Università degli Studi di Bari

DIPARTIMENTO DI INFORMATICA
Corso di Laurea Magistrale in Informatica

Progetto di Intelligenza Artificiale

Confronto tra due algoritmi di apprendimento

Esaminando:
Giuseppe Rizzi
Matricola 591275

Docenti:
Prof. Floriana Esposito
Prof. Nicola Di Mauro

Indice

Capitolo 1: Introduzione

Capitolo 2: Descrizione dei dati

Di seguito vengono descritti i 4 dataset utilizzati nella sperimentazione

2.1 German credit dataset

Capitolo 3: REPTree

Come primo algoritmo si è scelto di usare **REPTree**, che costruisce alberi di decisione usando l'*information gain* per i valori nominali e la varianza per i valori numerici[16].

Visto che sono stati presi in considerazione dataset con attributi di classe nominali per un task di classificazione e non di regressione, verrà discusso il criterio dell'information gain, analogo all'algoritmo *C4.5*[9].

3.1 Information Gain

Per selezionare l'attributo che meglio classifica i dati D ed in particolare, su quale dei suoi valori occorre fare uno *split*, può convenire usare l'*entropia*[13], ossia l'incertezza contenuta nei dati, che è calcolata come:

$$E(D) = -\sum_i p_i \log_2 p_i$$

cioè la media dei logaritmi delle probabilità di ciascun oggetto i pesato per la probabilità stessa. Nel contesto di classificazione, gli oggetti sono gli esempi nel dataset di cui viene calcolata la probabilità che essi appartengano o meno ad una delle classi presenti nell'attributo target. Più è probabile che un esempio appartenga ad una certa classe, più la sua influenza nel calcolo della media sarà mitigata dal logaritmo della sua stessa probabilità.

È possibile calcolare l'entropia anche per sottoinsiemi del dataset, in particolare quegli esempi D_v che presentano lo stesso valore v di un certo attributo a, poi sommare tutte le entropie relative a tutti valori V dell'attributo per ottenere l'entropia dei dati dopo aver preso in considerazione l'attributo a. Ogni entropia viene pesata per il numero di esempi che presentano quel valore diviso per il totale di esempi esistenti nel dataset:

$$E(D|a) = \sum_{v \in V} \frac{|D_v|}{|D|} \cdot E(D_v)$$

Da queste formule si ricava l'information gain, cioè la riduzione di incertezza totale prendendo in considerazione un attributo a:

$$IG = E(D) - E(D|a)$$

Come radice verrà utilizzato l'attributo che massimizza l'information gain, come archi i valori dell'attributo e si ripete la procedura per i nodi figli fino a generare le foglie.

3.2 Reduced Error Pruning

Per evitare l'*overfitting*, ossia un sovra-adattamento del modello ai dati di training che compromette la bontà delle sue predizioni su nuovi esempi, può essere ragionevole semplificare il modello, rischiando di commettere qualche errore ma garantendoci una migliore copertura per dati non visti.

Questa semplificazione viene chiamata *pruning* (potatura), in cui, una volta costruito il modello utilizzando i dati del *growing set*, esso viene testato su una parte dei dati, accantonati e non adoperati per la predizione, che fanno parte del *pruning set*.

Una tecnica di potatura è REP (**R**educed **E**rror **P**runing)[8] che utilizza un pruning set per stimare l'accuratezza dei nodi intermedi e confrontarla con quella dei suoi sottoalberi.

Viene calcolato il guadagno dall'eventuale potatura sottraendo il numero di errori (esempi classificati scorrettamente) al sottoalbero T al numero di errori al nodo radice v del sottoalbero:

$$Gain_{REP} = \varepsilon_T - \varepsilon_v$$

L'albero è potato se il guadagno è positivo quando vengono commessi più errori nell'intero sottoalbero, e non al nodo. C'è un'altra condizione da rispettare per procedere alla potatura: può avvenire solo se il sottoalbero T non ha un sottoalbero che ha un tasso d'errore minore di T stesso (*bottom-up restriction*).

