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

Il seguente lavoro si propone di confrontare due algoritmi di apprendimento supervisionato, *REPTree* e *RIPPER*: il primo sfrutta la metodologia di costruzione di alberi di decisione, il secondo quello di costruzione di regole.

Verranno testati su quattro dataset messi a disposione dall'*UCI Machine Learning Repository*[1], procedendo con la presentazione dei risultati e dei modelli di predizione ottenuti.

Il software utilizzato è *Weka*[2], una suite di algoritmi di *machine learning*, fortemente utilizzato sia in ambito accademico che industriale.

[1] `https://archive.ics.uci.edu/ml/datasets.html`
[2] `http://www.cs.waikato.ac.nz/ml/weka/`

Capitolo 2: Descrizione dei dati

Di seguito vengono descritti i 4 dataset utilizzati nella sperimentazione

2.1 German Credit dataset

Il dataset contiene informazioni in ambito finanziaro su clienti ritenuti a rischio o meno.

- Numero di istanze: 1000
- Numero di attributi: 21
- Attributo target: **class**
- Valori target: {`good, bad`}

Attributo	Tipo
checking_status	nominal
duration	numeric
credit_history	nominal
purpose	nominal
credit_amount	numeric
savings_status	nominal
employment	nominal
installment_commitment	numeric
personal_status	nominal
other_parties	nominal
residence_since	numeric
property_magnitude	nominal
age	numeric
other_payment_plans	nominal
housing	nominal
existing_credits	numeric
job	nominal
num_dependents	numeric
own_telephone	nominal
foreign_worker	nominal
class	nominal

2.2 Hepatitis dataset

Il dataset contiene informazioni su un vari casi di epatite.

- Numero di istanze: 135
- Numero di attributi: 20
- Attributo target: **Class**
- Valori target: {`DIE, LIVE`}

Attributo	Tipo
AGE	numeric
SEX	nominal
STEROID	nominal
ANTIVIRALS	nominal
FATIGUE	nominal
MALAISE	nominal
ANOREXIA	nominal
LIVER_BIG	nominal
LIVER_FIRM	nominal
SPLEEN_PALPABLE	nominal
SPIDERS	nominal
ASCITES	nominal
VARICES	nominal
BILIRUBIN	numeric
ALK_PHOSPHATE	numeric
SGOT	numeric
ALBUMIN	numeric
PROTIME	numeric
HISTOLOGY	nominal
Class	nominal

2.3 Vehicle Silhouettes dataset

Il dataset contiene informazioni per discriminare le silhouette di diversi veicoli tra automobili, van e bus.

- Numero di istanze: 846
- Numero di attributi: 19
- Attributo target: **Class**
- Valori target: {`opel, saab, bus, van`}

Attributo	Tipo.
COMPACTNESS	numeric
CIRCULARITY	numeric
DISTANCE CIRCULARITY	numeric
RADIUS RATIO	numeric
PR.AXIS ASPECT RATIO	numeric
MAX.LENGTH ASPECT RATIO	numeric
SCATTER RATIO	numeric
ELONGATEDNESS	numeric
PR.AXIS RECTANGULARITY	numeric
MAX.LENGTH RECTANGULARITY	numeric
SCALED VARIANCE_MAJOR	numeric
SCALED VARIANCE_MINOR	numeric
SCALED RADIUS OF GYRATION	numeric
SKEWNESS ABOUT_MAJOR	numeric
SKEWNESS ABOUT_MINOR	numeric
KURTOSIS ABOUT_MAJOR	numeric
KURTOSIS ABOUT_MINOR	numeric
HOLLOWS RATIO	numeric
Class	nominal

2.4 Wisconsin Breast Cancer dataset

Il dataset contiene informazioni riguardo a vari casi di tumore al seno, che permettono di stabilire se esso è benigno o maligno.

- Numero di istanze: 699
- Numero di attributi: 10
- Attributo target: **Class**
- Valori target: {`benign, malignant`}

Attributo	Tipo
Clump_Thickness	numeric
Cell_Size_Uniformity	numeric
Cell_Shape_Uniformity	numeric
Marginal_Adhesion	numeric
Single_Epi_Cell_Size	numeric
Bare_Nuclei	numeric
Bland_Chromatin	numeric
Normal_Nucleoli	numeric
Mitoses	numeric
Class	nominal

Capitolo 3: Naive Bayes

Il ragionamento bayesiano è un approccio probabilistico all'inferenza. Si basa sull'assunzione che i dati sono governati da distribuzioni di probabilità e che possono essere prese decisioni ottimali relative a queste probabilità insieme agli esempi a disposizione. Un modello bayesiano non è complicato da costruire, soprattutto su grandi dataset. Nonostante la sua semplicità, tale classificatore spesso si comporta meglio di altri classificatori più sofisticati[4].

3.1 Teorema di Bayes

Nel contesto di classificazione, quello che interessa è determinare la migliore ipotesi h appartenente ad uno spazio delle ipotesi H e i dati osservati D. Un modo per determinare la *migliore* ipotesi è ricercare la *più probabile*, grazie ai dati a disposizione più una conoscenza iniziale sulle probabilità a priori delle varie ipotesi in H[7]. Il teorema di Bayes fornisce un modo diretto per calcolare queste probabilità, in particolare:

- $P(h)$, la *probabilità a priori* che l'ipotesi h sia valida, prima di aver osservato i dati di training. Riflette una qualche conoscenza pregressa che abbiamo circa la possibiltà che h sia corretta.
- $P(D)$ denota la priorità a priori che i dati di training D saranno osservati, senza fare alcuna considerazione sulle ipotesi.
- $P(D|h)$ denota la probabilità di osservare i dati D in un mondo in cui l'ipotesi h regga.
- $P(h|D)$ è la probabilità che h sia valida dopo aver osservato i dati di training D. Si tratta della *probabilità a posteriori* di h perché riflette la confidenza che h sia corretta dopo aver visto D.

Quello che ci interessa è $P(h|D)$, che è possibile calcolare combinando le probabilità succitate:

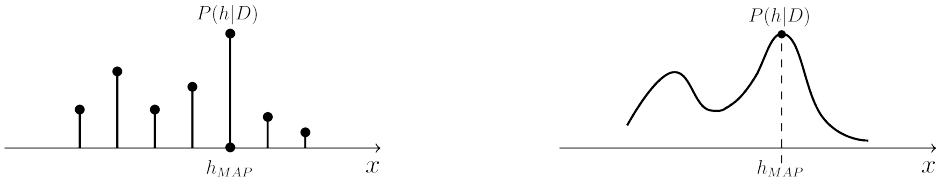
$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Come è facile intuire, $P(h|D)$ aumenta con $P(h)$ e con $P(D|h)$. Analogamente, $P(h|D)$ diminuisce all'aumentare di $P(D)$, perché più è probabile che D venga osservato non considerando h, meno evidenza D fornisce in supporto ad h.

