

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE
TECNOLOGIE DELL'INFORMAZIONE

CORSO DI INTELLIGENZA ARTIFICIALE

Elaborato Esercitativo

Work Projects

Milo Saverio
Previtera Gabriele
Pommella Michele

Prof. Neri Filippo

Anno 2016-2017

Decision Trees

Questo primo lavoro di gruppo si incentra sulla comprensione empirica dei Decision Trees, esempio di strumento ampiamente diffuso nel Machine Learning, e sul Problem Solving, metodo scientifico applicato dall'I.A. per la risoluzione dei problemi.

1.1 Esercizio 1

1.1.1

Un albero di decisione è un grafo delle decisioni e delle loro possibili conseguenze, costruito al fine di supportare l'azione decisionale. Rappresenta un importante strumento nel contesto dell'**Inductive Learning**, in cui ricopre il ruolo di modello predittivo su cui si basa il comportamento dell'agente. La sua struttura discende da un insieme di esempi dati e determina le regole di condizione-azione atte alla classificazione di esempi futuri. Dunque l'agente è in grado di apprendere da una serie di dati il comportamento da assumere in situazioni non specificate.

Come primo esperimento abbiamo adoperato un insieme di dati facenti riferimento a vari tipi di *Iris*, caratterizzati da quattro attributi. Queste grandezze sono dimensionalmente espresse tutte in cm. La dimensione complessiva del dataset è di 150 elementi. Per effettuare la prova è stato necessario formattare l'insieme dei dati originario, rendendolo consistente con le esigenze algoritmiche correlate al linguaggio utilizzato, *Python* nel nostro caso. Gli esempi del dataset si presentano nella forma:

5.1 , 3.5 , 1.4 , 0.2 , Iris —setosa

Abbiamo, dunque, determinato i valori di ciascun esempio attraverso le virgole delimitatorie, riconosciuto i valori numerici (precedentemente visti come stringhe) definendoli come tali ed eliminato eventualmente il carattere di *new line* al fine di evitare valori spuri.

```

1 def aprifile(fil="nomefile.txt"):
2     data=[]
3     for line in file(fil):
4         srt=line.split(',')
5         for count in range(0,len(srt)):
6             if(isfloat(srt[count])):
7                 srt[count]=float(srt[count])
8             else :
9                 srt[count]=srt[count].strip('\n')
10        data=data+[srt];
11    return data

```

Un campione di esempio formattato si presenterà, dunque, nella forma:

```
[5.1 , 3.5 , 1.4 , 0.2 , 'Iris -setosa ']
```