L'algoritmo di REP è il seguente:

- Si parte dall'albero completo e lo si visita in post-ordine.
- Per ogni nodo intermedio v
  - Calcolo l'accuratezza sul pruning set dell'albero completo.
  - Calcolo l'accuratezza sul pruning set rispetto a v e al suo sottoalbero T.

- Se l'accuratezza aumenta, pota. In caso di uguaglianza pota per semplificare (rasoio di Occam).

Inoltre è dimostrato che, tra tutti i possibili sottoalberi potati che è possibile generare, REP trova il sottalbero più piccolo e più accurato rispetto al pruning set[4].

Capitolo 4: RIPPER

Come secondo algoritmo si è scelto di usare **RIPPER**[3], in particolare nella versione implementata da Weka, **JRip**[17].

RIPPER (**R**epeated **I**ncremental **P**runing to **P**roduce **E**rror **R**eduction) è un algoritmo di induzione di regole proposto da `William W. Cohen` nel 1995. Esso si è dimostrato competitivo con C4.5Rules rispetto ai tassi di errore, scala in maniera lineare con il numero di esempi di training e può elaborare in maniera efficiente dataset rumorosi che contengono centinaia di migliaia di esempi. RIPPER si basa su IREP (**I**ncremental **R**educed **E**rror **P**runing))[5], di cui si discuterà nei prossimi paragrafi.

Molte delle tecniche usate nei moderni sistemi di apprendimento di regole sono stati adattate dall'apprendimento degli alberi di decisione. La maggior parte dei sistemi di apprendimento di alberi di decisione usa una strategia di appredimento *overfit-and-simplify* (sovradatta-e-semplifica) per gestire dati rumorosi: viene generata un'ipotesi prima facendo crescere un albero complesso che "overfitta" i dati, e poi si semplifica o pota tale albero (un'operazione di *pruning*). Una tecnica di pruning efficace è *reduced error pruning* (*REP*), discussa in 3.2. Essa può essere facilmente adattata ai sistemi di apprendimento di regole[6][1].

In REP per le regole, il training set viene diviso in *growing set* e *pruning set*. All'inizio, viene creato un *rule set* di partenza che overfitta il growing set, usando qualche metodo euristico. Questo rule set spropositato viene poi semplificato ripetutamente applicando qualche operatore di pruning. Ad ogni fase di semplificatione, l'operatore di pruning scelto è quello che produce la più grande riduzione di errore sul pruning set. La semplificazione finisce quando il tasso di errore non si riduce ulteriormente applicando gli operatori di pruning.

REP per le regole di solito migliora davvero la perfomance di generalizzazione sui dati rumorosi[6][1][14][5]; tuttavia è computazionalmente costoso per grandi dataset[2].

In risposta all'inefficienza di REP, Fürnkranz e Widmer [1994] proposero un algoritmo di apprendimento chiamato *incremental reduced error pruning*

(*IREP*)[5].

4.1 Incremental Reduced Error Pruning

L'idea di usare un pruning set separato per la potatura è REP. La variante che pota una regola subito dopo averla "fatta crescere" si chiama *incremental reduced error pruning* (IREP)[15]. Quest'ultima integra saldamente REP con un algoritmo di apprendimento di regole *separate-and-conquer*. L'algoritmo 1 ne presenta una versione a due classi. Come ogni algoritmo separate-and-conquer standard, IREP costruisce un ruleset in maniera *greedy*, una regola alla volta. Dopo averne trovata una, tutti gli esempi coperti da quella regola (sia positivi che negativi) sono cancellati. Questo processo si ripete finché non ci sono più esempi positivi, o finché la regola trovata da IREP non presenta un grande tasso di errore, cosa inaccettabile.

Per costruire una regola, IREP usa la seguente strategia. Prima, gli esempi non coperti sono partizionati a caso in due sottoinsiemi, un growing e un pruning set. Nell'implementazione di Cohen, il growing set contiene 2/3 degli esempi.

