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

Work Project 1

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: Decision Trees

1.1.1 Esperimento: creazione Decision Tree

Un albero di decisione è un modello di rappresentazione delle decisioni e delle loro possibili conseguenze, costruito al fine di supportare l'azione decisionale. Costituisce 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.

Iris

Come primo esperimento abbiamo adoperato un insieme di dati facenti riferimento a vari tipi di *Iris*, caratterizzati da quattro attributi: lunghezza del sepalo, larghezza del sepalo, lunghezza del petalo, larghezza del petalo. Queste grandezze sono dimensionalmente espresse tutte in cm. Il *target* è, appunto, la tipologia di Iris. 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.

```
def aprifile(fil="nomefile.txt"):
        data=[]
        for line in file(fil):
                srt=line.split(',')
                for count in range(0,len(srt)):
                        if(isfloat(srt[count])):
                                srt[count]=float(srt[count])
                        else :
                                srt[count]=srt[count].strip('\n')
                data=data+[srt];
        return data
```

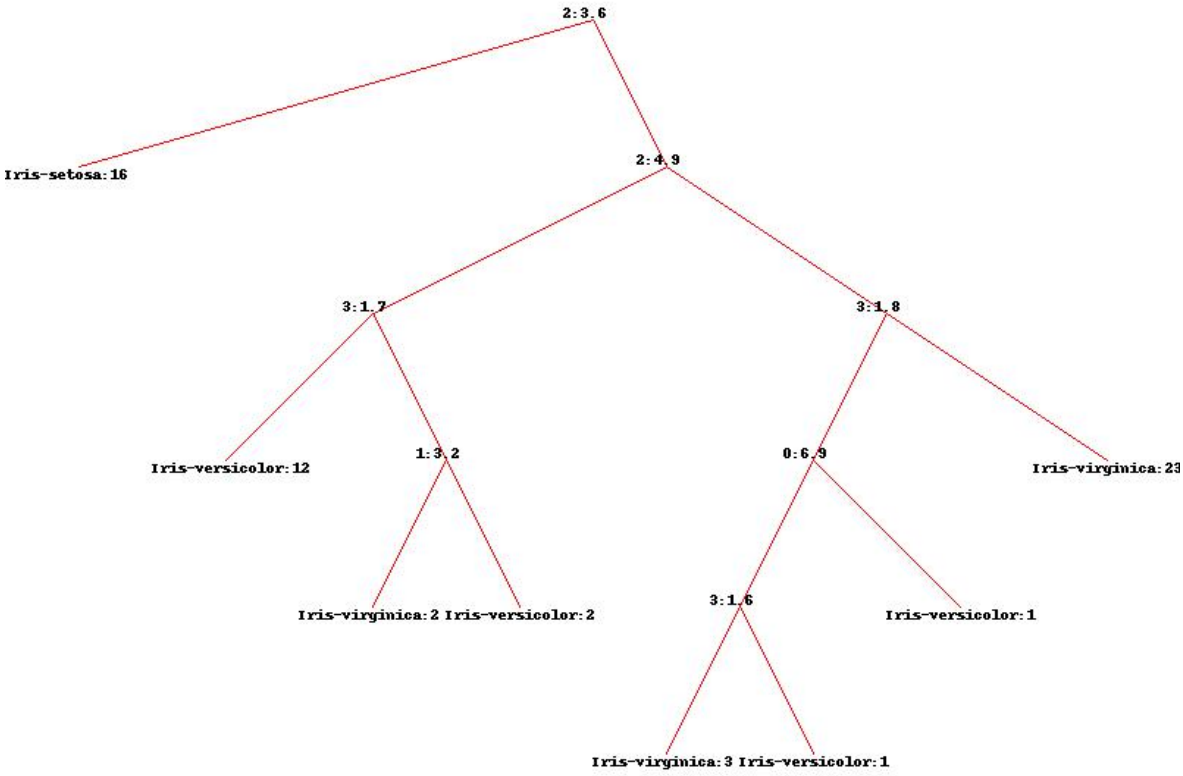
Un campione di esempio formattato si presenterà, dunque, nella forma:

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

Decidiamo di usare il 40% dei dati forniti dal data set come training set, quindi per l'apprendimento, ed il restante 60 % per il test. 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 in base alla percentuale desiderata.

```
def createdataset(data,numdati):
        tr=[]
        te=[]
        t=[]
        for i in range(0,numdati):
                t=random.choice(data);
                tr=tr+[t]
                num=data.index(t)
                del data[num]
        te=data
        return(tr,te)
```

Costituito il training set, vengono sfruttati gli algoritmi forniti per la costruzione dell'albero (`buildtree`) e la relativa rappresentazione (`drawtree`).



