Sviluppo di predittori ad albero, calcolo dell'errore e confronto con WEKA

Francesco Scarlata

Abstract—Questa relazione parlerá dell'implementazione utilizzata per sviluppare dei predittori ad albero. Verrá mostrato l'algoritmo di generazione dell'albero e gli algoritmi per stimare il rischio su questo tipo di algoritmi di apprendimento. Verranno poi commentati i risultati ottenuti confrontando i tempi degli algoritmi e il numero di nodi dei vari algoritmi con il classificatore ad albero implementato da Weka.

1 Richiami teorici

1.1 Famiglie di algoritmi per la generazione del predittore ad albero

Le principali famiglie di algoritmi per la generazione dell'albero di decisione, sono le seguenti:

- *ID3* (Iterative Dichotomiser 3)
- *C4.5* (successore di ID3, ideato dalla stessa persona che ha creato ID3)
- *C5.0/See5* (successore di c4.5, ideato dalla stessa persona ma commercializzato)
- *CART* (Classification And Regression Trees)

Poiché C5.0 ha solo miglioramenti in prestazioni, parleremo solo degli algoritmi C4.5 (usato da WEKA nel classificatore J48) e CART (usato in questo progetto).

C4.5 costruisce l'albero usando il concetto di entropia dell'informazione, che é il tasso medio a cui l'informazione é prodotta da una sorgente statistica di dati.

Descriviamo il procedimento di questo algoritmo. Per ogni nodo dell'albero, C4.5 sceglie l'attributo del dato che divide il piú efficacemente il suo set di esempi in sottoinsiemi. Il criterio di splitting é l'information gain normalizzato, ovvero la differenza in entropia. L'attributo con il piú alto information gain normalizzato viene scelto per fare la decisione (e quindi dividere il set di quel nodo in due subset in base a quell'attributo. L'algoritmo C4.5 quindi ricorre sulle sottoliste piú piccole.

In questo algoritmo ci sono tre casi base:

- Tutti gli esempi nel nodo appartengono alla stessa classe. Quando questo succede, si crea un nodo foglia nell'albero di decisione dicendo di scegliere quella classe.
- Nessuna delle feature fornisce alcun information gain. Quando questo succede, si crea un nodo di decisione sull'albero usando il valore atteso della classe.

Dati questi casi base, l'algoritmo segue i seguenti passi:

1) Controlla per i casi base.
2) Per ogni attributo a, trova l'information gain normalizzato che divide su a.
3) Sia a_{best} l'attributo con il piú alto information gain normalizzato.
4) Crea un nodo decisionale (quindi un nodo interno) che divide su a_{best}.
5) Ricorri sulle sottoliste ottenute dividendo su a_{best}, e aggiungi questi nodi come figli del nodo.

CART invece é una tecnica non paramentrica di apprendimento di un albero decisionale che puó produrre sia alberi di classificazione che alberi di regressione, dipendentemente dal fatto che l'etichetta sia categorica o numerica, rispettivamente. In questo caso, gli alberi decisionali sono formati da una collezione di regole basate sulle variabili del modello del dataset:

- Le regole sono basate sui valori delle variabili e sono selezionate per avere il miglior split per differenziare gli esempi basati sulla variabile dipendente.
- Una volta che una regola é selezionata, si divide in due il nodo e lo stesso processo é applicato a ciascun nodo "figlio".
- Lo splitting si arresta quando l'algoritmo rileva che non puó essere fatto ulteriore guadagno o delle regole di stop pre-impostate sono state soddisfatte (alternativamente, puó essere fatto il pruning dopo aver finito la generazione completa dell'albero)

Anche qui ogni ramo finisce in un nodo terminale (la foglia). Ciascun esempio cade in esattemente un solo nodo terminale, e ogni nodo terminale é unicamente definito da un insieme di regole.

I due metodi sono molto simili, ma la differenza piú importante é su come i due metodi definisco l'attributo "migliore".