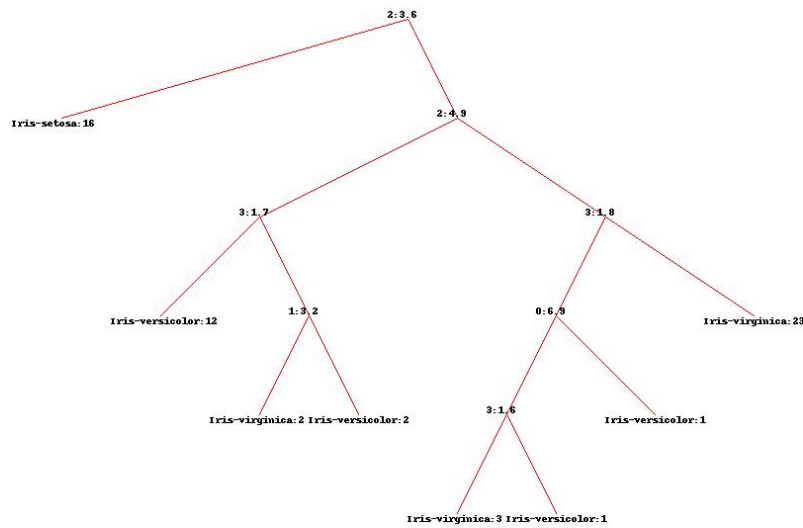
Per la determinazione di training set e test set si è utilizzata una funzione che selezioni un insieme casuale di esempi dal data set di partenza, con cardinalità determinata dai parametri di ingresso.

```

1 def createdataset(data,numdati):
2     tr=[]
3     te=[]
4     t=[]
5     for i in range(0,numdati):
6         t=random.choice(data);
7         tr=tr+[t]
8         num=data.index(t)
9         del data[num]
10    te=data
11    return(tr,te)

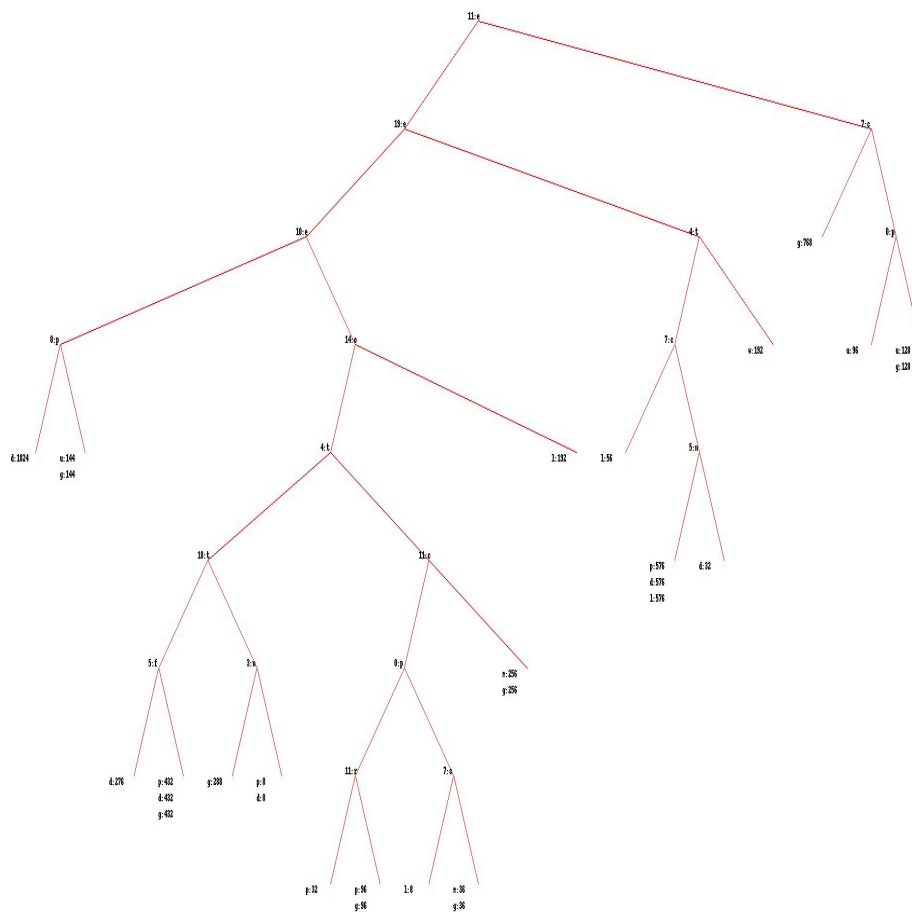
```

Decidiamo di usare un 40% per il training, quindi per l' apprendimento, ed un 60 % per il test.



Come si vede dalla figura, il nodo scelto come radice è la terza colonna(colonna numero 2 da programma, perché la numerazione effettuata dal codice della classificazione rimarca quella delle liste in python) ed il valore che minimizza la funzione d' entropia è 3.6, infatti ci permette di caratterizzare già 16 esempi.L'albero con l' informazioni di training, riesce a classificare tutti gli esempi, anche se il contenuto informativo non è alto, perché come vediamo, un attributo non riesce a caratterizzare nettamente un gruppo di esempi,infatti più volte sono richiamati per splittare nuovamente i dati, aumentando così la profondità dell' albero.Quest' ultimo,per tale ragione non è generico, molto probabilmente non ci offrirà ottimi risultati con dati a lui sconosciuti, si potrebbe pensare di applicare una potatura per renderlo più semplice, affinché da poter rispettare il rasoio di Okam.

Ora invece ci concentriamo su un diverso data set relativo ai funghi per determinare l' habitat in cui vivono, caratterizzato da ben 22 attributi, utilizziamo sempre la stessa percentuali di dati per il training ed il test set.