In molti scenari di apprendimento vengono considerate un insieme H di ipotesi candidate e, tra di esse, ci interessa trovare quella più probabile dopo aver osservato i dati D (o almeno quella massimamente più probabile, se ce n'è più di una). Questa ipotesi è chiamata ipotesi *maximum a posteriori* (MAP). Si può determinare l'ipotesi MAP usando il teorema di Bayes per calcolare la probabilità a posteriori di ogni ipotesi candidata, e poi trovare quella che massimizza tale probabilità:

$$\begin{aligned} h_{MAP} &\equiv \underset{h \in H}{\operatorname{argmax}} P(h|D) \\ &= \underset{h \in H}{\operatorname{argmax}} \frac{P(D|h)P(h)}{P(D)} \\ &= \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h) \end{aligned}$$

Il termine $P(D)$ può essere tolto perché è una costante indipendente da h.



(a) Discreto (b) Continuo

Figura 3.1: Rappresentazione di MAP

3.2 Naive Bayes Classifier

Un metodo di apprendimento molto efficace è il classificatore *naive Bayes*. Esso si applica a task di apprendimento in cui ogni istanza è descritta come una congiunzione di valori di attributo $\langle a_1, ..., a_n \rangle$ e dove l'attributo target può assumere qualsiasi valore da un insieme finito V.

Ad una nuova istanza viene assegnato il più probabile valore target v_{MAP}, considerati i valori di attributo $\langle a_1, ..., a_n \rangle$:

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, ..., a_n)$$

Riapplicando le trasformazioni relative alla MAP definite sopra, possiamo riscrivere l'espressione come:

$$\begin{aligned} v_{MAP} &= \underset{h \in H}{\operatorname{argmax}} \frac{P(a_1, ..., a_n | v_j) P(v_j)}{P(a_1, ..., a_n)} \\ &= \underset{h \in H}{\operatorname{argmax}} P(a_1, ..., a_n | v_j) P(v_j) \end{aligned}$$

Ora bisogna stimare le due probabilità sui dati di training. I vari $P(v_j)$ possono essere facilmente calcolati contando la frequenza con cui ogni valore target v_j occorre nei dati. Non è altrettanto semplice calcolare $P(a_1, ..., a_n | v_j)$ allo stesso modo. Il problema è che ci sono molte probabilità da calcolare e pochi dati a disposizione per ottenere delle stime affidabili: servirebbero dataset molto grandi.

Qui entra in gioco il punto cardine del NBC, ossia presupporre che esista l'*indipendenza condizionale* tra gli attributi. In altre parole, l'assunzione è che, dato il valore target, la probabilità di osservare la congiunzione $a_1, ..., a_n$ è semplicemente il prodotto delle probabilità dei singoli attributi:

$$P(a_1, ..., a_n | v_j) = \prod_i P(a_i | v_j)$$

Facendo le opportune sostituzioni otteniamo:

$$v_{NB} = v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j)$$

dove v_{NB} è il valore target restituito da NBC, che è uguale a v_{MAP} quando si assume l'indipendenza condizionale. Si noti che $P(a_i | v_j)$ è semplicemente il numero dei valori di attributo distinti moltiplicato il numero dei valori target distinti, chiaramente un numero molto più piccolo e gestibile rispetto a $P(a_1, ..., a_n | v_j)$ senza l'ipotesi di indipendenza.

L'algoritmo di un NBC è pertanto:

- Calcola i vari $P(v_j)$ e $P(a_i | v_j)$, basandoti sulle loro frequenze sui dati di training.
- Usa queste stime per formare l'ipotesi.

- Ogniqualvolta l'assunzione di indipendenza condizionale è soddisfatta, la classificazione v_{NB} è uguale alla classificazione MAP.

Un aspetto interessante è che NBC non effettua alcuna ricerca esplicita nello spazio delle ipotesi. Viene costruita l'ipotesi semplicemente contando le frequenze delle varie combinazioni dei dati all'interno del training set.

Attributi continui

Contare le frequenze delle diverse combinazioni di valori è possibile per gli attributi discreti, quindi bisogna trovare un modo per calcolare le probabilità degli eventuali attributi continui. Ci sono due strade: la prima è appunto discretizzare i valori numerici in vari *bin*, l'altra è assumere l'esistenza di una distribuzione normale (Gaussiana) per i valori $\langle a_1, ..., a_n \rangle$ dell'attributo A_k. Questa funzione ha bisogno di due parametri, media e deviazione standard degli attributi:

$$\mu_{kj} = E[A_k|v_j] = \frac{1}{n}\sum_{i=1}^{n} a_i$$

$$\sigma_{kj} = \sqrt{E[(A_k - \mu_{kj})^2|v_j]} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(a_i - \mu_{kj})^2}$$

Infine, serve la funzione di densità di probabilità della Gaussiana, che applicata al nostro contesto diventa:

$$P(a_i|v_j) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(a_i-\mu)^2}{2\sigma^2}}$$

Capitolo 4: REPTree

Come altro 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[18].

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*[11].

4.1 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*[15], 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.

4.2 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)[10] 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[5].

Capitolo 5: RIPPER

Come ultimo algoritmo si è scelto di usare **RIPPER**[3], in particolare nella versione implementata da Weka, **JRip**[19].

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))[6], 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 4.2. Essa può essere facilmente adattata ai sistemi di apprendimento di regole[8][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[8][1][16][6]; 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*)[6].

5.1 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)[17]. 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)[14]. 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[9].