L'algoritmo C4.5 utilizza, come detto prima, l'information gain che si basa sul concetto

F. Scarlata, Corso di Metodi statistici per l'apprendimento, A/A 2017-2018, Universitá di Milano, via Comelico 39/41, Milano, Italy
E-mail: francesco.scarlata@studenti.unimi.it

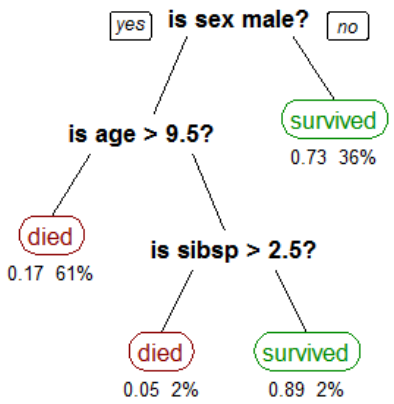

Fig. 1: Esempio di albero di decicisione usando CART

di entropia. L'entropia é definita come segue: $H(T) = I_E(p_1, p_2, ..., p_J) = -\sum_{i=1}^{J} p_i \log_2 p_i$ dove i p_i rappresenta la percentuale che la classe i sia presente nel nodo figlio che risulta dalla divisione dell'albero, con $\sum_{i=1}^{J} p_i = 1$. L'information gain é quindi definito come $IG(T, a) = H(T) - H(T|a)$ dove $H(T)$ é l'entropia del nodo padre e $H(T|a)$ é la somma pesata dell'entropia dei figli divisi tramite l'attributo a.

Nel caso dell'algoritmo CART, viene usato il concetto di "impuritá di Gini" (o Gini Impurity) come criterio di scelta. L'impuritá di Gini é una misura di quanto spesso un elemento scelto randomicamente dal set, sarebbe etichettato incorrettamente se fosse etichettato randomicamente in accordo alla distribuzione delle etichette del sottoinsieme. L'impuritá di Gini puó essere quindi calcolata come la somma delle probabilitá p_i di un oggetto con etichetta i che viene scelta per la probabilitá $\sum_{k \neq i} p_k = 1 - p_i$ dell'errore nel categorizzare quell'esempio. Il valore minimo di questa misura é lo zero, in cui tutti i casi nel nodo ricadono in una singola categoria. Quindi per calcolare l'impuritá per un set di oggetti con J classi, si supponga $i \in \{1, 2, ..., J\}$, e sia p_i la frazione degli oggetti etichettati con la classe i nell'insieme, cioé:

$$I_G(p) = \sum_{i=1}^{J} p_i \sum_{k \neq i} p_k = \sum_{i}^{J} p_i(1 - p_i) =$$

$$= \sum_{i}^{J} (p_i - p_i^2) = \sum_{i}^{J} p_i - \sum_{i}^{J} p_i^2 = 1 - \sum_{i}^{J} p_i^2$$

In questo caso, l'information gain viene calcolato come $IG(T, a) = I_G(T) - I_G(T|a)$ dove $I_G(T)$ é l'impuritá di Gini nel nodo e $I_G(T|a)$ é la somma pesata delle impuritá di Gini calcolate sui due figli che verrebbero creati dividendo il nodo per l'attributo a.

2 Algoritmi usati

In questo progetto é stata implementata una variante dell'algoritmo CART iterativo che utilizza il numero di nodi corrente come condizione per stoppare l'algoritmo al numero di nodi dato in input senza fare successivo pruning. Inoltre, non essendo piú ricorsivo, la scelta del nodo da splittare viene scelto con il criterio del maggior information gain tra i nodi, cercando cosí sempre prima il nodo che riduce di piú l'errore (piú informazioni a riguardo verranno date nella sezione 3.2).

Come algoritmi per la scelta del parametro ottimale, sono stati utilizzati i seguenti algoritmi:

- Cross validazione interna
- Approccio che segue dall'analisi di rischio per i classificatori ad albero

Questi vengono anche utilizzati per poi stimare il rischio del classificatore trovato.

2.1 Cross validazione interna

L'algoritmo di cross-validazione interna funziona nel seguente modo:

- Divide il training set in N blocchi uguali (creando quindi N sottoinsiemi di dati)
- Per ogni valore del parametro i, calcola la cross-validazione esterna dell'algoritmo con parametro i. Questo significa che per ogni valore i bisogna:
  - Creare N classificatori dividendo il training set S in N blocchi e usare $N - 1$ blocchi come training set (il blocco restante viene detto "validation set")
  - Per ogni classificatore, calcolare il validation error come:

    $$\tilde{er}(h^{(k)}) = \frac{N}{m} \sum_{(x,y) \in D_k} l(y, h^{(k)}(x))$$

    dove $\tilde{er}$ é il validation error, D_k é il blocco di indice k non usato nel training set, l é la funzione di perdita, e $h^{(k)}$ é il classificatore trovato usando come training set l'insieme $S \setminus D_k$
  - Fare la media dei validation error sugli N classificatori calcolati con parametro i, cioé