Poi, una regola viene "fatta crescere". L'implementazione di Cohen di *GrowRule* è una versione proposizionale di FOIL (First Order Inductive Learner), dove i letterali non si servono di predicati ma di uguaglianze (per valori discreti) e confronti numerici (per valori continui)[12]. Esso inizia con una congiunzione vuota di condizioni (la regola vuota) e considera di aggiungere a questa qualsiasi condizione nella forma $A_d = v$, $A_c \leq \theta$ oppure $A_c \geq \theta$ dove A_d è un attributo discreto e v è un valore che può assumere, mentre A_c è un attributo continuo e θ è un valore soglia. *GrowRule* aggiunge ripetutamente la condizione che massimizza un'euristica di *information gain*, nello specifico quella di FOIL, finché la regola non copre più esempi negativi nel growing set.

Siano R_0 e R_1 due regole, la seconda ottenuta dall'aggiunta di una condizione nel corpo della prima. L'information gain viene così calcolato:

$$Gain_{IREP}(R_0, R_1) = t \cdot \left(\log \frac{p_1}{p_1 + n_1} - \log \frac{p_0}{p_0 + n_0} \right)$$

dove t riguarda gli esempi positivi coperti da R_0 che soddisfano anche R_1 dopo aver aggiunto una condizione, p_0 (rispettivamente p_1) sono gli esempi positivi coperti da R_0 (rispettivamente R_1) e n_0 (rispettivamente n_1) sono gli esempi negativi coperti da R_0 (rispettivamente R_1).

L'idea alla base è che l'informazione totale che si guadagna è dato dal numero di tuple che soddisfano la nuova condizione moltiplicato l'informazione guadagnata in merito a ciascuna[7].

Dopo aver espanso una regola, essa viene immediatamente potata. Per prunarla, l'implementazione di Cohen cancella qualsiasi sequenza finale di condizioni dalla regola e sceglie l'eliminazione che massimizza la funzione

$$v(Rule, PrunePos, PruneNeg) \equiv \frac{p + (N - n)}{P + N} \tag{4.1}$$

dove P (rispettivamente N) è il numero totale di esempi in $PrunePos$ ($PruneNeg$) e p (n) è il numero di esempi in $PrunePos$ ($PruneNeg$) coperti da $Rule$. Questo processo è ripetuto finché nessun altra cancellazione migliora il valore di v.

L'algoritmo IREP descritto sopra è per i problemi di apprendimento a due classi. L'implementazione di Cohen gestisce classi multiple, come spiegato di seguito:

1. Le classi vengono ordinate secondo la prevalenza, cioè l'ordine è $C_1, ..., C_k$ dove C_1 è la classe di minoranza e C_k è la classe di maggioranza.

2. Viene trovata una regola che separi C_1 dal resto delle classi; questo viene fatto con una singola chiamata ad IREP dove $PosData$ contiene gli esempi di classe C_1 e $NegData$ contiene gli esempi di classi $C_2, C_3, ..., C_k$.

3. Tutte le istanze coperte dal ruleset appena addestrato sono rimosse dal dataset e IREP si appresta a separare C_2 dalle classi $C_3, ..., C_k$.

4. Si ripete finché rimane la sola classe C_k. Quest'ultima verrà usata come classe di default.

L'implementazione di Cohen differisce da quella di Fürnkranz e Widmer sotto molti aspetti. Quando le regole vengono potate, la nuova implementazione permette di cancellare qualsiasi sequenza finale di condizioni, mentre l'implementazione di Fürnkranz e Widmer permette solo la cancellazione di una singola condizione finale. L'algoritmo rivisitato permette anche di fermare l'aggiunta di regole al ruleset quando la regola appresa ha un tasso di errore superiore al 50%, mentre quello di Fürnkranz e Widmer la ferma quando l'accuratezza della regola è minore dell'accuratezza della regola vuota.