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

$$v(Rule, PrunePos, PruneNeg) \equiv \frac{p + (N - n)}{P + N} \tag{5.1}$$

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
```

5.2 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 (5.1)). 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[13].

Il metodo usato per la codifica è lo stesso usato in *C4.5rules*[12]. 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 6: Esecuzione

Qui vengono confrontati i due algoritmi REPTree e RIPPER/JRip. Entrambi hanno sfruttato una *10-fold cross validation*.

6.1 Risultati su `German Credit`

L'esecuzione ha coinvolto 900 istanze di training e 100 istanze di testing ad ogni iterazione del CV.

Esecuzione NaiveBayesSimple

```
=== Run information ===

Scheme:weka.classifiers.bayes.NaiveBayesSimple
Relation:     german_credit
Instances:    1000
Attributes:   21
              checking_status
              duration
              credit_history
              purpose
              credit_amount
              savings_status
              employment
              installment_commitment
              personal_status
              other_parties
              residence_since
              property_magnitude
              age
              other_payment_plans
              housing
              existing_credits
              job
              num_dependents
              own_telephone
```

```
              foreign_worker
              class
Test mode:10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         756               75.6    %
Incorrectly Classified Instances       244               24.4    %
Kappa statistic                          0.3876
Mean absolute error                      0.294
Root mean squared error                  0.4209
Relative absolute error                 69.9613 %
Root relative squared error             91.8429 %
Total Number of Instances             1000

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.864     0.497      0.802     0.864     0.832      0.785    good
                 0.503     0.136      0.614     0.503     0.553      0.785    bad
Weighted Avg.    0.756     0.388      0.746     0.756     0.748      0.785

=== Confusion Matrix ===

   a   b   <-- classified as
 605  95 |   a = good
 149 151 |   b = bad
```

Esecuzione REPTree

```
=== Run information ===

Scheme:weka.classifiers.trees.REPTree -M 2 -V 0.001 -N 3 -S 1 -L -1
Relation:     german_credit
Instances:    1000
Attributes:   21
              checking_status
              duration
              credit_history
              purpose
              credit_amount
              savings_status
              employment
              installment_commitment
              personal_status
```

```
               other_parties
               residence_since
               property_magnitude
               age
               other_payment_plans
               housing
               existing_credits
               job
               num_dependents
               own_telephone
               foreign_worker
               class
Test mode:10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         718               71.8    %
Incorrectly Classified Instances       282               28.2    %
Kappa statistic                          0.2702
Mean absolute error                      0.3417
Root mean squared error                  0.4424
Relative absolute error                 81.3157 %
Root relative squared error             96.532  %
Total Number of Instances             1000

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.859     0.61       0.767     0.859     0.81       0.72     good
                 0.39      0.141      0.542     0.39      0.453      0.72     bad
Weighted Avg.    0.718     0.469      0.699     0.718     0.703      0.72

=== Confusion Matrix ===

   a   b   <-- classified as
 601  99 |   a = good
 183 117 |   b = bad
```

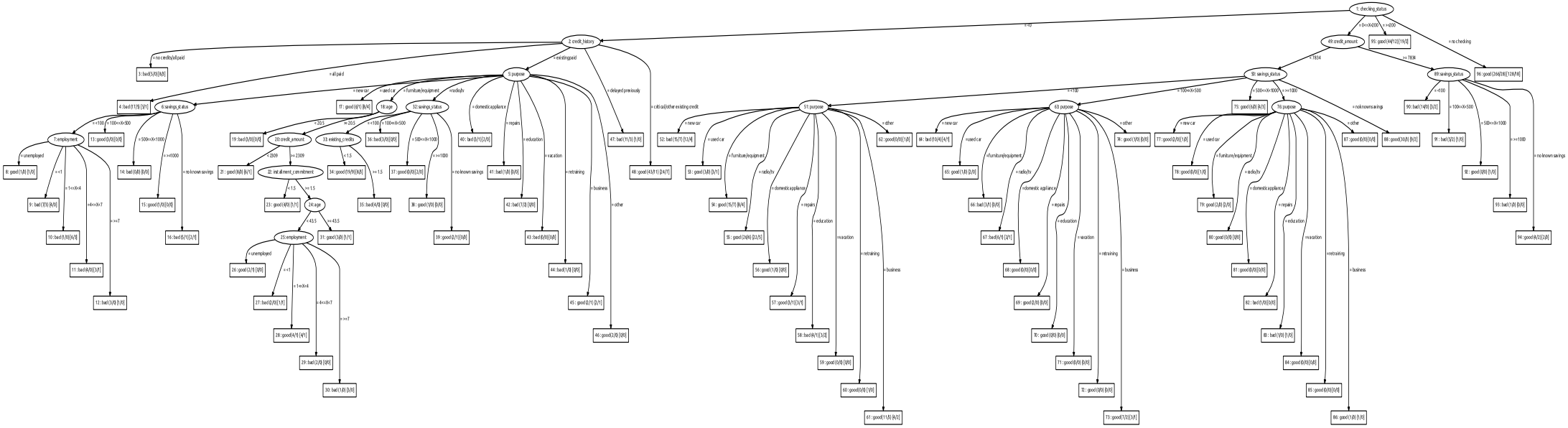


Figura 6.1: Modello di REPTree

Esecuzione JRip

```
=== Run information ===

Scheme:weka.classifiers.rules.JRip -F 3 -N 2.0 -O 2 -S 1
Relation:     german_credit
Instances:    1000
Attributes:   21
              checking_status
              duration
              credit_history
              purpose
              credit_amount
              savings_status
              employment
              installment_commitment
              personal_status
              other_parties
              residence_since
              property_magnitude
              age
              other_payment_plans
              housing
              existing_credits
              job
              num_dependents
              own_telephone
              foreign_worker
              class
Test mode:10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         717               71.7    %
Incorrectly Classified Instances       283               28.3    %
Kappa statistic                          0.2513
Mean absolute error                      0.3781
Root mean squared error                  0.4472
Relative absolute error                 89.9974 %
Root relative squared error             97.5906 %
Total Number of Instances             1000

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.873     0.647      0.759     0.873     0.812      0.593    good
                 0.353     0.127      0.544     0.353     0.428      0.593    bad
```

```
Weighted Avg.    0.717     0.491      0.694     0.717     0.697      0.593

=== Confusion Matrix ===

   a   b   <-- classified as
 611  89 |   a = good
 194 106 |   b = bad
```

Regole:

1. (checking_status = "<0") **AND** (job = skilled) ⇒ class=bad (172.0/76.0)
2. (checking_status = "0<=X<20") **AND** (duration >= 24) **AND** (savings_status = "<100") ⇒ class=bad (61.0/19.0)
3. [*Empty Rule*] ⇒ class=good (767.0/162.0)

6.2 Risultati su `Hepatitis`

L'esecuzione ha coinvolto 139 istanze di training e 16 istanze di testing ad ogni iterazione del CV.

Esecuzione NaiveBayesSimple

