Work Project 1 – Decision Trees

Esercizio 1

Ho inizialmente portato il codice preso dal libro *Programming Collective Intelligence,* del capitolo *Modeling with Decision Trees,* da Python2 a Python3, poi ho studiato il codice per capire quale funzione svolgesse quale compito e così via e l'ho eseguito sul dataset presente già nel programma. La prima cosa che ho notato è che la rappresentazione dell'albero di decisione è diversa da quella vista a lezione, ovvero da quella mostrata nel libro di Russell e Norvig: l'algoritmo produce in uscita un albero binario in cui i nodi non foglia non sono gli attributi, ma i valori degli attributi, e gli archi uscenti sono (a destra) *true* ed (a sinistra) *false*. In altre parole, in un dato nodo non si confronta il valore di un attributo dell'osservazione con ogni possibile valore dell'attributo, ma solo con il valore specificato dal nodo, producendo un risultato booleano. L'albero quindi raggiunge una profondità maggiore, dato che un attributo può essere testato più volte, ma è più conveniente a livello implementativo un albero binario rispetto ad un albero in cui ogni nodo ha N archi uscenti. A tal punto bisognava scegliere un dataset, da UC Irvine ML Repository, ed ho scelto il dataset *Car Evaluation*: http://archive.ics.uci.edu/ml/datasets/Car+Evaluation
La funzione target è l'accettabilità di un'automobile, che si valuta secondo degli attributi raggruppabili in Prezzo Totale, Caratteristiche Tecniche e Comfort. Questi gruppi si estendono nel seguente modo:

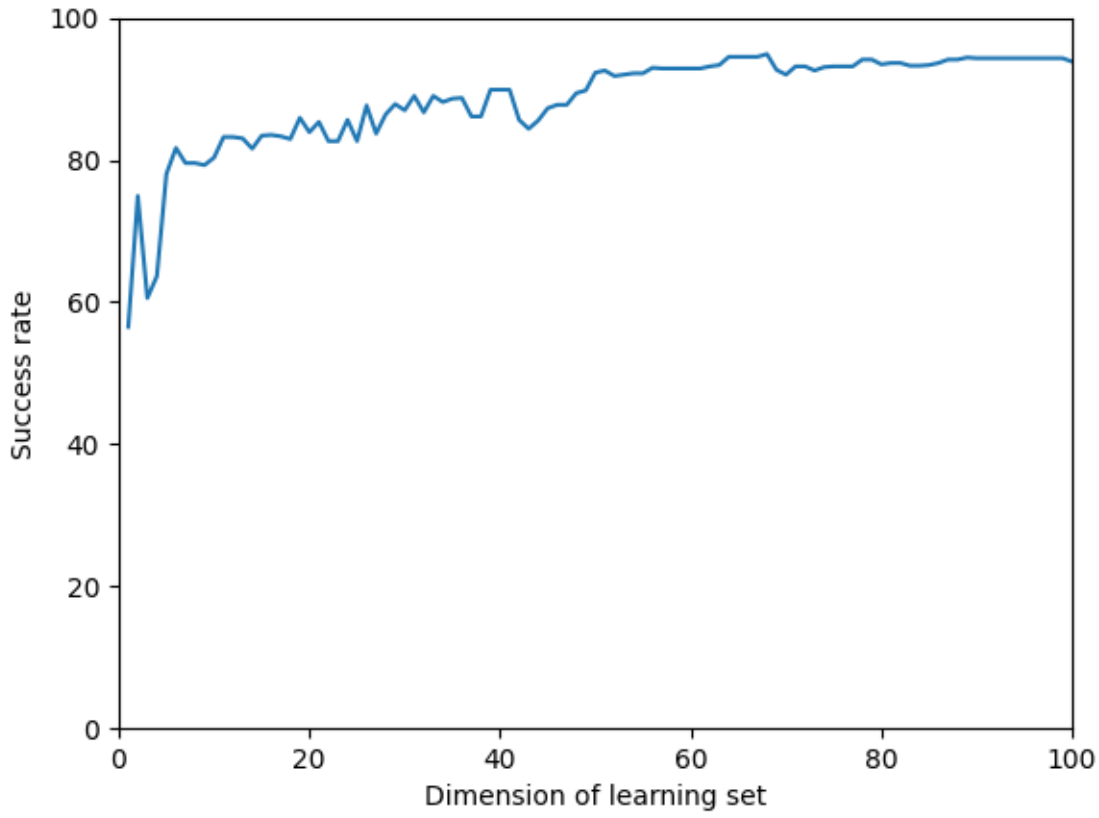
```
CAR                     accettabilità dell'automobile
. PRICE                 prezzo totale
. . buying              prezzo d'acquisto
. . maint               costo della manutenzione
. TECH                  caratteristiche tecniche
. . COMFORT             comfort
. . . doors             numero di porte
. . . persons           capacità in termini di persone trasportabili
. . . lug_boot          dimensione del bagagliaio
. . safety              sicurezza stimata dell'automobile

Dunque gli attributi per la classificazione sono buying, maint, doors, persons, lug_boot
e safety. I possibili valori che tali attributi possono assumere sono i seguenti:

buying       v-high, high, med, low
maint        v-high, high, med, low
doors        2, 3, 4, 5-more
persons      2, 4, more
lug_boot     small, med, big
safety       low, med, high
```