    $$er_i^{CV} = \frac{1}{N} \sum_{k=1}^{N} \tilde{er}(h^{(k)})$$

    Questa é la stima dell'accuratezza dell'algoritmo con parametro i mediante cross-validazione di grado N, ed é chiamato errore di cross-validazione dell'algoritmo con parametro i fissato.
- trovare il parametro i^* tale che sia il paramento con il piú basso errore di Cross-validazione, cioé

  $$i^* = \arg\min_i er_i^{CV}$$

  Quindi, a fine esecuzione, avremo il parametro ottimale i^* e la stima di accuratezza dell'algoritmo mediante cross-validazione.

```
Is safety == low?
--> True:
  Predicts unacc
--> False:
  Is persons == 2?
  --> True:
    Predicts unacc
  --> False:
    Is maint == vhigh?
    --> True:
      Is buying == high?
      --> True:
        Predicts unacc
      --> False:
        Is buying == vhigh?
        --> True:
          Predicts unacc
        --> False:
          Predicts acc
    --> False:
      Predicts acc
```

Fig. 2: Esempio di albero di decisione creato usando l'approccio derivato dal rischio

2.2 Scelta deritata dallo studio del rischio nei predittori ad albero

Si lascia al lettore l'analisi del rischio statistico per predittori ad albero in cui si usano i pesi dei predittori $w(h)$ soddisfacenti la proprietá

$$\sum_{h \in H} w(h) \leqslant 1$$

e come si arriva al risultato seguente con probabilitá $1 - \delta$ rispetto all'estrazione del training set:

$$er(h) \leqslant \min_{h \in H} \widehat{er}(h) + \sqrt{\frac{1}{2m}\left(\ln \frac{1}{w(h)} + \ln \frac{2}{\delta}\right)}$$

in cui m é la taglia del training set.

Questo ci suggerisce di prendere il parametro $\widehat{h}$ come:

$$\widehat{h} = \arg\min_{h \in H} \left(\widehat{er}(h) + \sqrt{\frac{1}{2m}\left(\ln \frac{1}{w(h)} + \ln \frac{2}{\delta}\right)}\right)$$

3 Simulazione e esperimenti

3.1 Dataset

Sono stati utilizzati due dataset per testare questo progetto, entrambi presi dal sito " UCI Machine learning":

- Car Evaluation data set
- Nursery data set

Nell'indice appendice (sezione 6) si possono trovare i link del sito di UCI machine learning, e le pagine dei due dataset.

Di seguito si mostrano un paio di informazioni sui due dataset.

Il dataset "Car Evaluation" ha 6 attributi, é creato per problemi di classificazione e il numero di esempi é 1728. Il dataset "Nursery" é creato anch'esso per problemi di classificazione ma ha 8 attributi e il numero di esempi é 12960.

Questi dataset vengono divisi, prima di passare il set agli algoritmi, in due set: il training set e test set, con percentuale 75% e 25% rispettivamente.

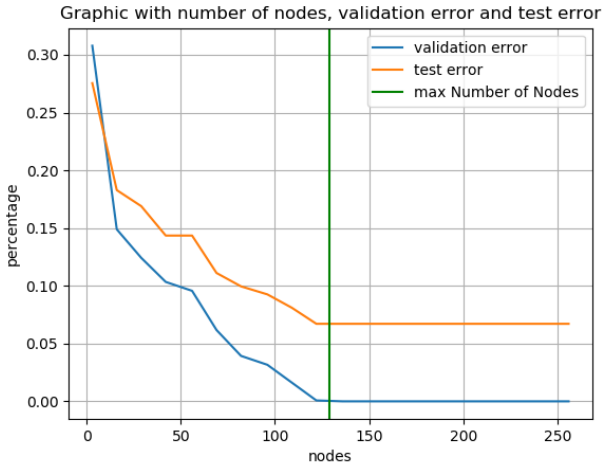

Fig. 3: Grafico che mostra l'errore di cross-validazione e il test error sul dataset "Car Evaluation"