```
=== Run information ===

Scheme:weka.classifiers.bayes.NaiveBayesSimple
Relation:     hepatitis
Instances:    155
Attributes:   20
              AGE
              SEX
              STEROID
              ANTIVIRALS
              FATIGUE
              MALAISE
              ANOREXIA
              LIVER_BIG
              LIVER_FIRM
              SPLEEN_PALPABLE
              SPIDERS
              ASCITES
              VARICES
              BILIRUBIN
              ALK_PHOSPHATE
              SGOT
              ALBUMIN
              PROTIME
              HISTOLOGY
              Class
Test mode:10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         131               84.5161 %
Incorrectly Classified Instances        24               15.4839 %
Kappa statistic                          0.5483
Mean absolute error                      0.1688
Root mean squared error                  0.3665
Relative absolute error                 51.118  %
Root relative squared error             90.5193 %
Total Number of Instances              155

=== Detailed Accuracy By Class ===
```

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.688     0.114      0.611     0.688     0.647      0.863    DIE
                 0.886     0.313      0.916     0.886     0.901      0.863    LIVE
Weighted Avg.    0.845     0.271      0.853     0.845     0.848      0.863

=== Confusion Matrix ===

   a   b   <-- classified as
  22  10 |   a = DIE
  14 109 |   b = LIVE
```

Esecuzione REPTree

```
=== Run information ===

Scheme:weka.classifiers.trees.REPTree -M 2 -V 0.001 -N 3 -S 1 -L -1
Relation:     hepatitis
Instances:    155
Attributes:   20
              AGE
              SEX
              STEROID
              ANTIVIRALS
              FATIGUE
              MALAISE
              ANOREXIA
              LIVER_BIG
              LIVER_FIRM
              SPLEEN_PALPABLE
              SPIDERS
              ASCITES
              VARICES
              BILIRUBIN
              ALK_PHOSPHATE
              SGOT
              ALBUMIN
              PROTIME
              HISTOLOGY
              Class
Test mode:10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         122               78.7097 %
```

```
Incorrectly Classified Instances        33               21.2903 %
Kappa statistic                          0.0554
Mean absolute error                      0.304
Root mean squared error                  0.4067
Relative absolute error                 92.0602 %
Root relative squared error            100.448  %
Total Number of Instances              155

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.063     0.024      0.4       0.063     0.108      0.533    DIE
                 0.976     0.938      0.8       0.976     0.879      0.533    LIVE
Weighted Avg.    0.787     0.749      0.717     0.787     0.72       0.533

=== Confusion Matrix ===

   a   b   <-- classified as
   2  30 |   a = DIE
   3 120 |   b = LIVE
```

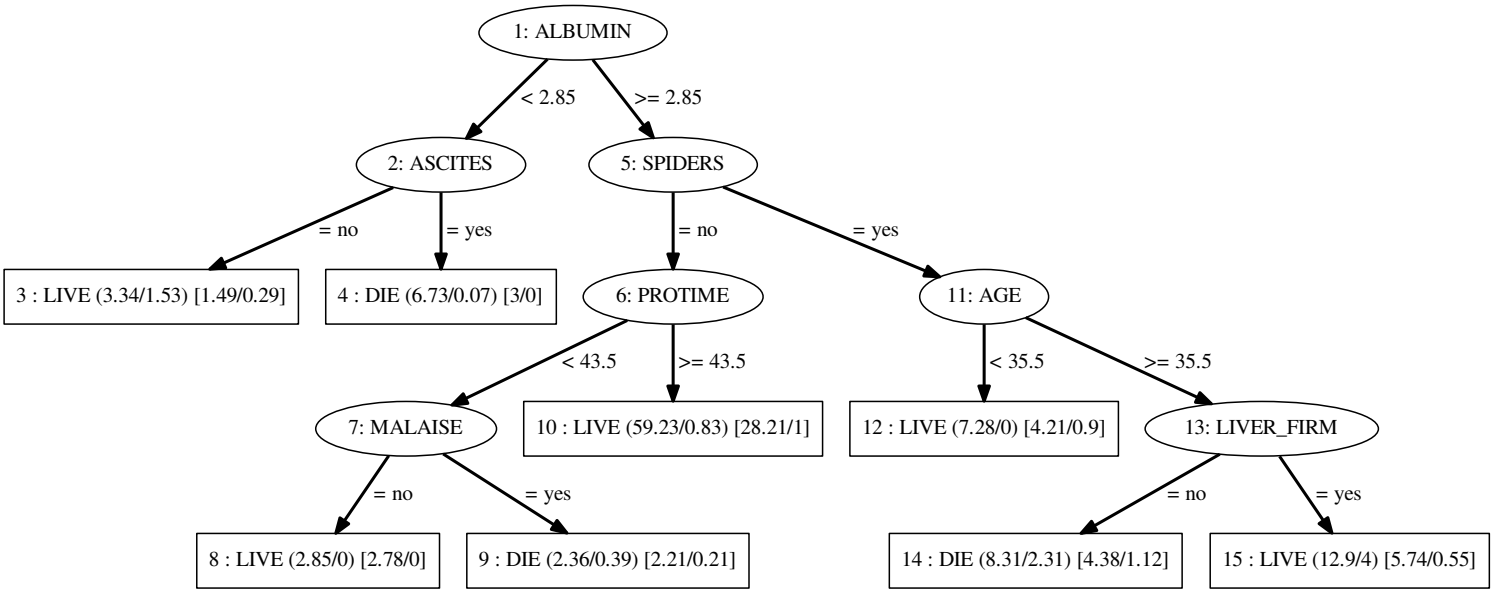


Figura 6.2: Modello di REPTree

Esecuzione JRip

```
=== Run information ===

Scheme:weka.classifiers.rules.JRip -F 3 -N 2.0 -O 2 -S 1
Relation:     hepatitis
Instances:    155
Attributes:   20
              AGE
              SEX
              STEROID
              ANTIVIRALS
              FATIGUE
              MALAISE
              ANOREXIA
              LIVER_BIG
              LIVER_FIRM
              SPLEEN_PALPABLE
              SPIDERS
              ASCITES
              VARICES
              BILIRUBIN
              ALK_PHOSPHATE
              SGOT
              ALBUMIN
              PROTIME
              HISTOLOGY
              Class
Test mode:10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         121               78.0645 %
Incorrectly Classified Instances        34               21.9355 %
Kappa statistic                          0.2623
Mean absolute error                      0.2594
Root mean squared error                  0.4122
Relative absolute error                 78.5662 %
Root relative squared error            101.7911 %
Total Number of Instances              155

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.344     0.106      0.458     0.344     0.393      0.664    DIE
                 0.894     0.656      0.84      0.894     0.866      0.664    LIVE
Weighted Avg.    0.781     0.543      0.761     0.781     0.768      0.664
```

```
=== Confusion Matrix ===

   a   b   <-- classified as
  11  21 |   a = DIE
  13 110 |   b = LIVE
```

Regole:

1. (ALBUMIN <= 3.8) **AND** (ALBUMIN <= 2.8) ⇒ Class=DIE (13.0/2.0)
2. (PROTIME <= 42) ⇒ Class=DIE (15.0/7.0)
3. (SPIDERS = yes) **AND** (BILIRUBIN >= 2) ⇒ Class=DIE (11.0/4.0)
4. [*Empty Rule*] ⇒ Class=LIVE (116.0/6.0)

6.3 Risultati su `Vehicle Silhouettes`

L'esecuzione ha coinvolto 761 istanze di training e 85 istanze di testing ad ogni iterazione del CV.