Il dataset contiene abbastanza istanze da coprire completamente lo spazio degli attributi (1728 osservazioni) dunque il problema è ben posto. Tuttavia, la distribuzione delle classificazioni è piuttosto sbilanciata, ovvero nella maggior parte dei casi, l'automobile è classificata come "non accettabile". Vediamo la distribuzione delle classificazioni ed allo stesso tempo le possibili diverse classificazioni:

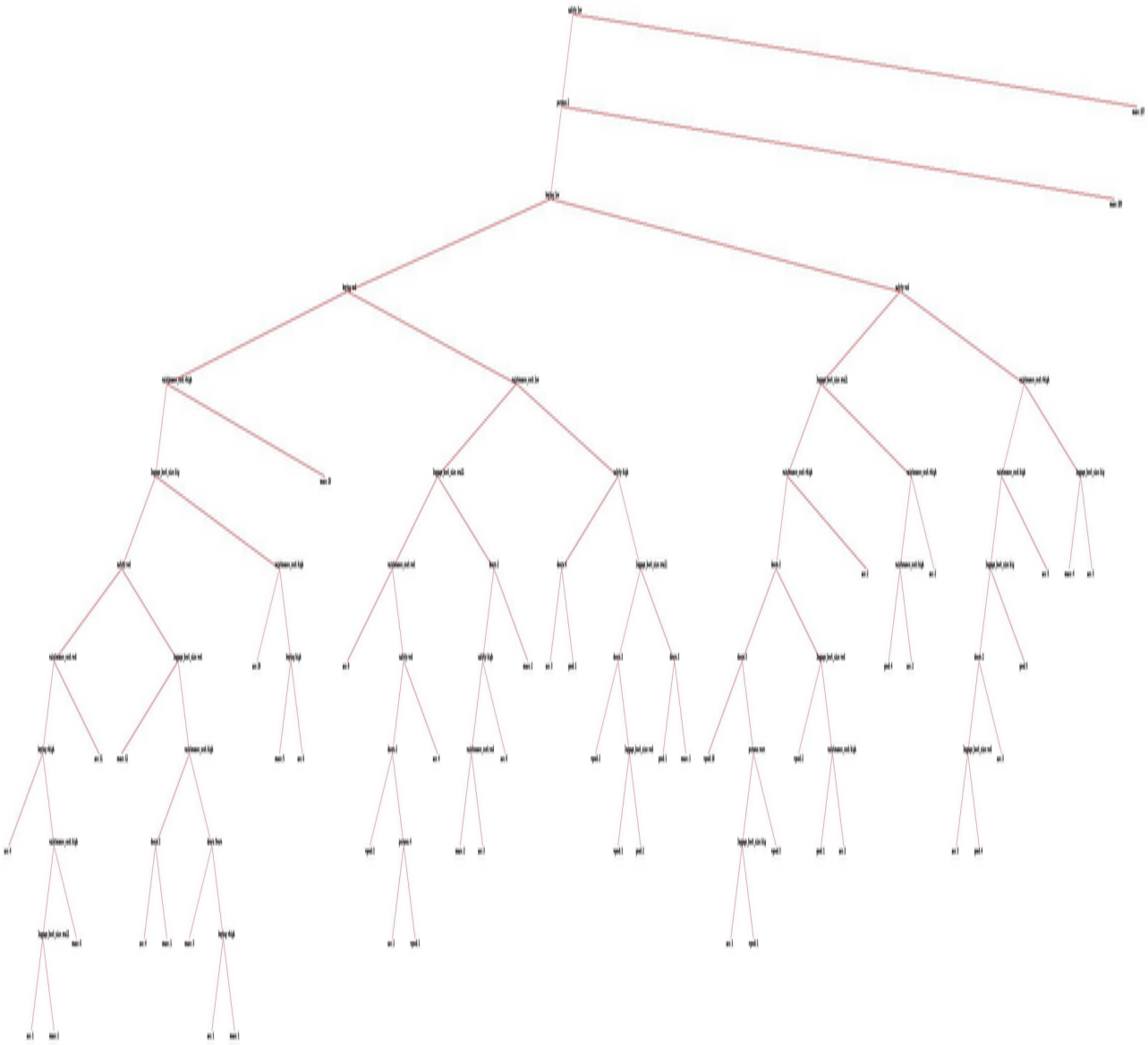
```
class      N          N[%]
------------------------------
unacc     1210     (70.023 %)
acc        384     (22.222 %)
good        69     ( 3.993 %)
v-good      65     ( 3.762 %)
```

A questo punto ho implementato il parsing del dataset nel mio programma, compito abbastanza semplice dato che le tuple erano ordinate ed i valori separati da virgole, con le prime 6 colonne contenenti gli attributi nel seguente ordine:
buying,maint,doors,persons,lug_boot,safety