Otteniamo, infine, l'agognato albero di decisione!
Come si evince dalla figura, il nodo scelto come radice è relativo alla terza colonna (colonna numero 2 da programma, poiché la numerazione rimarca quella delle liste in python) ed il valore che ne minimizza la funzione d'entropia è 3.6. Notiamo che questa scelta ci permette già la classificazione di 16 esempi. L'albero, attraverso le informazioni del training set, classifica tutti gli esempi, anche se, evidentemente, il *guadagno informativo* degli attributi non sarà alto. Osserviamo, infatti, che un attributo non riesce a caratterizzare nettamente un gruppo di esempi. Conseguentemente, ciascuno di essi viene richiamato più volte per lo *split* dei dati, aumentando inevitabilmente la profondità dell' albero.
Alla luce di queste considerazioni, molto probabilmente questa classificazione non sarà abbastanza generalizzante ed, ipotizzando di utilizzare un test set sufficientemente vario, non ci offrirà ottimi risultati in termini di

performance. Si potrebbe, però, ricercare una maggiore generalizzazione tramite l'applicazione di tecniche di *pruning*, che diminuirebbero la profondità dell'albero.

Mushrooms

Analizziamo ora il data set *Muschroom*, relativo ai funghi. Il fine sarà di determinarne l' habitat. Il dataset è caratterizzato da ben 21 attributi e 8124 istanze. Utilizziamo sempre le medesime percentuali di dati per il training set ed il test set (40%-60%).
L' albero, come si può osservare dalla figura 1.1 nella pagina seguente, non è in grado di classificare gli esempi distintamente. In questo stato l'albero può portare a decisioni inconcludenti. Risultato analogo si è ottenuto al seguito di differenti esperimenti sul medesimo data set, rivelandosi tutt'altro che un *unicum*. Ciò ci ha portato ad interpretare questo esito come conseguenza del fatto che gli attributi utilizzati non siano abbastanza rappresentativi, non riuscendo a classificare pienamente i vari esempi.
Notiamo, in conclusione, che un grande numero di attributi non necessariamente porta a spiegare tutti gli esempi.