Algoritmo 1 IREP(Pos, Neg)

```
1: Ruleset ← ∅
2: while Pos ≠ ∅ do
3:     dividi (Pos, Neg) in (GrowPos, GrowNeg) e (PrunePos, PrunNeg)
4:     Rule ← GrowRule(GrowPos, GrowNeg)
5:     Rule ← PruneRule(Rule, PrunePos, PruneNeg)
6:     if il tasso di errore su (PrunePos, PrunNeg) > 50% then
7:         return Ruleset
8:     else
9:         aggiungi Rule a Ruleset
10:        rimuovi gli esempi coperti da Rule da (Pos, Neg)
11: return Ruleset
```

4.2 Miglioramenti ad IREP

Sono state implementate tre modifiche ad IREP: una metrica alternativa per determinare il valore delle regole nella fase di potatura; una nuova euristica per dedidere quando fermare l'aggiunta di regole al ruleset; un successivo passaggio di "ottimizzazione" del ruleset per tentare di avvicinarsi di più al REP convenzionale (cioè, non incrementale).

Metrica per il valore delle regole

Il fallimento occasionale di IREP a convergere al crescere del numero degli esempi può essere facilmente fatto risalire alla metrica usata per guidare la potatura (ossia la (4.1)). Le scelte intraprese nella definizione di tale metrica non sono intuitive; per esempio (assumendo che P e N siano fissati) la metrica preferisce una regola R_1 che copre $p_1 = 2000$ esempi positivi e $n_1 = 1000$ esempi negativi rispetto ad una regola R_2 che copre $p_2 = 1000$ esempi positivi e $n_2 = 1$ esempio negativo; si noti comunque che R_2 è altamente predittiva, al contrario di R_1. Quindi si è deciso di sostituire la metrica di IREP con

$$v^*(Rule, PrunePos, PruneNeg) \equiv \frac{p - n}{p + n}$$

che sembra avere un comportamento più intuitivo e soddisfacente.

Condizione di stop

L'implementazione di IREP di Cohen si ferma in maniera greedy aggiungendo regole al ruleset quando l'ultima regola costruita ha un tasso d'errore

maggiore del 50% sui dati di pruning. Questa euristica, spesso, si ferma troppo presto con campioni di dimensioni moderate; questo è vero soprattutto quando si apprende un concetto con regole a bassa copertura (pochi esempi coperti).

La soluzione a questo problema è la seguente. Dopo l'aggiunta di ogni regola, viene calcolata la *description-length* totale del ruleset e degli esempi. La nuova versione di IREP ferma l'aggiunta di regole quando questa description-length è maggiore di d bit rispetto alla più piccola description-length ottenuta sinora, o quando non ci sono più esempi positivi. Nell'implementazione si è usato $d = 64$. Il ruleset viene poi semplificato esaminando ogni regola a turno (cominciando dall'ultima) e cancellando regole così da ridurre la description-length totale.

Il principio *MDL* (Minimum Description Length) può essere meglio espresso immaginando un modello di comunicazione in cui un mittente trasmette ad un ricevente una descrizione che consiste in una teoria T e i dati D da cui essa è derivata[11].

Il metodo usato per la codifica è lo stesso usato in *C4.5rules*[10]. Esso parte da un *bias* in cui il numero di falsi positivi e falsi negativi sia lo stesso e si procede come segue: i messaggi da inviare si presentano con probabilità p_j, e servono $-\log(p_j)$ bit (in base 2) per costrurli: più un messaggio è frequente, meno bit saranno necessari per rappresentarlo. Si inviano i dati codificati, poi anziché inviare i messaggi di errore per tutti i dati, il mittente prima trasmette gli errori e nei C casi coperti dalla teoria e poi negli U casi non coperti. Sotto l'assunzione che i falsi positivi fp e i falsi negativi fn siano bilanciati, la probabilità di errore nei casi coperti è $e/2C$ e questa probabilità è usata per codificare i messaggi di errore per i casi coperti. Una volta che i falsi positivi sono stati identificati, il destinatario può calcolare il vero numero dei falsi negativi come $e - fp$, quindi la probabilità di errore oer i casi non coperti è fn/U. Il costo totale quindi diventa:

$$
\begin{aligned}
&\log(|D| + 1) \\
&+ fp \times (-\log(\frac{e}{2C})) \\
&+ (C - fp) \times (-\log(1 - \frac{e}{2C})) \\
&+ fn \times (-\log(\frac{fn}{U})) \\
&+ (U - fn) \times (-\log(1 - \frac{fn}{U}))
\end{aligned}
$$