E l’ultima colonna contenente la classificazione. Dunque, essendo pronto per fare dei test, ho diviso il dataset in learning set e test set, ottenendo due distinte liste di osservazioni. Inizialmente ho fatto una divisione del tipo 10%-90%, 20%-80% … 90%-10%. Tuttavia, ero consapevole del fatto che l’aumento del success rate solidale con l’aumento della dimensione del learning set era un falso positivo, dato che si facevano meno test. Allora ho diviso 50/50 usando cioè un test set fisso e modificando la quantità di osservazioni da usare per l’apprendimento in quel restante 50%, ovvero ad esempio il 10% del 50% del dataset totale come learning set. Ho modificato questa percentuale applicata al 50% in modo lineare dall’1% al 100%, valutando il success rate, ed ho appurato che la percentuale di successi, in cui il risultato fornito dall’albero di decisione era in accordo con il risultato previsto, non andava oltre il 72%; inoltre chiaramente il massimo, cioè questo 72%, non veniva raggiunto usando il 100% delle osservazioni per l’apprendimento, quindi il problema non era dovuto al fatto di aver utilizzato poche osservazioni. Osservando meglio il dataset, ho notato che questo è ordinato in base agli attributi, quindi ad esempio prendendo solo le prime N righe come learning set può capitare che in queste righe l’attributo della prima colonna ha sempre lo stesso valore. Dunque, a monte della divisione del dataset in learning set e test set, ho fatto un random shuffle delle righe, in modo tale che entrambi i set coprissero completamente lo spazio degli attributi. Così facendo ho ottenuto un success rate massimo del 95%, sufficiente da poter dire che il modello è realizzabile, anche se a rigore la curva dovrebbe tendere asintoticamente al 100%.
Ecco il grafico che mostra il success rate all’aumentare della quantità di dati usati per l’apprendimento:



Ed ecco l’albero di decisione relativo al *best fitting*, che si è rivelato del 68% (del 50% preso come candidato per l’apprendimento, dunque il best fitting è il learning set composto dal 34% delle osservazioni del dataset totale), con un success rate del 94.89% cioè circa del 95%.



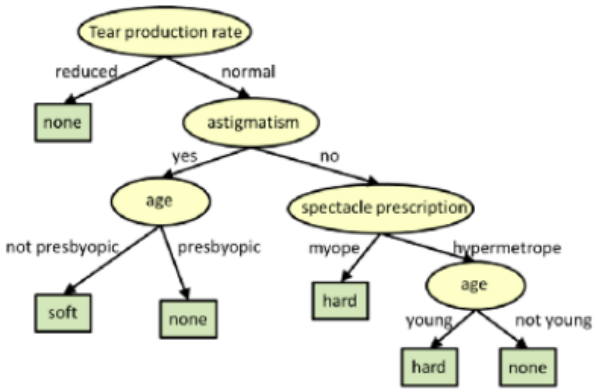
Purtroppo l’albero risulta piuttosto grande, quindi potrebbe essere conveniente navigarlo seguendo una rappresentazione testuale:

safety:low?
T->{'unacc': 157}
F->persons:2?
  T->{'unacc': 109}
  F->buying:low?
    T->safety:med?
      T->maintenance_cost:vhigh?
        T->luggage_boot_size:big?
          T->{'acc': 3}
          F->{'unacc': 4}
        F->maintenance_cost:high?
          T->{'acc': 5}
          F->luggage_boot_size:big?
            T->{'good': 5}

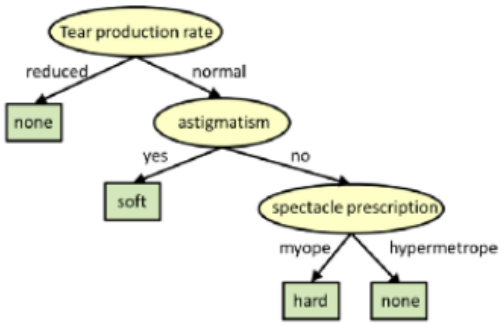
```
        F->doors:2?
          T->{'acc': 3}
          F->luggage_boot_size:med?
            T->{'good': 4}
            F->{'acc': 3}
    F->luggage_boot_size:small?
      T->maintenance_cost:vhigh?
        T->{'acc': 2}
        F->maintenance_cost:high?
          T->{'acc': 2}
          F->{'good': 4}
      F->maintenance_cost:vhigh?
        T->{'acc': 2}
        F->doors:2?
          T->luggage_boot_size:med?
            T->maintenance_cost:high?
              T->{'acc': 2}
              F->{'good': 1}
            F->{'vgood': 2}
          F->doors:3?
            T->persons:more?
              T->{'vgood': 3}
              F->luggage_boot_size:big?
                T->{'vgood': 1}
                F->{'acc': 1}
            F->{'vgood': 10}
F->buying:med?
  T->maintenance_cost:low?
    T->safety:high?
      T->luggage_boot_size:small?
        T->doors:2?
          T->{'unacc': 1}
          F->{'good': 1}
        F->doors:2?
          T->luggage_boot_size:med?
            T->{'good': 2}
            F->{'vgood': 1}
          F->{'vgood': 2}
      F->doors:4?
        T->{'good': 1}
        F->{'acc': 3}
    F->luggage_boot_size:small?
      T->doors:2?
        T->{'unacc': 2}
        F->safety:high?
          T->{'acc': 8}
          F->maintenance_cost:med?
            T->{'acc': 3}
            F->{'unacc': 2}
      F->maintenance_cost:med?
        T->safety:med?
          T->{'acc': 4}
```

```
          F->doors:2?
            T->persons:4?
              T->{'vgood': 1}
              F->{'acc': 1}
            F->{'vgood': 2}
        F->{'acc': 8}
    F->maintenance_cost:vhigh?
     T->{'unacc': 28}
     F->luggage_boot_size:big?
      T->maintenance_cost:high?
       T->buying:high?
        T->{'acc': 6}
        F->{'unacc': 5}
       F->{'acc': 20}
      F->safety:med?
       T->luggage_boot_size:med?
        T->maintenance_cost:high?
          T->doors:5more?
            T->buying:vhigh?
              T->{'unacc': 1}
              F->{'acc': 1}
            F->{'unacc': 3}
          F->doors:2?
            T->{'unacc': 1}
            F->{'acc': 4}
        F->{'unacc': 12}
       F->maintenance_cost:med?
        T->{'acc': 11}
        F->buying:vhigh?
          T->maintenance_cost:high?
            T->{'unacc': 6}
            F->luggage_boot_size:small?
              T->{'unacc': 1}
              F->{'acc': 1}
          F->{'acc': 4}
```