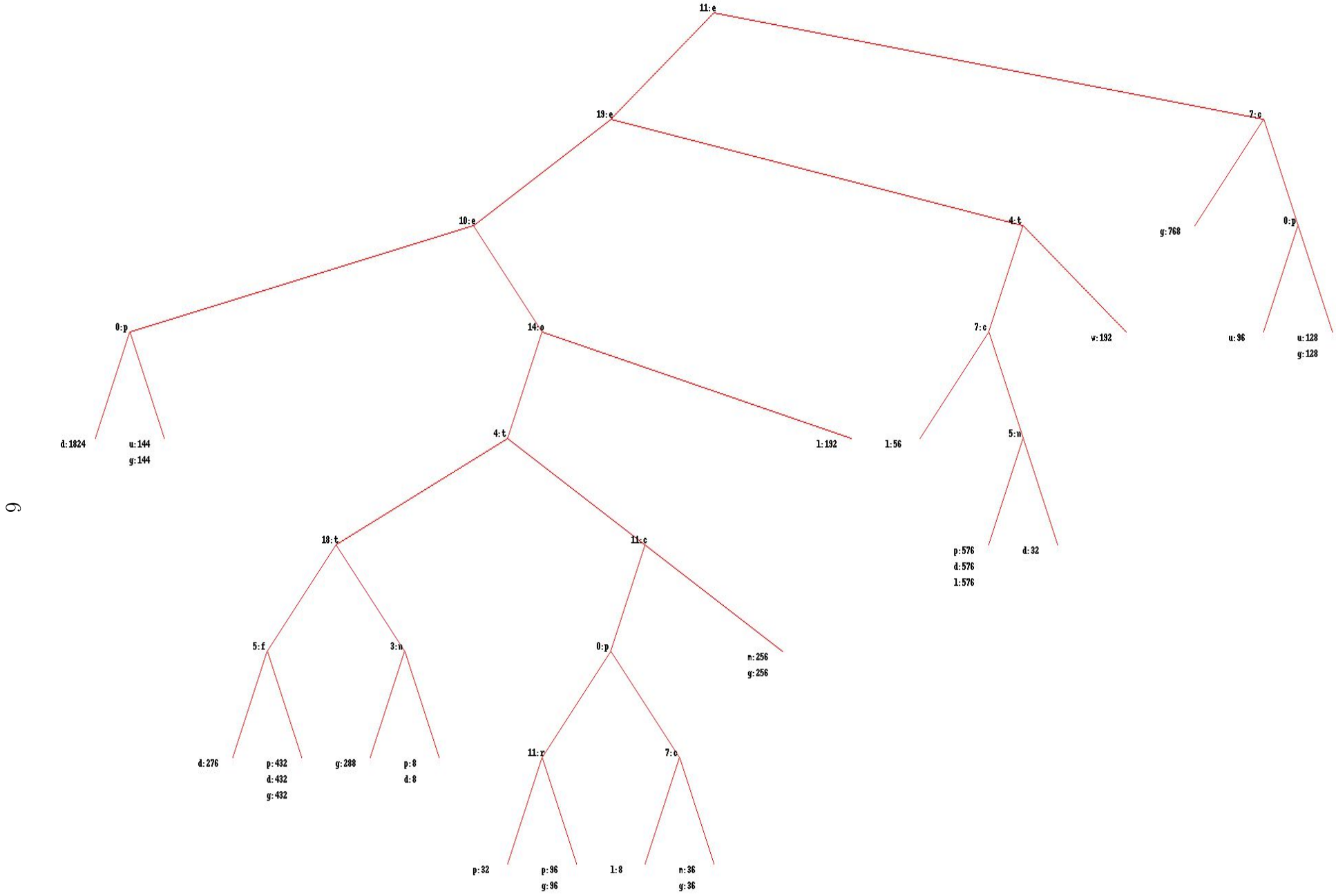


Figura 1.1: Mushrooms Decision Tree

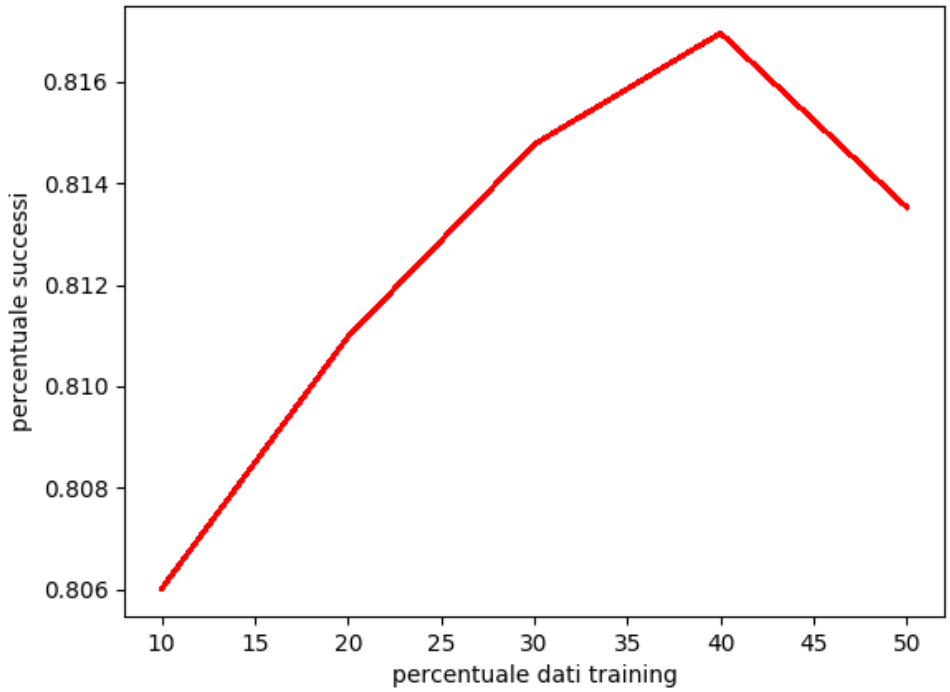
1.1.2 Esperimento: Learning Curve

In questo esempio mostreremo al variare del training set e del data set, il 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,delle inizializzazioni 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 create data set divide i dati in train set e test set,i dati del train vengono assegnati ad una lista t che verrà usata effettivamente per il training, 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 alla fine si confronta la classificazione ottenuta con il valore del test, se l' albero è riuscito a classificare bene l' esempio allora si avrà un riscontro positivo, altrimenti negativo, performance ha come valore di ritorno la percentuale di esiti 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 dataset 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 essere ridondati o quelli che ci danno un basso contenuto informativo, con altri o addirittura rifare il modello dell' ambiente .

1.1.3 Domanda 1

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 Domanda 2

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, poiché, **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: Problem Solving

Il problema descritto è del tipo *non deterministico* e *parzialmente osservabile*, quindi classificabile come Contingency Problem.Essendo le distanze geografiche elastiche, possiamo immaginare che forze esterne ad ogni cambio di città del robot fanno cambiare la geografia del pianeta incrementando o

diminuendo le distanze tra le varie città, per tale motivo l'algoritmo di navigazione proposto fa uso di un albero di ricerca con strategia di analisi 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.