Esecuzione NaiveBayesSimple

```
=== Run information ===

Scheme:weka.classifiers.bayes.NaiveBayesSimple
Relation:     vehicle
Instances:    846
Attributes:   19
              COMPACTNESS
              CIRCULARITY
              DISTANCE CIRCULARITY
              RADIUS RATIO
              PR.AXIS ASPECT RATIO
              MAX.LENGTH ASPECT RATIO
              SCATTER RATIO
              ELONGATEDNESS
              PR.AXIS RECTANGULARITY
              MAX.LENGTH RECTANGULARITY
              SCALED VARIANCE_MAJOR
              SCALED VARIANCE_MINOR
              SCALED RADIUS OF GYRATION
              SKEWNESS ABOUT_MAJOR
              SKEWNESS ABOUT_MINOR
              KURTOSIS ABOUT_MAJOR
              KURTOSIS ABOUT_MINOR
              HOLLOWS RATIO
              Class
Test mode:10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         383               45.2719 %
Incorrectly Classified Instances       463               54.7281 %
Kappa statistic                          0.2759
Mean absolute error                      0.2793
Root mean squared error                  0.4581
Relative absolute error                 74.4992 %
Root relative squared error            105.8072 %
Total Number of Instances              846

=== Detailed Accuracy By Class ===
```

```
                 TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                   0.415     0.167      0.454     0.415     0.433      0.706    opel
                   0.387     0.122      0.522     0.387     0.444      0.714    saab
                   0.17      0.024      0.712     0.17      0.274      0.851    bus
                   0.874     0.41       0.396     0.874     0.545      0.827    van
Weighted Avg.      0.453     0.176      0.524     0.453     0.422      0.774

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
  88  60   0  64 |   a = opel
  63  84   3  67 |   b = saab
  38   9  37 134 |   c = bus
   5   8  12 174 |   d = van
```

Esecuzione REPTree

```
=== Run information ===

Scheme:       weka.classifiers.trees.REPTree -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0
Relation:     vehicle
Instances:    846
Attributes:   19
              COMPACTNESS
              CIRCULARITY
              DISTANCE CIRCULARITY
              RADIUS RATIO
              PR.AXIS ASPECT RATIO
              MAX.LENGTH ASPECT RATIO
              SCATTER RATIO
              ELONGATEDNESS
              PR.AXIS RECTANGULARITY
              MAX.LENGTH RECTANGULARITY
              SCALED VARIANCE_MAJOR
              SCALED VARIANCE_MINOR
              SCALED RADIUS OF GYRATION
              SKEWNESS ABOUT_MAJOR
              SKEWNESS ABOUT_MINOR
              KURTOSIS ABOUT_MAJOR
              KURTOSIS ABOUT_MINOR
              HOLLOWS RATIO
              Class
Test mode:    10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         612               72.3404 %
Incorrectly Classified Instances       234               27.6596 %
Kappa statistic                          0.6313
Mean absolute error                      0.1617
Root mean squared error                  0.3109
Relative absolute error                 43.1254 %
Root relative squared error             71.8227 %
```

```
Total Number of Instances              846

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,575    0,137    0,584      0,575    0,580      0,440    0,842     0,586     opel
                 0,475    0,129    0,560      0,475    0,514      0,366    0,795     0,561     saab
                 0,945    0,040    0,892      0,945    0,918      0,889    0,968     0,895     bus
                 0,910    0,063    0,815      0,910    0,860      0,816    0,972     0,879     van
Weighted Avg.    0,723    0,093    0,711      0,723    0,716      0,625    0,893     0,728

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 122  68   8  14 |   a = opel
  81 103  13  20 |   b = saab
   1   4 206   7 |   c = bus
   5   9   4 181 |   d = van
```