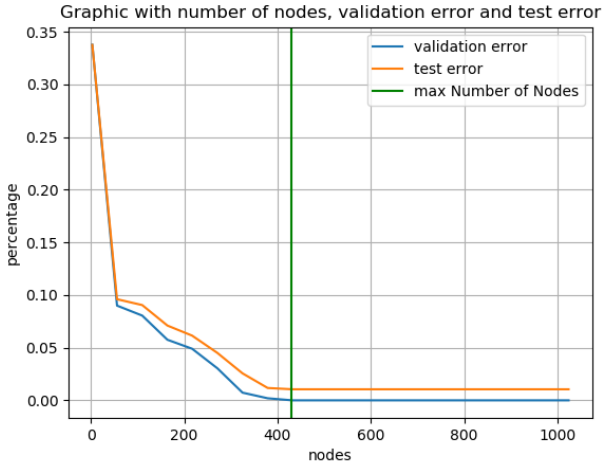

Fig. 4: Grafico che mostra l'errore di cross-validazione e il test error sul dataset "Nursery"

Inoltre un particolare da fare notare é che questi due dataset sono poco rumorosi, ma non completamente privi di rumore.

Questo é possibile notarlo mostrando un grafico di come il cross-validation error e il test error si comportano dato un certo intervallo che va dall'underfitting all'overfitting. In figura, i grafici utilizzano come intervallo i nodi 3 e 2^{d+2}.

3.2 Dettagli implementativi

3.2.1 Implementazione della variante di CART

Come era stato detto nel paragrafo 1.1, é stata utilizzata una variante dell'algoritmo CART. Al posto di utilizzare un algoritmo ricorsivo su cui poi fare pruning, si é preferito un approccio iterativo in cui quando un certo numero di nodi é stato creato, interrompe l'esecuzione. In breve, questa variante iterativa controlla ad ogni iterazione se il numero di nodi non é stato superato e, se non lo é, cerca il nodo con il piú alto information gain tra tutti i nodi che possono essere

divisi. Abbiamo quindi tre diversi tipi di nodi all'interno di questo algoritmo prima della fine dell'esecuzione: I nodi foglia che sono formati da esempi della stessa categoria o comunque da esempi che danno un information gain pari a zero (se gli esempi hanno etichette diverse in questo nodo, allora ci sono delle etichette anomale); dei nodi intermedi, che salvano il test con l'information gain migliore, e i nodi decisionali (o nodi interni), in cui la lista di esempi era stata spezzata in due sotto nodi in base al test trovato: gli esempi che si classificano veri rispetto al test vanno sul figlio "vero", altrimenti sul figlio "falso". In questo modo abbiamo quindi un albero binario. Si vuole fare notare che poiché abbiamo un algoritmo che divide un nodo sempre in 2 altri nodi, avremo che il numero di nodi dell'albero sará sempre dispari. Poiché un nodo in piú o in meno é stato considerato poco rilevante, si é preferito concludere con il numero minimo di nodi che supera un numero pari ricevuto in input (per esempio, se si crea un albero decisionale con 4 nodi, ne verranno utilizzati 5 invece che 3). Ogni volta che un nodo viene creato, viene controllato il suo information gain. Se questo valore é uguale a zero, significa che quel nodo non ha piú altro da separare e viene creato un nodo foglia, altrimenti crea un nodo intermedio.

Il speudo codice per un passo di iterazione é il seguente:

1) Scegli il nodo intermedio con il massimo information gain (se la lista é vuota termina l'esecuzione)
2) Usa il test per dividere in due sotto nodi il nodo. Elimina il nodo intermedio dalla lista dei nodi intermedi.
3) Crea un nodo decisionale dal nodo intermedio con i due sottonodi come figli usando l'attributo salvato precedentemente.
4) Se il sottonodo ha information gain uguale a 0, allora crea una foglia. Altrimenti crea un nodo intermedio e inseriscilo nella lista dei nodi intermedi.
5) Se il numero di nodi corrente é maggiore di quello inserito, termina le iterazioni.

Il ciclo delle iterazioni quindi puó finire per due motivi: i nodi sono piú del numero inserito, oppure non ci sono piú elementi nella lista di nodi intermedi. Nel primo caso significa che ci sono dei nodi intermedi non ancora diventati foglie, quindi questi nodi vengono trasformati in nodi foglia. Nel secondo caso invece significa che l'albero ha meno nodi di quelli richiesti ed inserito in entrata. Un'altra cosa che si vuole fare notare é che in questa maniera, vengono sempre esplorate e divise le strade che hanno piú information gain, quindi il predittore con $i + 2$ nodi avrá solo due nodi in piú come differenza rispetto al predittore con i nodi usando lo stesso training set. Non é una differenza sfruttata in altri algoritmi, ma si voleva fare notare questo elemento.