Notare che secondo questo modello l’attributo che incide di più sull’accettabilità dell’automobile è la sicurezza, ovvero se la sicurezza è bassa, l’automobile è subito classificata come non accettabile; dopodiché, un altro attributo che incide molto è il numero di persone che l’automobile può trasportare: se può trasportare solo 2 persone, l’automobile è classificata come non accettabile. Se si è curiosi si può continuare a seguire lo sviluppo dell’albero in profondità. Le classificazioni, cioè le foglie, sono rappresentate nelle parentesi graffe ed è presente anche informazione riguardante il numero di osservazioni che, navigando l’albero in un certo modo, ha portato ad una determinata classificazione. La presenza di foglie in cui è indicata una sola osservazione, o comunque un numero basso di osservazioni, potrebbe essere una prova dell’*overfitting* del modello.
Un’operazione con la quale si può semplificare il modello, rimuovendo questi sotto-alberi dati da poche osservazioni e quindi statisticamente poco validi, si chiama *pruning*. Per fare il pruning bisogna definire un valore di soglia detto *information gain*, quindi ci vuole un’analisi accurata dei dati che si hanno a disposizione e quindi è un argomento che va oltre questa trattazione. Tuttavia può essere interessante visualizzarne un semplice esempio:



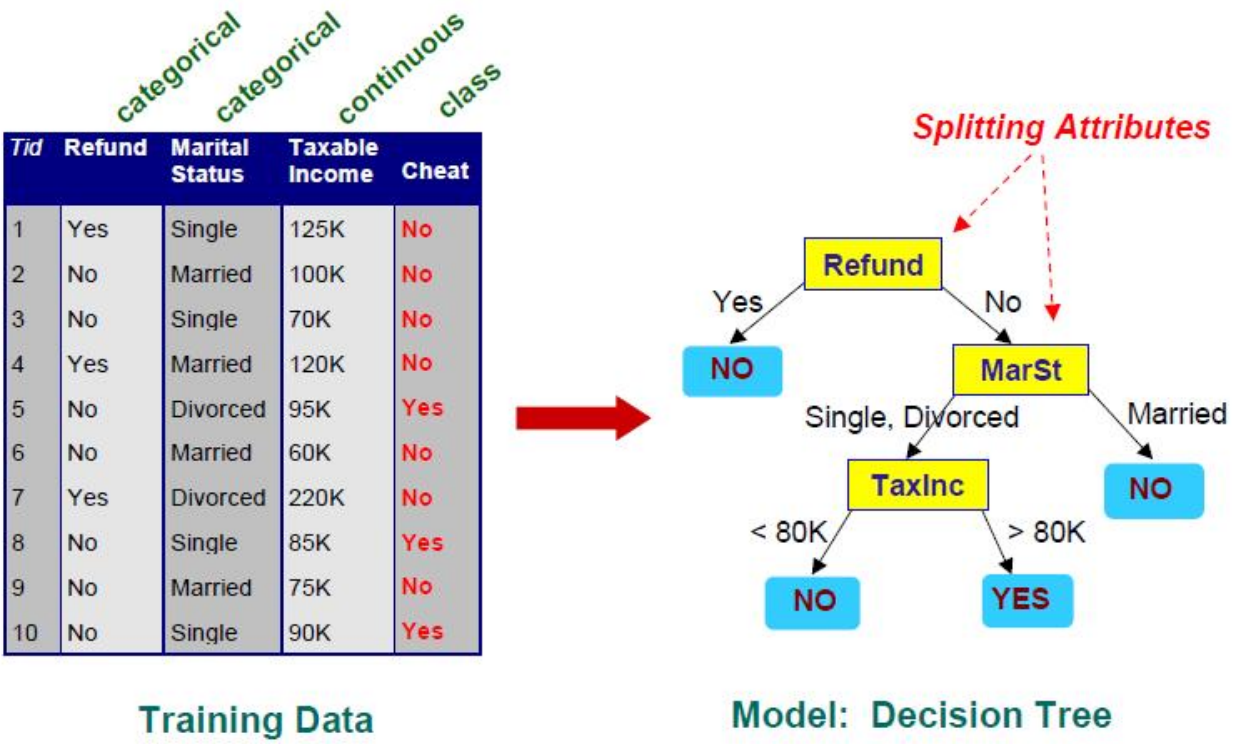
Original Tree



Pruned Tree

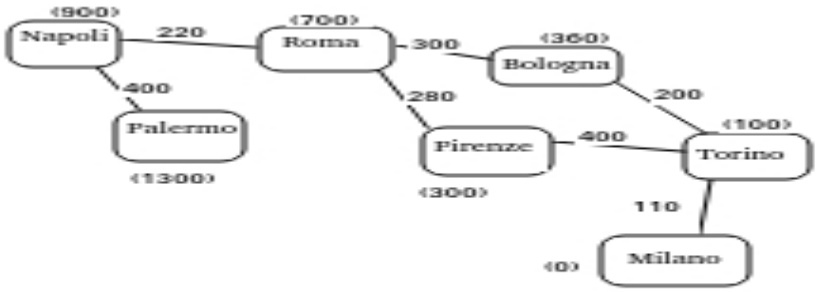
Esercizio 2

Un agente in grado di apprendere Alberi di Decisione è considerato intelligente perché è in grado di fornire risposte, ovvero di prendere una decisione, anche quando gli si presentano situazioni nuove, le quali comprendono casi imprevisti; a tal proposito, una misura della sua intelligenza la si può fare misurando l'error rate sui casi nuovi. Ovvero, dato un insieme di osservazioni, detto *dataset*, dividendo tale insieme in due sottoinsiemi, *learning set* e *test set*, usando il learning set per costruire un albero di decisione e confrontando le decisioni prese dall'agente sui casi del test set con le decisioni previste dalle osservazioni, è possibile fare una stima statistica su come l'agente si comporterà nei casi futuri. Dunque, più l'error rate si avvicina allo 0, più l'agente si considera intelligente. Per apprendimento si intende la costruzione dell'Albero di Decisione, ovvero è un apprendimento induttivo: dal particolare al generale. Questo vuol dire che si ha un insieme di punti ed una funzione ignota, e si fa un'ipotesi riguardo a questa funzione facendo un'interpolazione dei punti dati come esempio. Si cerca un modello semplice, per evitare di fare ipotesi troppo stringenti sulla funzione, ovvero il modello non deve "dipendere troppo" dai punti dati come esempio; in caso contrario il modello mancherebbe di generalità e si parlerebbe di *overfitting*. I punti in realtà sono delle tuple, composte da un insieme di attributi e da una