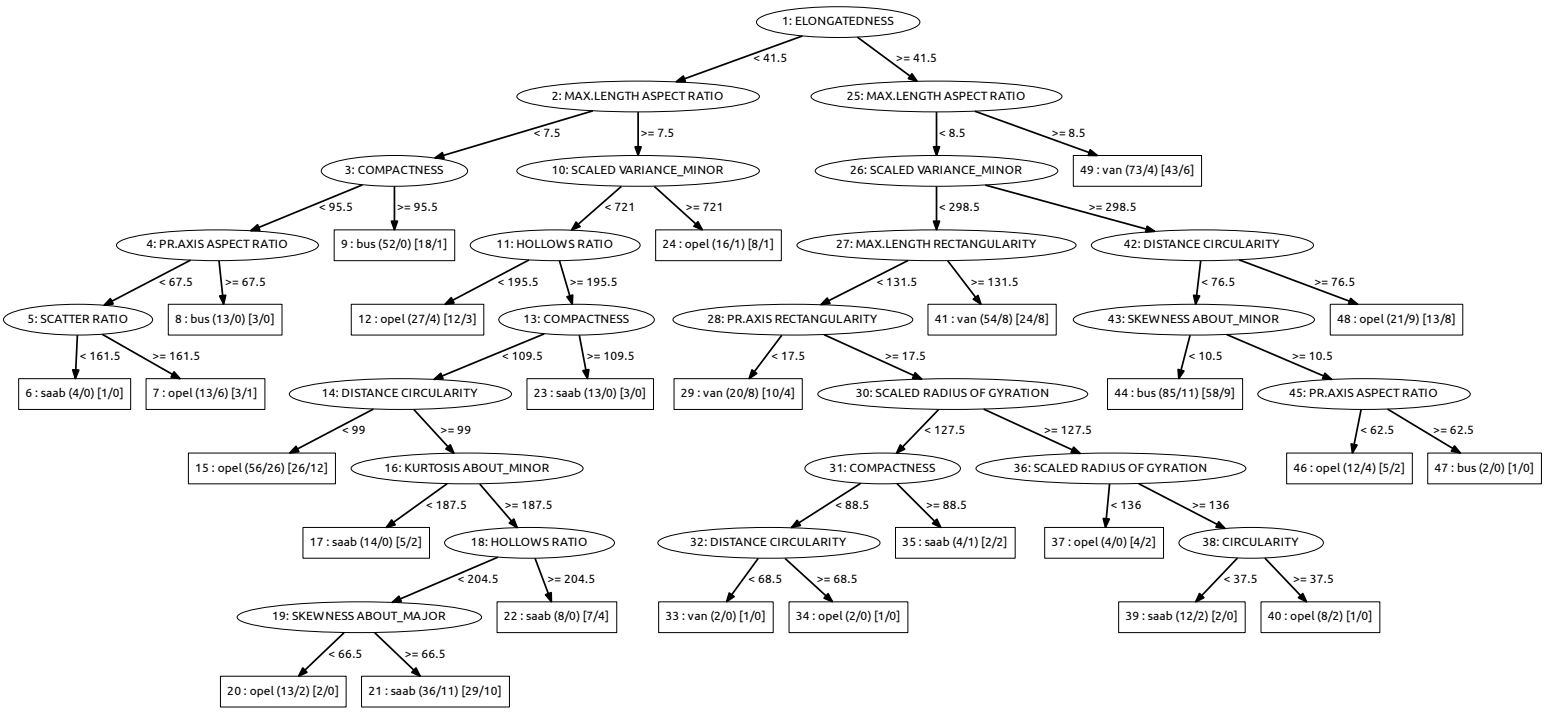


Figura 6.3: Modello di REPTree

Esecuzione JRip

```
=== Run information ===

Scheme:       weka.classifiers.rules.JRip -F 3 -N 2.0 -O 2 -S 1
Relation:     vehicle
Instances:    846
Attributes:   19
              COMPACTNESS
              CIRCULARITY
              DISTANCE CIRCULARITY
              RADIUS RATIO
              PR.AXIS ASPECT RATIO
              MAX.LENGTH ASPECT RATIO
              SCATTER RATIO
              ELONGATEDNESS
              PR.AXIS RECTANGULARITY
              MAX.LENGTH RECTANGULARITY
              SCALED VARIANCE_MAJOR
              SCALED VARIANCE_MINOR
              SCALED RADIUS OF GYRATION
              SKEWNESS ABOUT_MAJOR
              SKEWNESS ABOUT_MINOR
              KURTOSIS ABOUT_MAJOR
              KURTOSIS ABOUT_MINOR
              HOLLOWS RATIO
              Class
Test mode:    10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         584               69.0307 %
Incorrectly Classified Instances       262               30.9693 %
Kappa statistic                          0.5868
Mean absolute error                      0.1914
Root mean squared error                  0.3323
Relative absolute error                 51.0598 %
Root relative squared error             76.7524 %
Total Number of Instances              846

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,481    0,126    0,560      0,481    0,518      0,374    0,785     0,545     opel
                 0,484    0,146    0,533      0,484    0,507      0,349    0,793     0,501     saab
                 0,940    0,099    0,768      0,940    0,845      0,792    0,940     0,812     bus
                 0,864    0,043    0,860      0,864    0,862      0,820    0,927     0,861     van
Weighted Avg.    0,690    0,105    0,677      0,690    0,680      0,580    0,860     0,677

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 102  71  27  12 |   a = opel
  74 105  27  11 |   b = saab
   2   6 205   5 |   c = bus
   4  15   8 172 |   d = van
```

Regole:

1. (ELONGATEDNESS >= 43) **AND** (MAX.LENGTH ASPECT RATIO >= 9) **AND** (DISTANCE CIRCULARITY >= 73) ⇒ Class=van (86.0/0.0)

2. (SCALED VARIANCE_MINOR <= 309) **AND** (MAX.LENGTH RECTANGULARITY >= 132) **AND** (DISTANCE CIRCULARITY <= 64) **AND** (SCALED RADIUS OF GYRATION <= 157) ⇒ Class=van (23.0/0.0)

3. (PR.AXIS RECTANGULARITY <= 18) **AND** (MAX.LENGTH RECTANGULARITY >= 128) **AND** (SCALED RADIUS OF GYRATION <= 140) ⇒ Class=van (42.0/6.0)

4. (SCALED VARIANCE_MINOR <= 309) **AND** (MAX.LENGTH RECTANGULARITY >= 142) ⇒ Class=van (33.0/5.0)

5. (ELONGATEDNESS >= 53) **AND** (SCALED RADIUS OF GYRATION >= 137) ⇒ Class=van (15.0/5.0)

6. (SCALED VARIANCE_MAJOR <= 177) **AND** (MAX.LENGTH ASPECT RATIO >= 10) ⇒ Class=van (8.0/1.0)

7. (MAX.LENGTH ASPECT RATIO >= 8) **AND** (MAX.LENGTH RECTANGULARITY >= 173) ⇒ Class=opel (45.0/8.0)

8. (MAX.LENGTH ASPECT RATIO >= 8) **AND** (COMPACTNESS <= 103) **AND** (ELONGATEDNESS <= 37) **AND** (HOLLOWS RATIO <= 195) ⇒ Class=opel (14.0/0.0)

9. (MAX.LENGTH ASPECT RATIO >= 8) **AND** (HOLLOWS RATIO <= 198) **AND** (KURTOSIS ABOUT_MINOR >= 189) ⇒ Class=opel (42.0/17.0)

10. (SKEWNESS ABOUT_MAJOR <= 67) **AND** (HOLLOWS RATIO <= 203) ⇒ Class=opel (66.0/30.0)

11. (SCALED RADIUS OF GYRATION <= 142) **AND** (HOLLOWS RATIO <= 194) **AND** (DISTANCE CIRCULARITY >= 57) ⇒ Class=opel (17.0/2.0)

12. (MAX.LENGTH ASPECT RATIO >= 9) **AND** (DISTANCE CIRCULARITY >= 100) **AND** (SCALED VARIANCE_MAJOR <= 231) ⇒ Class=saab (71.0/9.0)

13. (MAX.LENGTH ASPECT RATIO >= 9) **AND** (PR.AXIS ASPECT RATIO <= 61) ⇒ Class=saab (23.0/7.0)

14. (SCALED VARIANCE_MAJOR <= 165) **AND** (DISTANCE CIRCULARITY <= 66) ⇒ Class=saab (36.0/11.0)

15. (SKEWNESS ABOUT_MAJOR <= 72) **AND** (PR.AXIS ASPECT RATIO <= 65) **AND** (DISTANCE CIRCULARITY >= 81) **AND** (SKEWNESS ABOUT_MAJOR >= 66) ⇒ Class=saab (27.0/7.0)

16. (CIRCULARITY <= 40) **AND** (RADIUS RATIO <= 144) ⇒ Class=saab (16.0/6.0)

17. [*Empty Rule*] ⇒ Class=bus (282.0/69.0)

6.4 Risultati su `Wisconsin Breast Cancer`

L'esecuzione ha coinvolto 629 istanze di training e 70 istanze di testing per ogni fold.

