica per limitare dal basso lo spazio di ricerca**
- utilizza una euristica chiamata "compressione"
- e' scritto in C

Clausola più specifica



Clausola più specifica

- Abbiamo sempre considerato la clausola piu' generale per poi specializzarla
  - Si parte da top (lub) del reticolo e si scende fino a che non ci sono esempi negativi coperti.

- Si puo' pensare di avere anche una clausola che limiti da sotto il reticolo, il greatest lower bound del reticolo: $\perp$
- Quindi un sottoinsieme delle soluzioni per H può essere trovato considerando le clausole che θ-sussumono

$\forall$ $\perp$ prende il nome di clausola più specifica

Algoritmi bottom-up

- Gli algoritmi bottom-up sono basati sul concetto di least general generalization (lgg). L'lgg di due clausole C1 e C2 è una clausola C che:
  - è più generale di C1 e C2,
  - è la più specifica tra le clausole più generali di C1 e C2.

Least general generalization

- La θ-sussunzione ha l'importante proprietà che introduce un reticolo nell'insieme delle clausole. Questo significa che ciascuna coppia di clausole ha un least upper bound (lub) e un greatest lower bound (glb).

- **Definizione: least general generalization [Plo69]**
- La least general generalization di due clausole C1 e C2, denotata da lgg(C1,C2), è il least upper bound di C1 e C2 nel reticolo della θ-sussunzione.

Algoritmo per calcolare la lgg

- L'lgg di due **termini** f1(s1,...,sn) e f2(t1,...,tn) è:
  - f1(lgg(s1,t1),...,lgg(sn,tn)) se f1=f2 oppure
  - una nuova variabile V se f1 ≠ f2.
- V è usata per rappresentare l'lgg di f1(s1,...,sn),f2(t1,...,tn) e deve essere utilizzata per tutte le occorrenze dell'lgg degli stessi termini.
- Esempi:
  - lgg(f(a,b,c),f(a,c,d))=f(a,X,Y)
  - lgg(f(a,a),f(b,b))=f(lgg(a,b),lgg(a,b))=f(X,X)
- Si noti che la stessa variabile X è utilizzata in entrambi gli argomenti del secondo esempio perché indica l'lgg degli stessi due termini a e b.

Algoritmo per calcolare la lgg

- L'lgg di due **letterali** L1=(~)p(s1,...,sn) e L2=(~)q(t1,...,tn) è indefinito se L1 e L2 non hanno lo stesso simbolo predicativo e segno, altrimenti è definito come

- lgg(L1,L2)=(~) p(lgg(s1,t1),... lgg(sn,tn))
- **Esempi:**
  - lgg(parent(john,mary),parent(john,steve))=parent(john,X)
  - lgg(parent(john,mary),~ parent(john,steve))=indefinito
  - lgg(parent(john,mary),father(john,steve))=indefinito

Algoritmo per calcolare la lgg

L'lgg di due clausole C1={L1,...,Ln} C2={K1,...,Km} è definito come:

lgg(C1,C2)={lgg(Li,Kj) | Li $\in$ C1, Kj $\in$ C2 e lgg(Li,Kj) è definito}

Lgg di due clausole

Esempi:

C1=father(john,mary) ← parent(john,mary),male(john)

C2=father(david,steve) ← parent(david,steve),male(david)

lgg(C1,C2)=father (X,Y) ← parent(X,Y),male(X)

C1=win(conf1) ← occ(place1,x,conf1),occ(place2,o,conf1)

C2=win(conf2) ← occ(place1,x, conf2),occ(place2,x, conf2)

lgg(C1,C2)=win(Conf) ← occ(place1,x,Conf),occ(L,x,Conf),
occ(M,Y,Conf), occ(place2,Y,Conf)

Metodi bottom-up

- Nei metodi bottom-up, le clausole sono generate partendo dalla clausola più specifica che copre uno o più esempi positivi e nessun esempio negativo e applicando ripetutamente operatori di generalizzazione alla clausola finchè non può più essere ulteriormente generalizzata senza coprire esempi negativi.
- In genere inoltre è presente un passo ulteriore in cui si eliminano letterali dal body della clausola generata finchè la clausola non copre degli esempi negativi

Risorse disponibili in rete

Sito dell'organizzazione MLNET: www.mlnet.org

Contiene link a conferenze, gruppi di ricerca, materiale didattico, sistemi e dataset relativi all'apprendimento automatico

In particolare: materiale didattico

http://kiew.cs.uni-dortmund.de:8001/

Sito specifico sulla ricerca in ILP in Europa (ILNet2)

http://www.cs.bris.ac.uk/~ILPnet2/

Libro:

"Inductive Logic Programming Techniques and Applications" Nada Lavrac and Saso Dzeroski

http://obelix.ijs.si/SasoDzeroski/ILPBook/

Archivi di set di dati in rete

Dataset si possono trovare sul sito di MLNET e su quello di ILPNet2.

Inoltre:

Raccolta di dati di tipo attributo valore:

University of California at Irvine Machine Learning Repository

http://www.ics.uci.edu/~mlearn/MLRepository.html

Motore di ricerca di articoli su Machine Learning (e in generale su computer science)

http://researchindex.org/cs

Bibliografia

[Ber96] F. Bergadano e D. Gunetti, *Inductive Logic Programming - From Machine Learning to Software Engineering*, MIT Press, Cambridge, Massachusetts, 1996

[Cam94] R. M. Cameron-Jones e J. Ross Quinlan, *Efficient Top-down Induction of Logic Programs*, SIGART, 5, pag. 33—42, 1994.

[Lav94] N. Lavrac and S. Dzeroski, *Inductive Logic Programming Techniques and Applications*, Ellis Horwood, 1994

Bibliografia

[Mug95] Stephen Muggleton, *Inverse Entailment and Progol*, New Gen. Comput., 13:245-286, ftp://ftp.cs.york.ac.uk/pub/ML_GROUP/Papers/InvEnt.ps.gz.

[Mug90] S. Muggleton e C. Feng, *Efficient induction of logic programs*, Proceedings of the 1st Conference on Algorithmic Learning Theory Ohmsma, Tokyo, pag. 368-381, 1990.

[Plo69] G.D. Plotkin. *A note on inductive generalisation*. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 153-163. Edinburgh University Press, Edinburgh, 1969.

Bibliografia

[Qui91] J. R. Quinlan, *Determinate literals in inductive logic programming*, Proceedings of Twelfth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1991.

[Qui90] J. R. Quinlan, *Learning logical definitions from relations*, Machine Learning, 5:239-- 266, 1990.

[Qui93a] J. R. Quinlan e R. M. Cameron-Jones, *FOIL: A Midterm Report*, Proceedings of the 6th European Conference on Machine Learning, Springer-Verlag", 1993.