L'

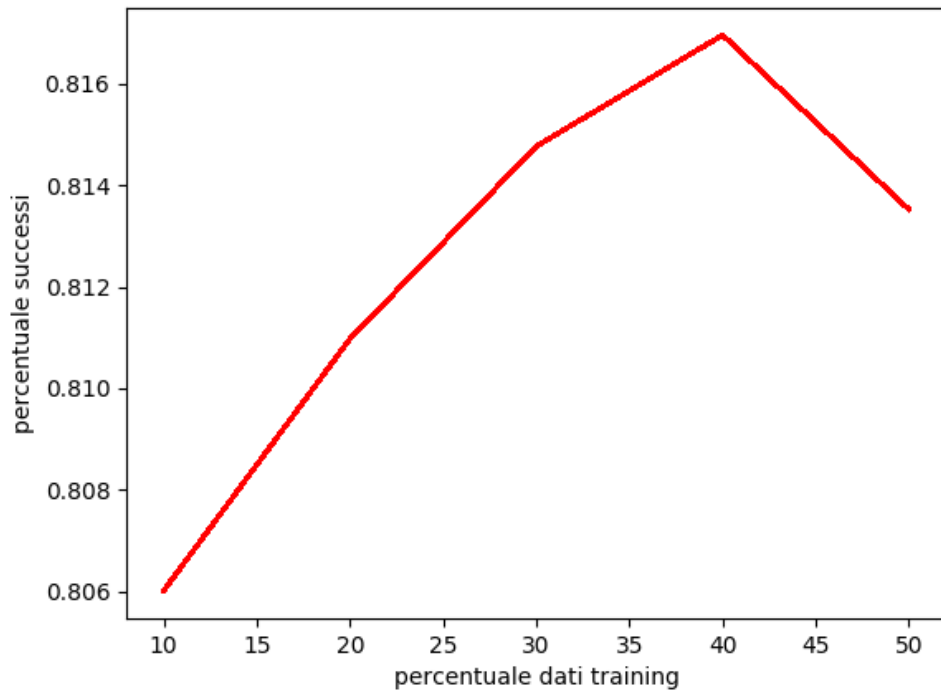
albero, come si può osservare non è in grado di classificare gli esempi distintamente. Tale risultato può essere conseguenza dal fatto che il numero di esempi usato per il training sia troppo piccolo, quindi non capace di permettere di discriminare appieno una scelta, oppure gli attributi utilizzati non sono abbastanza rappresentativi, non riuscendo a classificare appieno i vari esempi, quindi bisognerebbe introdurre altri attributi oppure modificare quelli usati, in questo stato l' albero può portare a decisioni includenti. Come si può osservare dai 2 esempi di data set, un alto numero di attributi, non è detto che ci porti a spiegare tutti gli esempi, infatti l' albero relativo ai funghi non utilizza tutte le caratteristiche ma solo alcune, quelle con un contenuto informativo più alto, abolendo le altre, però l' albero dei funghi molto probabilmente riuscirà a darci buoni risultati anche con casi sconosciuti essendo molto generico, quello degli iris no.

1.1.2

In questo esempio mostreremo al variare del training set e del data set il cambiamento valore della performance dell' albero visualizzando tramite un grafico. Il data set che andremo ad utilizzare è stato preso da un censimento, per identificare le persone che guadagnano più di cinquanta mila dollari all' anno. Le righe di codice adibite alla generazione del grafico sono quelle scritte in questa funzione:

```
1 def fperformance(data):
2     testc=data
3     percent=10
4     p=[]
5     perc=[]
6     t=[]
7     numdati=(int)((float)(len(testc))/100*percent);
8     for i in range(0,5):
9         (train,testc)=createdataset(testc,numdati)
10        t=t+train;
11        tree=buildtree(t)
12        p=p+[performance(tree,testc)]
13        perc=perc+[percent]
14        percent=percent+10;
15        line,=plt.plot(perc,p,'r-')
16    plt.xlabel('percentuale dati training')
17    plt.ylabel('percentuale successi')
18    line.set_antialiased(False)
19    plt.show()
```

la funzione essenzialmente dopo aver fatto una copia dei dati passati, fatto un paio di inizializzazione di liste che ci occorrono per il plot dei dati e aver determinato il numero di elementi da aggiungere al training set(togliendoli al test set), iterativamente per 5 volte divide i dati in train set e test set, dopo di che il train viene assegnata ad una lista t che incrementa ad ogni variazione e scritta su file da createdataset, viene costruito l' albero, se ne calcola il valore di performance, richiamando al suo interno la funzione classify che ritorna la classificazione fatta dall ' albero con gli attributi dell' esempio e confrontandoli con il valore del test, se l' albero è riuscito a classificare bene l' esempio allora si avrà un riscontro positivo, altrimenti negativo, alla fine performance ritorna la percentuale di positivi sul numero di test effettuati.Dopo di che si aggiungono i punti alla linea per il plot e alla fine viene mostrata su schermo, un esempio di curva è questo:



Come si vede dal grafico la nostra percentuale di apprendimento migliora fino al 40% dopodiché si percepisce un degrado dell'apprendimento quando arriviamo al 50% di dati utilizzati per il training-set.