Esecuzione NaiveBayesSimple

```
=== Run information ===

Scheme:weka.classifiers.bayes.NaiveBayesSimple
Relation:     wisconsin-breast-cancer
Instances:    699
Attributes:   10
              Clump_Thickness
              Cell_Size_Uniformity
              Cell_Shape_Uniformity
              Marginal_Adhesion
              Single_Epi_Cell_Size
              Bare_Nuclei
              Bland_Chromatin
              Normal_Nucleoli
              Mitoses
              Class
Test mode:10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         671               95.9943 %
Incorrectly Classified Instances        28                4.0057 %
Kappa statistic                          0.9127
Mean absolute error                      0.0405
Root mean squared error                  0.1987
Relative absolute error                  8.9562 %
Root relative squared error             41.8102 %
Total Number of Instances              699

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.952     0.025      0.986     0.952     0.969      0.99     benign
                 0.975     0.048      0.914     0.975     0.944      0.984    malignant
Weighted Avg.    0.96      0.033      0.962     0.96      0.96       0.988

=== Confusion Matrix ===

   a   b   <-- classified as
 436  22 |   a = benign
```

```
   6 235 |   b = malignant
```

Esecuzione REPTree

```
=== Run information ===

Scheme:weka.classifiers.trees.REPTree -M 2 -V 0.001 -N 3 -S 1 -L -1
Relation:     wisconsin-breast-cancer
Instances:    699
Attributes:   10
              Clump_Thickness
              Cell_Size_Uniformity
              Cell_Shape_Uniformity
              Marginal_Adhesion
              Single_Epi_Cell_Size
              Bare_Nuclei
              Bland_Chromatin
              Normal_Nucleoli
              Mitoses
              Class
Test mode:10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         656               93.8484 %
Incorrectly Classified Instances        43                6.1516 %
Kappa statistic                          0.8653
Mean absolute error                      0.083
Root mean squared error                  0.2234
Relative absolute error                 18.3662 %
Root relative squared error             47.0064 %
Total Number of Instances              699

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.941     0.066      0.964     0.941     0.952      0.964    benign
                 0.934     0.059      0.893     0.934     0.913      0.964    malignant
Weighted Avg.    0.938     0.064      0.94      0.938     0.939      0.964

=== Confusion Matrix ===

   a   b   <-- classified as
 431  27 |   a = benign
  16 225 |   b = malignant
```