Ottimizzazione delle regole

L'approccio ripetuto *grow-and-simplify* usato in IREP può produrre risultati abbastanzi differenti dal REP convenzionale (non incrementale). Un modo per migliorarlo è elaborare a posteriori le regole prodotte da IREP così da avvicinarsi di più all'effetto del REP convenzionale. Per esempio, si potrebbe ri-potare ogni regola al fine di minimizzare l'errore del ruleset completo.

Il metodo sviluppato per ottimizzare un ruleset $R_1, R_2, ..., R_k$ consiste del costruire due regole alternative per ogni R_i. La *sostituta* di R_i viene generata espandendo e poi potando R_i. La *revisione* di R_i viene generata in maniera analoga, tranne per il fatto che la revisione è espansa in modo greedy aggiungendo condizioni a R_i, piuttosto che alla regola vuota. Infine si sceglie tra le tre regole quale includere nella teoria. Questa decisione viene presa in base all'euristica MDL. L'implementazione di questo metodo in IREP avviene in questo modo:

1. Viene usato IREP per ottenere un ruleset iniziale.
2. Esso viene ottimizzato, come descritto sopra.
3. Vengono aggiunte le regole in modo tale da coprire gli esempi positivi rimanenti.

L'ottimizzazione può essere ripetuta più volte elaborando il ruleset ottenuto dalla passata precedente dell'algoritmo.

IREP, con l'aggiunta del passo di post-ottimizzazione, forma un nuovo algoritmo che è stato chiamato **RIPPER** (**R**epeated **I**ncremental **P**runing to **P**roduce **E**rror **R**eduction).

L'implementazione in Weka di RIPPER si chiama JRip.