3.2.2 Minimizzazione dell'errore e scelta dell'insieme H da confrontare

Per la scelta del parametro abbiamo detto nel paragrafo 2 che vengono utilizzati la cross-validazione e un approccio che deriva dall'analisi del rischio, ed era stato visto che entrambi richiedono una minimizzazione dato un insieme di predittori $h \in H$, ma quali e quanti predittori usare? Un massimo numero di classificatori ci viene dato come fatto dall'analisi del rischio, ma chiaramente si puó fare di meglio. In questo progetto é stata utilizzata la seguente scelta: sia d il numero di attributi di un dataset, allora scegliamo come massimo numero di nodi il valore 2^{d+2}. In questo modo possiamo studiare solo i classificatori da 3 nodi (cioé un albero con solo la radice e due foglie) a 2^{d+2}. Anche questo é abbastanza grande come insieme di classificatori, specialmente quando il numero di attributi é grande.

É stato quindi utilizzato un altro concetto: fare un numero di passi massimi di confronti prima di fermarsi, nel caso di questo progetto il numero di passi massimi é 15. Nello specifico, iniziamo con il creare una lista contenente delle coppie formate dall'errore ottenuto dal predittore (errore che vogliamo minimizzare) e il numero di nodi del predittore. Utilizzeremo anche un'altra lista (formata da solo due elementi) che racchiude il "confine dell'intervallo" sinistro e destro.

Per chiarire meglio l'idea passiamo ad un esempio mostrando il procedimento per il primo passo. Consideriamo l'errore che si ottiene generando il predittore con 3 nodi e il predittore con 2^{d+2} nodi e usiamo questi due come confini dell'intervallo. Poiché sono uno in underfitting e l'altro in overfitting, il valore ottimale sarebbe all'interno dell'intervallo. Valutiamo l'errore del predittore con parametro $i = \frac{3+2^{d+2}}{2}$. Dopo averlo calcolato, lo confrontiamo con l'errore dei predittori ai confini correnti del nostro intervallo, cioé di 3 e di 2^{d+2}. Se il predittore con i nodi ha un errore minore o uguale all'errore del predittore con 2^{d+2} nodi, allora il nuovo confine destro dell'intervallo diventa i. Procedimento é analogo per il confine sinistro. L'algoritmo continua a cercare un nodo medio tra i due confini finché il numero di nodi del confine sinistro é uguale per due iterazioni di fila o il numero di passi da fare é stato superato.

Una problematica che puó essere banalmente pensata é la seguente: e se i due confini venissero modificati nella stessa iterazione? Non si potrebbe piú continuare ad iterare e il valore sarebbe quello al centro dell'intervallo invece di quello ottimale! Per questa ragione, se in un'iterazione i due confini sono uguali, il confine sinistro torna al valore del confine precedente prima della fine dell'iterazione.

3.2.3 Scelta della funzione $|\sigma(h)|$

Durante l'analisi del rischio, come si diceva, vengono utilizzati dei pesi $w(h)$ tale che $\sum_{h \in H} w(h) \leqslant 1$. Utilizzando la teoria dei codici, possiamo codificare ogni predittore ad albero h con N_h nodi usando una stringa binaria $\sigma(h)$ di lunghezza $O(n \log n)$ in maniera che non ci siano h e h' tali che $\sigma(h)$ é prefisso di $\sigma(h')$. In questi casi, poiché soddisfa la disuguaglianza di Kraft abbiamo che

$$2^{-|\sigma(h)|} \leqslant 1$$

Quindi $|\sigma(h)|$ rispetta la proprietá quando é una funzione del tipo $O(n \log n)$. Tuttavia, empiricamente si é notato che anche una funzione del tipo $O(n)$ sembra rispettare

la prorietá, ma converge verso un training error piú basso rispetto al $O(n \log n)$. Per questo é stata data all'utente la facoltá di scegliere tra le due funzioni durante l'esecuzione.

4 Risultati ottenuti

4.1 Confronto dei risultati

I risultati sono stati confrontati con il metodo "J48" di Weka in termini di tempo di esecuzione e il numero di nodi dell'albero prodotti dal predittore ottenuto.

Si mostra in tabella 1 i risultati ottenuti applicando la cross validazione sviluppato nel progetto sul dataset "Car Evaluation", applicando l'approccio derivato dal rischio, con $|\sigma(h)|$ uguale a $O(n)$ e a $O(n \log n)$, rispettivamente e il risultato ottenuto con Weka.