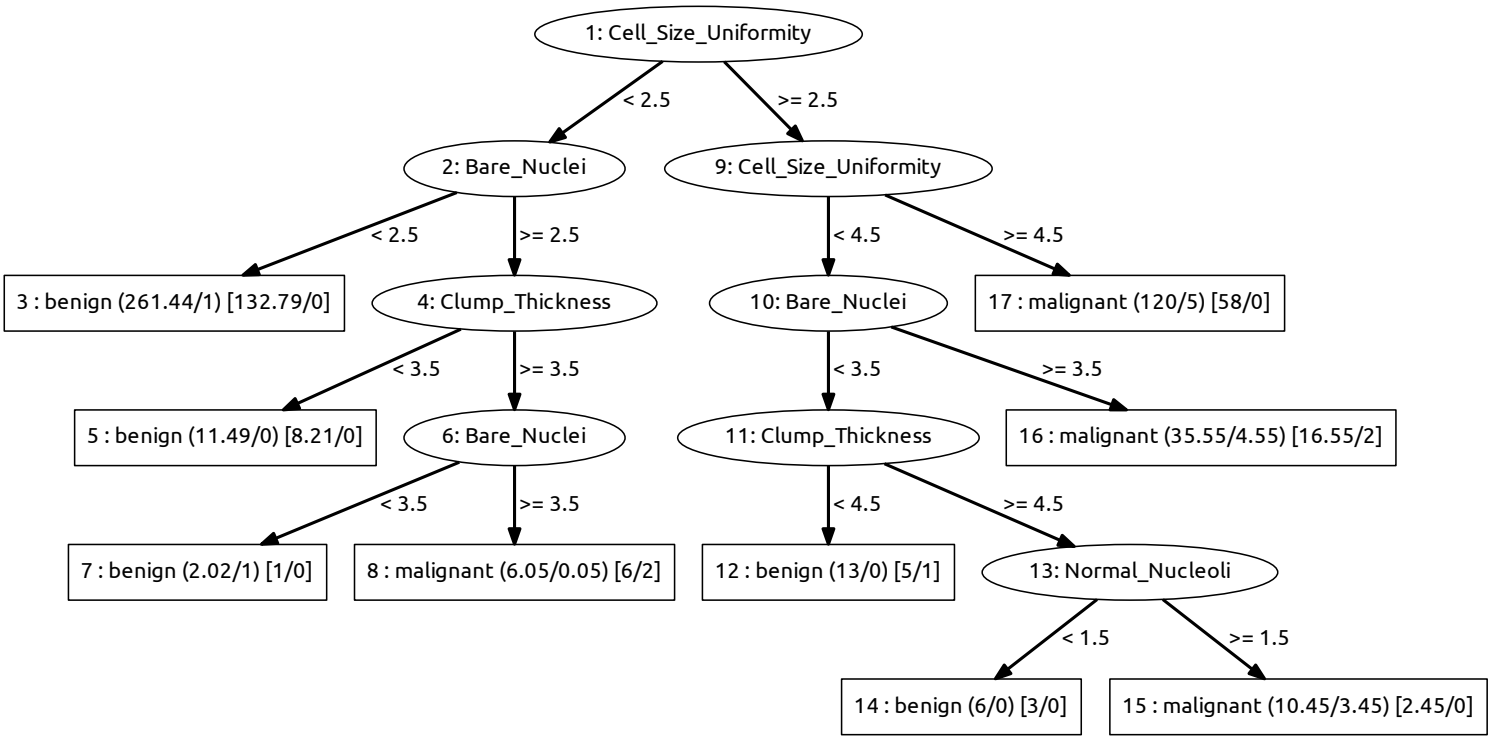


Figura 6.4: Modello di REPTree

Esecuzione JRip

```
=== Run information ===

Scheme:weka.classifiers.rules.JRip -F 3 -N 2.0 -O 2 -S 1
Relation:     wisconsin-breast-cancer
Instances:    699
Attributes:   10
              Clump_Thickness
              Cell_Size_Uniformity
              Cell_Shape_Uniformity
              Marginal_Adhesion
              Single_Epi_Cell_Size
              Bare_Nuclei
```

```
              Bland_Chromatin
              Normal_Nucleoli
              Mitoses
              Class
Test mode:10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         667               95.422  %
Incorrectly Classified Instances        32                4.578  %
Kappa statistic                          0.8999
Mean absolute error                      0.0618
Root mean squared error                  0.2022
Relative absolute error                 13.676  %
Root relative squared error             42.5462 %
Total Number of Instances              699

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.952     0.041      0.978     0.952     0.965      0.973    benign
                 0.959     0.048      0.913     0.959     0.935      0.973    malignant
Weighted Avg.    0.954     0.044      0.955     0.954     0.954      0.973

=== Confusion Matrix ===

   a   b   <-- classified as
 436  22 |   a = benign
  10 231 |   b = malignant
```

Regole:

1. (Cell_Size_Uniformity >= 3) **AND** (Cell_Size_Uniformity >= 5) ⇒ Class=malignant (178.0/5.0)
2. (Bare_Nuclei >= 4) **AND** (Bare_Nuclei >= 7) ⇒ Class=malignant (48.0/4.0)
3. (Normal_Nucleoli >= 3) **AND** (Clump_Thickness >= 6) ⇒ Class=malignant (13.0/1.0)
4. (Bare_Nuclei >= 3) **AND** (Clump_Thickness >= 5) ⇒ Class=malignant (11.0/3.0)
5. (Marginal_Adhesion >= 8) ⇒ Class=malignant (2.0/0.0)
6. [*Empty Rule*] ⇒ Class=benign (447.0/2.0)

Capitolo 7: Analisi

In questo capitolo verranno analizzati i risultati dell'esperimento, ottenuti utilizzando lo strumento *Experimenter* di Weka. La configurazione ha riguardato un task di classificazione con una 10-fold cross validation su 4 dataset con 10 iterazioni. Gli algoritmi in gioco sono REPTree e RIPPER. I risultati così ottenuti sono 800.

7.1 Test

Il confronto è stato fatto su tre misure, la percentuale di predizioni corrette, la *F-Measure*, che è la media armonica di precisione e richiamo. Per fini di completezza riportiamo le relative formule:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negastives}$$

$$F\text{-}measure = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Altra misura utilizzata è il tempo necessario per costruire il modello. Il test utilizzato è un *Paired T-test*.

Di seguito vengono mostrate le tabelle che raccolgono i risultati di esecuzione del test per le misure, più un'altra che presenta una classifica degli algoritmi.

Come *baseline* è stato scelto RIPPER. Accanto ai risultati di REPTree comparirà un pallino bianco (○) in caso di miglioramento o un pallino nero (●) in caso di peggioramento. Se i risultati non dovessero differire in maniera statisticamente significativa, non comparirà nulla.

Dataset	(1)	(2)
wisconsin-breast-cancer	95.61	94.77
german-credit	72.21	72.02
segment	95.24	95.26
vehicle	68.31	70.18

○, ● miglioramento o peggioramento statisticamente significativo

Tabella 7.1: Confronto sulla percentuale di predizioni corrette

Dataset	(1)	(2)
wisconsin-breast-cancer	0.97	0.96
german-credit	0.81	0.81
segment	0.97	0.97
vehicle	0.47	0.50

◦, • miglioramento o peggioramento statisticamente significativo

Tabella 7.2: Confronto su F-measure

Dataset	(1)	(2)
wisconsin-breast-cancer	0.02	0.00 •
german-credit	0.08	0.01 •
segment	0.66	0.05 •
vehicle	0.18	0.02 •

◦, • miglioramento o peggioramento statisticamente significativo

Tabella 7.3: Confronto sul tempo di addestramento

Resultset	Wins− Losses	Wins	Losses
(2)	0	0	0
(1)	0	0	0

Tabella 7.4: Ranking sulla percentuale di predizioni corrette

Resultset	Wins− Losses	Wins	Losses
(2)	0	0	0
(1)	0	0	0

Tabella 7.5: Ranking su F-measure

Resultset	Wins− Losses	Wins	Losses
(1)	4	4	0
(2)	-4	0	4

Tabella 7.6: Ranking sul tempo di addestramento

Legenda:
(1) RIPPER
(2) REPTree

7.2 Interpretazione dei risultati

Come si evince dai risultati, i due algoritmi si eguagliano per quanto riguarda la percentuale di predizioni corrette e F-measure, in quanto nessuno si comporta statisticamente meglio o peggio rispetto all'altro, per la mancanza di pallini bianchi (◦) e pallini neri (•) accanto ai risultati. Tutt'altra storia riguarda il tempo di costruzione del modello: REPTree si comporta peggio di RIPPER su tutti i dataset.

Il *ranking test* classifica gli algoritmi in base al totale di vittorie e sconfitte significative rispetto agli altri. La prima colonna è la differenza tra le vittorie e le sconfitte. Questa differenza è usata per generare il ranking. Con zero vittorie e zero sconfitte, gli algoritmi raggiungono un pareggio per percentuale di predizioni corrette e F-measure. Per quanto riguarda il tempo di addestramento, c'è una debacle evidente di REPTree rispetto a RIPPER su tutti i dataset.

Capitolo 8: Conclusioni

In questo documento sono stati confrontati due algoritmi, RIPPER e REPTree, su quattro dataset provienti dall'UCI Machine Learning Repository.

I risultati hanno evidenziato un sostanziale pareggio tra le due tecniche in quanto ad efficacia di predizione, ma è chiaro il vantaggio di utilizzare RIPPER grazie al minore tempo di addestramento rispetto a REPTree.

In futuro si potrebbero eseguire ulteriori test su misure diverse per vagliare altri aspetti degli algoritmi.

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] P. Domingos and M. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Mach. Learn.*, 29(2-3):103–130, Nov. 1997. ISSN 0885-6125.

[5] 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.

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

[7] T. M. Mitchell. *Machine Learning*, chapter 6. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072.

[8] G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Mach. Learn.*, 5(1):71–99, May 1990. ISSN 0885-6125.

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

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

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

[12] 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.

[13] 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.

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.