Capitolo 5: Risultati

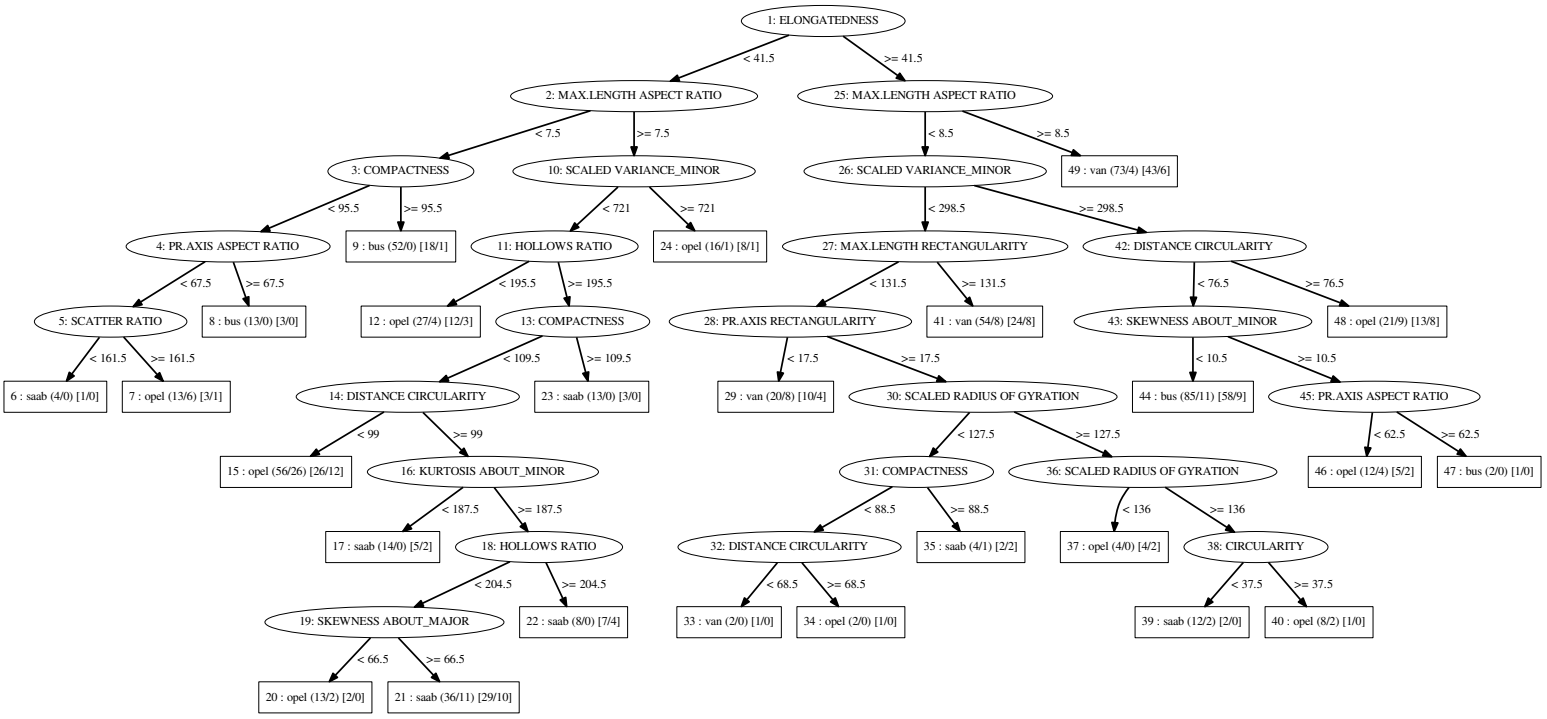

Figura 5.1: Albero

Bibliografia

[1] C. Brunk and M. J. Pazzani. An investigation of noise-tolerant relational concept learning algorithms, 1991.

[2] W. W. Cohen. Efficient pruning methods for separate-and-conquer rule learning systems. In *In Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 988–994. Morgan Kaufmann, 1993.

[3] W. W. Cohen. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[4] F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(5):476–491, May 1997. ISSN 0162-8828.

[5] J. Fürnkranz and G. Widmer. Incremental reduced error pruning, 1994.

[6] G. Pagallo, D. Haussler, and P. Rosenbloom. © 1990 kluwer academic publishers. manufactured in the netherlands. boolean feature discovery in empirical learning, 1990.

[7] J. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990. ISSN 1573-0565.

[8] J. R. Quinlan. Simplifying decision trees. *Int. J. Man-Mach. Stud.*, 27 (3):221–234, Sept. 1987. ISSN 0020-7373.

[9] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0.

[10] J. R. Quinlan. MDL and categorical theories (continued). In *In Machine Learning: Proceedings of the Twelfth International Conference, Lake Taho*, pages 464–470. Morgan Kaufmann, 1995.

[11] J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Inf. Comput.*, 80(3):227–248, Mar. 1989. ISSN 0890-5401.

[12] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003. ISBN 0137903952.

[13] C. E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[14] S. M. Weiss and N. Indurkhya. Reduced complexity rule induction. In *In Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91*, pages 678–684. Morgan Kaufmann, 1991.

[15] I. H. Witten, E. Frank, and M. A. Hall. Decision trees. In *Data Mining: Practical Machine Learning Tools and Techniques*, chapter 6.2, page 206. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011. ISBN 0123748569, 9780123748560.

[16] I. H. Witten, E. Frank, and M. A. Hall. Decision trees. In *Data Mining: Practical Machine Learning Tools and Techniques*, chapter 11.4, page 456. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011. ISBN 0123748569, 9780123748560.

[17] I. H. Witten, E. Frank, and M. A. Hall. Decision trees. In *Data Mining: Practical Machine Learning Tools and Techniques*, chapter 6.11, page 303. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011. ISBN 0123748569, 9780123748560.