decisione, detta *classificazione*, associata ai valori che tali attributi assumono. Tali attributi possono essere categorici, ovvero assumere valori discreti, oppure continui, ed in tal caso bisogna discretizzarli definendo degli intervalli. Nell'Albero di Decisione, che quindi rappresenta la conoscenza dell'agente, i nodi non-foglia sono attributi, gli archi da un attributo a nodi inferiori sono i valori che l'attributo assume e le foglie sono le classificazioni. La questione del modello semplice in questo caso vuol dire avere un albero con un numero di nodi possibilmente minimo. Questo è un esempio di come si può ottenere un Albero di Decisione da un insieme di tuple:



Esercizio 3

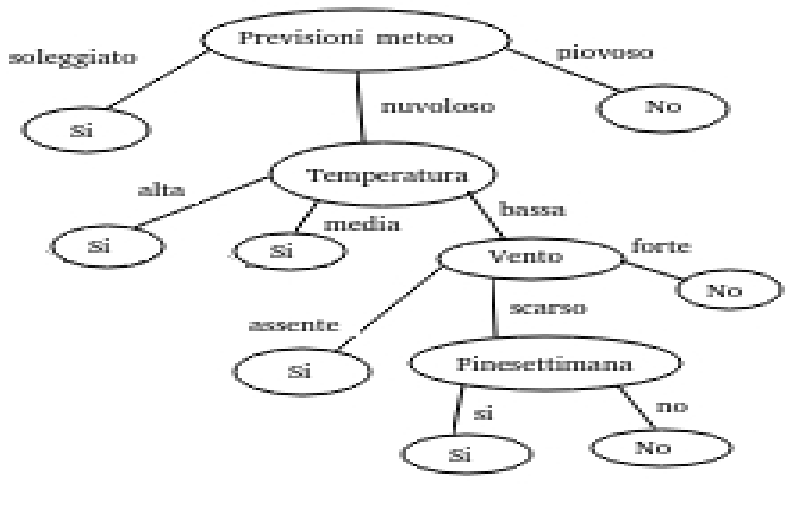
L'algoritmo A* è un algoritmo *informato* di *problem-solving* che si differenzia dagli altri per la strategia di espansione della frontiera (*fringe*). Ovvero, dato un grafo dello spazio degli stati, si vuole trovare un albero di ricerca che mostri il percorso da un dato nodo iniziale ad un determinato nodo finale; per ogni nodo, la lista di possibili nodi ciascuno dei quali è candidato ad essere il prossimo nodo visitato si chiama frontiera, e dipende sia dai vicini (*neighbors*) del nodo corrente, sia dai vicini dei nodi precedentemente visitati. La strategia di espansione influisce sull'ordine di questa lista di nodi. A* ordina la lista in ordine crescente assegnando un peso f(n) ai nodi:
$f(n) = g(n) + h(n)$
dove g(n) è la distanza percorsa dal nodo iniziale, ovvero dal nodo in cui è presente lo stato iniziale, al nodo candidato per l'espansione, e dunque questa è un'informazione legata agli archi; h(n) invece è un'euristica che costituisce una stima della distanza tra il nodo candidato per l'espansione ed il nodo obiettivo, ovvero il nodo in cui è presente lo stato che si vuole raggiungere. I casi in cui si applica sono quelli in cui si vuole evitare di continuare ad espandere un percorso se ad un certo punto diventa troppo costoso, ovvero serve per la ricerca del percorso *ottimale*. Abbiamo detto che h(n) è un'euristica, e da qui si ha il fatto che l'algoritmo è informato: in quali casi questa euristica va bene? Formalmente, un'euristica che "va bene" si dice *ammissibile*, ed è tale se non sovrastima mai la distanza tra il nodo corrente ed il nodo obiettivo, ovvero detta c(n) la funzione del costo effettivo, h(n) è ammissibile se $h(n) \leq c(n)$ per ogni n. Consideriamo un esempio di navigazione stradale, con distanze tra le città più che altro immaginarie ma con dati sufficienti a mostrare l'applicazione dell'algoritmo; il modello infatti può dipendere da vari fattori, la cosa importante è vedere come si comporta l'algoritmo per un dato modello. Sugli archi mettiamo il costo necessario per la transizione da un nodo ad un altro, e quindi sommando i costi dal nodo iniziale al nodo candidato si ottiene la g(n), mentre sui nodi, in parentesi, mettiamo la h(n).