Dalla tabella possiamo osservare che gli algoritmi sviluppati in questo progetto utilizzano tutti meno nodi di Weka, tuttavia hanno un tempo di esecuzione é molto piú lungo, in special modo nella cross validazione questo periodo cresce molto.

Car Evaluation Dataset

	Numero di nodi	Tempo di esecuzione
Cross Validazione	129	5 secondi
$\|\sigma(h)\| = O(n)$	129	<1 secondo
$\|\sigma(h)\| = O(n \log n)$	11	1 secondo
Weka J48	182	0,07 secondi

TABLE 1: Tabella con il numero di nodi e tempo di esecuzione degli algoritmi sul dataset "Car Evaluation"

Vediamo ora i risultati ottenuti sul dataset "Nursery".

Come prima, si mostra in tabella 2 i risultati ottenuti applicando la cross validazione sul dataset "Nursery", i risultati ottenuti applicando l'approccio derivato dal rischio, con $|\sigma(h)|$ uguale a $O(n)$ e a $O(n \log n)$ e infine il risultato ottenuto con Weka.

Nursery Dataset

	Numero di nodi	Tempo di esecuzione
Cross Validazione	429	180 secondi
$\|\sigma(h)\| = O(n)$	227	12 secondi
$\|\sigma(h)\| = O(n \log n)$	15	15 secondi
Weka J48	511	0,09 secondi

TABLE 2: Tabella con il numero di nodi e tempo di esecuzione degli algoritmi sul dataset "Nursery"

In tabella 3 si puó invece vedere il test error calcolato sullo stesso test set con i diversi approcci sviluppati.

Osservando questi dati possiamo dire che, come si immaginava, l'algoritmo di cross validazione é molto lento e anche se da un risultato molto buono, non sembra valerne la pena. Il risultato in termini di tempo dell'algoritmo derivato con $|\sigma(h)| = O(n \log n)$ va molto meglio, non ai livelli di Weka ma é molto piú veloce della cross validazione. Ha un numero di nodi di gran lunga piú piccolo e di conseguenza anche l'errore é piú alto. Il metodo derivato dall'analisi del rischio ma con $|\sigma(h)| = O(n)$ invece é stato apprezzato perché é un tipo di equazione molto meno penalizzante sul numero di nodi rispetto a $O(n \log n)$ ma che comunque cerca un punto ottimale con pochi nodi. Inoltre é l'algoritmo piú veloce dei tre algoritmi, con un test error molto buono. L'algoritmo Weka invece, é di gran lunga il piú veloce, ma ha un numero di nodi decisamente piú grande rispetto a quelli sviluppati.

Test Error

	Car Evaluation	Nursery Evaluation
Cross Validazione	0.0555	0.0086
$\|\sigma(h)\| = O(n)$	0.0671	0.0496
$\|\sigma(h)\| = O(n \log n)$	0.1875	0.1287

TABLE 3: Tabella con i test error dei metodi sviluppati sui due dataset testati

5 Commenti conclusivi

Per quanto i classificatori implementati da WEKA siano il metodo piú veloce per classificare un dataset, é buono notare che non é il massimo in termini di numero di nodi, quindi nel caso di un numero grande di attributi ed e con molte etichette diverse, potrebbe non essere il modo migliore in caso di utilizzi questo tipo di struttura per decision making in videogames o in campi dove la velocitá di classificazione di dati nuovi é piú importante di quella generazione dell'albero stesso. Nei casi in cui il real time possa essere un problema, una soluzione che sembra interessante potrebbe essere quella di utilizzare il classificatore dopo aver fatto il training e utilizzarlo sui nuovi dati durante il tempo. Si pensi ad una intelligenza artificiale che utilizza un albero decisionale ad ogni istante per decidere che azione intraprendere, con ciascuna etichetta corrispondente ad un'azione.

Detto questo, una cosa che si é notata, anche se era immaginabile, é che l'algoritmo di cross validazione é computazionalmente molto pesante, perché bisogna controllare piú volte la lista per creare gli alberi, calcolare il validation error per ogni blocco del creato, e trovare la media dei validation error calcolati per ogni valore del parametro da controllare.

6 Appendice

Link sito **UCI Machine Learning**:
https://archive.ics.uci.edu/ml/index.php
Link pagina **Car Evaluation data set**:
https://archive.ics.uci.edu/ml/datasets/car+evaluation
Link pagina **Nursery data set**:
https://archive.ics.uci.edu/ml/datasets/nursery
Sito **WEKA**:
https://www.cs.waikato.ac.nz/ml/weka/