Su tale risultato siamo giunti alla seguente considerazione " aumentando il training-set si è costruito un albero più specifico e di conseguenza con meno capacità di classificare gli esempi ". Dalla figura che mostra l'andamento della curva di apprendimento, si intuisce che la rappresentazione dell'ambiente così descritto dai data-set è ridondante. Infatti la crescita è lenta, al crescere del training-set dal 10% al 50% si evince che l'aumento è di circa del 1%. Per migliorare la rapidità della curva di performance dell'albero di decisione si potrebbero rimuovere alcuni attributi, se si è sicuri della rappresentazione, testando altre configurazioni dell' ambiente fino a che non vediamo miglioramenti evidenti nell' apprendimento, se ciò non avviene si può pensare di cambiare gli attributi che pensiamo ridondanti o quelli che ci danno un basso contenuto informativo, con altri o addirittura rifare il modello dell' ambiente .

1.1.3

Un agente in grado di apprendere mediante alberi di decisione fonda questo suo processo su principi di apprendimento induttivo (**inductive learning**).

L'apprendimento induttivo è una forma di apprendimento basata sull'induzione a partire da esempi dati. Esso, data una collezione di esempi (**training set**) della funzione **target** f che si vorrebbe imparare, mira a restituire una funzione h (**hypothesis**) che approssimi la f , anche se non è detto che riesca a trovare una che l'approssimi bene nello spazio delle funzioni. Concettualmente, il criterio nella determinazione di h tra le differenti funzioni dello spazio delle ipotesi dovrebbe essere legato, più che alla consistenza nello spiegare i dati, alla bontà dell'approssimazione e quindi alla capacità di generalizzazione per predire esempi non ancora incontrati. In questo senso, l'agente agisce in modo razionale poichè cerca di **decidere come comportarsi in situazioni a lui sconosciute basandosi su quelle già note**. Possiamo individuare proprio in questo aspetto una **forma di intelligenza**, determinata dall'agire razionalmente.

1.1.4

Le procedure di **Decision Tree Learning** consentono la costruzione di un albero di decisione "piccolo", consistente con gli esempi forniti in input per la costruzione di tale struttura. Ogni nodo interno all'albero corrisponde ad una condizione sul valore di un attributo(quindi una decisione), gli archi verso i nodi figli ai possibili valori per quell'attributo(una scelta intrapresa), le foglie alla classificazione(target raggiunto). Si ottiene così, attraverso i **path** dell'albero, una rappresentazione compatta delle regole di condizione-azione. L'albero di decisione prende in input una situazione descritta da un insieme di attributi e restituisce una decisione, ovvero il valore predetto di uscita per tale input, sulla base del cammino percorso. In questo senso possiamo parlare di apprendimento, poichè, **alla luce di un dato insieme di esempi, si viene a costituire un albero di decisione dalla ben determinata topologia e legge condizione-azione, utilizzabile per la classificazione di esempi futuri**.

1.2 Esercizio 2

Il problema descritto è del tipo *non deterministico e parzialmente osservabile*, quindi classificabile come **Contingency Problem**. L'algoritmo di navigazione proposto fa uso di un albero di ricerca con strategia di ricerca in profondità (**depth-first search**), la quale prevede di espandere primariamente il nodo più

profondo non espanso.

L'alterazione non deterministica delle distanze ci ha spinto a sottolineare l'importanza di **minimizzare il numero di città percorse** lungo il tragitto verso l'obiettivo. Un maggior numero di città attraversate aumenterebbe probabilisticamente la distanza percorsa. Sarebbe, quindi, auspicabile evitare, o minimizzare, il numero di processi di risalita dell'albero, che prevederebbero il ritorno alle stesse città più volte, e continuare nella navigazione in profondità, sfruttando, nel migliore dei casi, la possibilità di arrivare al **goal** senza attuare un **backtracking**. Inoltre, per evitare l'insorgere di cicli, si è previsto di tenere traccia dell'insieme dei nodi già esplorati, mediante l'utilizzo di una lista, ottenendo in definitiva un algoritmo del tipo **Graph Search**. La scelta dell'utilizzo della struttura dati lista semplice, è stata dettata dalla natura del problema, in quanto a differenza del problema analogo, ma diverso trattato in classe, in questo esempio le distanze tra le città non sono costanti e quindi la struttura dati lista ordinata in base alla distanza non si adatta a tale tipologia di problema.