Il nodo di partenza è Napoli, il nodo di arrivo è Milano.
Partendo da Napoli, i candidati per l'espansione sono Palermo e Roma, e rispettivamente si ha:
f(Palermo) = 400 + 1300 = 1700
f(Roma) = 220 + 700 = 920
Quindi la frontiera sarà [Roma, Palermo], e sarà scelto il nodo in testa alla lista, ovvero Roma.
Dopodiché, da Roma i candidati per l'espansione sono Bologna e Firenze, per i quali:
f(Bologna) = (220 + 300) + 360 = 880
f(Firenze) = (220 + 280) + 300 = 800
La frontiera sarà [Firenze, Bologna, Palermo]: notare che Palermo fa ancora parte della frontiera, naturalmente. Sarà scelto il nodo in cui è contenuto lo stato identificato dal nome Firenze.
Come candidato per l'espansione, si ha Torino come vicino di Firenze, che per comodità chiameremo Torino_da_Firenze:
f(Torino_da_Firenze) = (220 + 280 + 400) + 100 = 1000
La frontiera sarà [Bologna, Torino_da_Firenze, Palermo], quindi piuttosto che andare a Torino passando per Firenze, conviene andare a Bologna e vedere se passando da lì si ottiene un percorso più breve.
f(Torino_da_Bologna) = (220 + 300 + 200) + 100 = 820
La frontiera sarà [Torino_da_Bologna, Torino_da_Firenze, Palermo]. Quindi l'algoritmo favorisce il percorso che attraversa Bologna per arrivare a Torino. Dopodiché:
f(Milano_passando_per_Bologna) = (220 + 300 + 200 + 110) + 0 = 830
La frontiera sarà [Milano_…, Torino_da_Firenze, Palermo]. Dato che il nodo in testa alla lista è il nodo obiettivo, l'algoritmo termina. Tuttavia, se fossimo arrivati al nodo obiettivo, ma questo non fosse in testa alla lista, l'algoritmo continuerebbe. E' possibile infatti dimostrare che, in caso di euristica ammissibile, l'algoritmo A* è ottimale, ovvero trova sempre il percorso migliore, cioè con il costo più basso. Notare che, nell'espandere i nodi, non si è mai considerato come nodo candidato il nodo precedente al nodo corrente: questa è la distinzione tra *tree search* e *graph search.*

Esercizio 4

Supponiamo di avere un albero di decisione riguardante la scelta di giocare oppure no a tennis, quindi classificazione binaria. Gli attributi con i quali le osservazioni sono descritte sono i seguenti: Previsioni meteo, Vento, Temperature, Finesettimana, Quantità di cibo assunto in giornata.
Abbiamo per esempio il seguente Albero di Decisione:



Notare come, dalle osservazioni del learning set sulla base delle quali, seguendo un determinato algoritmo, è stato costruito l'albero di decisione, sia ininfluente l'attributo riguardante la Quantità di cibo assunto in giornata; questo potrebbe essere dovuto o al fatto che sono state utilizzate poche osservazioni, oppure al fatto che tale attributo sia ridondante, ovvero effettivamente non influisce sulle decisioni. Supponiamo ora che Marco debba decidere se giocare a tennis oggi oppure no, date le seguenti condizioni:
Previsioni meteo: nuvoloso
Temperatura: bassa
Vento: scarso
Finesettimana: no
Quantità di cibo assunto in giornata: molto

Dunque si costruisce una tupla con i valori ordinati coerentemente con gli attributi a cui fanno riferimento, e si dà la tupla in pasto all'albero di decisione. Il primo attributo, cioè quello che sta alla radice dell'albero, è "Previsioni meteo", dunque si prende dalla tupla il valore relativo a tale attributo e nell'albero si segue l'arco etichettato con tale valore. Si itera questo procedimento fino ad arrivare ad un nodo foglia, il quale è la classificazione. Se la classificazione è "No", vuol dire che l'albero di decisione dirà a Marco di non giocare a tennis, e se è "Si" vorrà dire che Marco giocherà a tennis. Con le condizioni date in input, la risposta sarà "No".

Esercizio 5

Supponiamo di avere un insieme di esempi riguardanti la scelta di un individuo di giocare a tennis oppure no, date una serie di condizioni. Queste condizioni sono descritte dagli attributi Previsioni meteo, Vento, Temperatura, Finesettimana, Quantità di cibo assunto in giornata. Notare che gli attributi Vento e Temperatura, così come in realtà anche Quantità di cibo, in realtà non sono categorici, ma sono continui. Tuttavia è possibile farne una discretizzazione, ad esempio una temperatura maggiore di 25 °C si può etichettare come alta, compresa tra 10 e 25 °C media e minore di 10 °C bassa. Vediamo quindi gli esempi:

Esempio	Previsioni meteo	Vento	Temperatura	Fine settimana	Quantità cibo	Esito: giocherà a tennis?
X_1	Soleggiato	Scarso	Alta	Si	Bassa	Si
X_2	Soleggiato	Forte	Media	No	Media	Si
X_3	Soleggiato	Forte	Bassa	Si	Media	Si
X_4	Nuvoloso	Forte	Media	No	Media	Si
X_5	Piovoso	Assente	Bassa	Si	Bassa	No
X_6	Nuvoloso	Forte	Bassa	No	Media	No
X_7	Piovoso	Scarso	Alta	No	Alta	No
X_8	Nuvoloso	Assente	Bassa	Si	Alta	Si
X_9	Nuvoloso	Scarso	Bassa	Si	Media	No
X_{10}	Nuvoloso	Scarso	Bassa	No	Media	No
X_{11}	Piovoso	Forte	Media	Si	Media	No
X_{12}	Nuvoloso	Assente	Alta	No	Media	Si

A tal punto, da questi esempi si vuole ottenere una ipotesi generale, un modello che si ipotizza essere valido per tutte le possibili combinazioni di valori degli attributi. La struttura dati che cattura questo scopo è l'albero di decisione, e si può costruire, a partire da esempi attribute-based, con il seguente algoritmo:

Decision Tree Learning Algorithm

function DTL(*examples*, *attributes*, *default*) **returns** a decision tree

  if *examples* is empty **then return** *default*
  else if all *examples* have the same classification **then return** the classification
  else if *attributes* is empty **then return** MODE(*examples*)
  else
    best ← CHOOSE-ATTRIBUTE(*attributes*, *examples*)
    tree ← a new decision tree with root test *best*
    for each value v_i of *best* **do**
      $examples_i$ ← {elements of *examples* with *best* = v_i}
      subtree ← DTL($examples_i$, *attributes* − *best*, MODE(*examples*))
      add a branch to *tree* with label v_i and subtree *subtree*
    return *tree*

Dato che in 7 esempi su 12 la decisione è Si, costruiamo l'albero di decisione con questo algoritmo usando come valore iniziale di default "Si" (anche se non ci servirà, questa considerazione è solo per completezza, dato che l'algoritmo in passi successivi passa come valore di default la moda degli esempi, che inizialmente è proprio pari al valore che noi gli abbiamo passato).

Il punto cruciale è la linea " best = Choose-Attribute(attributes, examples) ", in cui si sceglie l'attributo con la minore entropia, in modo da ottenere un albero più semplice. L'entropia di un insieme si definisce:
$E(s) = - \Sigma\ p_i \log_2(p_i)$
Questo nel caso in cui ci siano 2 possibili classificazioni; in generale, con N possibili classificazioni, l'entropia si normalizza, ovvero si porta tra 0 e 1, dividendo il risultato della sommatoria per $\log_2(N)$. L'entropia di un attributo si definisce come somma pesata delle entropie dei sottoinsiemi che si ottengono considerando i soli esempi in cui l'attributo preso in considerazione assume i suoi possibili valori. Tale somma si può pesare semplicemente come media, ma statisticamente ha più senso dare un peso maggiore ai sottoinsiemi in cui sono presenti più osservazioni, quindi detto M il numero di esempi dell'insieme iniziale e K_i il numero di esempi del sottoinsieme relativo al valore i dell'attributo in considerazione, si associa all'entropia del valore i il valore K_i / M. Calcoliamo allora le entropie degli attributi per determinare qual è il nodo radice, ovvero quello che incide di più sulla decisione riguardo alla partita di tennis.

E(Previsioni Meteo) = 3/12 E(soleggiato) + 6/12 E(nuvoloso) + 3/12 E(piovoso) = ¼ [(-3/3 $\log_2(3/3)$ - 0/3 $\log_2(0/3)$) + 2(-2/6 $\log_2(2/6)$ - 4/6 $\log_2(4/6)$) + (-0/3 $\log_2(0/3)$ – 3/3 $\log_2(3/3)$)] = 0.459

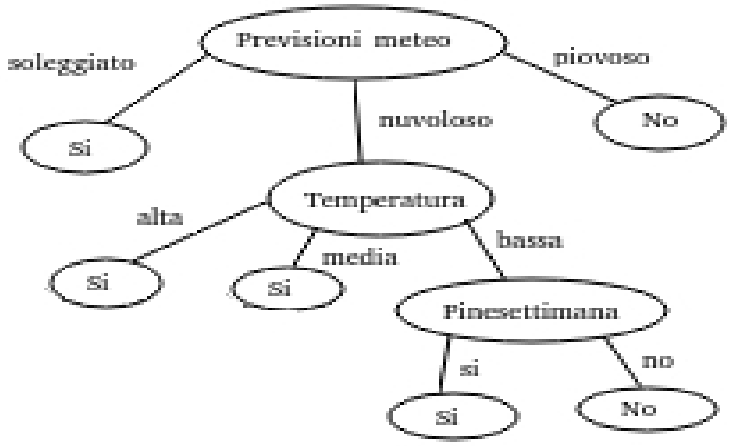
E(Vento) = 4/12 E(scarso) + 5/12 E(forte) + 3/12 E(assente) = 4/12 (-2/4 $\log_2(2/4)$ – 2/4 $\log_2(2/4)$) + 5/12 (-3/5 $\log_2(3/5)$ – 2/5 $\log_2(2/5)$) + 3/12 (-2/3 $\log_2(2/3)$ – 1/3 $\log_2(1/3)$) = 0.967

E(Temperatura) = 3/12 E(alta) + 3/12 E(media) + 6/12 E(bassa) = 3/12 (-2/3 $\log_2(2/3)$ – 1/3 $\log_2(1/3)$) + 3/12 (-2/3 $\log_2(2/3)$ – 1/3 $\log_2(1/3)$) + 6/12 (-3/6 $\log_2(3/6)$ – 3/6 $\log_2(3/6)$) = 0.959

E(Finesettimana) = 6/12 E(si) + 6/12 E(no) = 6/12 (-4/6 $\log_2(4/6)$ – 2/6 $\log_2(2/6)$) + 6/12 (-3/6 $\log_2(3/6)$ – 3/6 $\log_2(3/6)$) = 0.959

E(Quantità cibo) = 2/12 E(bassa) + 8/12 E(media) + 2/12 E(alta) = 2/12 (-1/2 $\log_2(1/2)$ – ½ $\log_2(1/2)$) + 8/12 (-5/8 $\log_2(5/8)$ – 3/8 $\log_2(3/8)$) + 2/12 (-1/2 $\log_2(1/2)$ – ½ $\log_2(1/2)$) = 0.969

Dunque si ha che il best attribute è "Previsioni meteo"; itero sui valori che tale attributo può assumere ottenendo i sottoinsiemi di esempi, e di questi l'unico che ha incertezza è quello con valore "nuvoloso" (gli altri ricadono nella clausola dell'algoritmo "else if all examples have the same classification then return the classification"). Dunque gli esempi che restano, cioè che sono passati ricorsivamente a DTL, e che non hanno tutti la stessa classificazione, sono X_4, X_6, X_8, X_9, X_{10}, X_{12}. Calcolando le entropie degli attributi rimanenti (cioè tutti tranne Previsioni meteo) su questi esempi, si ottiene che le minime sono quelle di Temperatura e Vento, alla pari a 4/6. Supponiamo che la funzione Choose-Attribute tra le due per qualche motivo scelga l'attributo Vento. Anche in questo caso, solo per un valore dell'attributo si ha ancora incertezza, cioè per il valore "bassa", e gli attributi rimanenti sono X_6, X_8, X_9, X_{10}. Dopodiché, ripetendo il procedimento, si ha che l'entropia minima è associata all'attributo Finesettimana, ed è 0. Si ottiene il seguente albero di decisione:



Confrontando quest'albero con quello dell'esercizio 4, si nota che entrambi sono *consistenti* con l'insieme di esempi dati, tuttavia questo è più semplice ed è pertanto più generale.

Esercizio 6

Lo spazio degli stati è l'insieme di tutti i possibili stati in cui l'ambiente può trovarsi; non sempre lo spazio degli stati è completamente noto a priori, ad esempio in un problema di esplorazione si ha solo una visione parziale dello spazio degli stati. Il grafo degli stati, o meglio grafo dello spazio degli stati, è un grafo in cui ogni nodo contiene uno stato appartenente allo spazio degli stati, ed ogni arco rappresenta un'azione il cui effetto è quello di far evolvere l'ambiente da uno stato ad un altro. L'albero di ricerca è una visione del grafo dello spazio degli stati che si ottiene a partire da un nodo iniziale visitando i vicini di tale nodo, poi i vicini dei vicini e così via, fino ad arrivare ad un nodo finale, detto obiettivo. In generale, un Tree Search si costruisce inizializzando l'albero con un nodo radice che contiene lo stato iniziale del problema, dopodiché: se non ci sono candidati per l'espansione, l'algoritmo termina e fallisce, altrimenti a seconda della strategia utilizzata si sceglie un determinato nodo per l'espansione; se questo nodo contiene lo stato obiettivo allora l'algoritmo ritorna la soluzione cioè la strada che porta dal nodo iniziale al nodo obiettivo, altrimenti si aggiunge il nodo all'albero di ricerca e si controlla di nuovo se ci sono nodi candidati per l'espansione, fino alla terminazione dell'algoritmo. Gli algoritmi Tree Search si differenziano tra loro per la strategia di espansione della frontiera, ovvero per la condizione che regola la scelta del prossimo nodo da visitare. La prima distinzione che si fa è tra algoritmi non-informati ed informati. Quelli non-informati non hanno appunto l'informazione necessaria per stimare quanto lo stato presente nel nodo corrente disti dallo stato obiettivo, mentre quelli informati sì e dunque si basano su euristiche. I principali algoritmi non-informati sono BFS, DFS e Uniform-Cost Search.
BFS: la frontiera è una coda FIFO.
DFS: la frontiera è una coda LIFO.
Uniform-Cost Search: la frontiera è una coda ordinata in base alla distanza dal nodo iniziale al nodo candidato per l'espansione, quindi sul costo di raggiungimento di un nodo.
I parametri su cui si valutano gli algoritmi sono: completezza, complessità temporale, complessità spaziale ed ottimalità. Di quelli non-informati, l'unico completo ed ottimale (nel caso in cui il costo delle azioni sia costante in tutto il grafo) è il BFS. Tuttavia, esso è poco conveniente in termini di complessità sia temporale sia spaziale, in quanto esse sono entrambe esponenziali rispetto alla profondità dell'albero. I principali algoritmi informati sono il greedy search (anche detto best-first) e l'A*.
Greedy search: la frontiera è ordinata in ordine crescente secondo l'euristica h(n).
A*: la frontiera è ordinata in ordina crescente secondo la funzione $f(n) = g(n) + h(n)$, dove $g(n)$ è il costo complessivo del percorso dal nodo iniziale al nodo candidato per l'espansione.
Il greedy search può andare in loop se non si tiene conto dei nodi precedentemente visitati, ed in generale non è ottimale. L'A* è invece completo ed ottimale, in caso di euristica ammissibile, ed è veloce in termini di complessità temporale e spaziale perché evita di espandere percorsi costosi, dunque trova il percorso migliore e lo trova espandendo pochi nodi.