Teoria dell'Informazione/RTI **Anno Accademico 2022–2023**

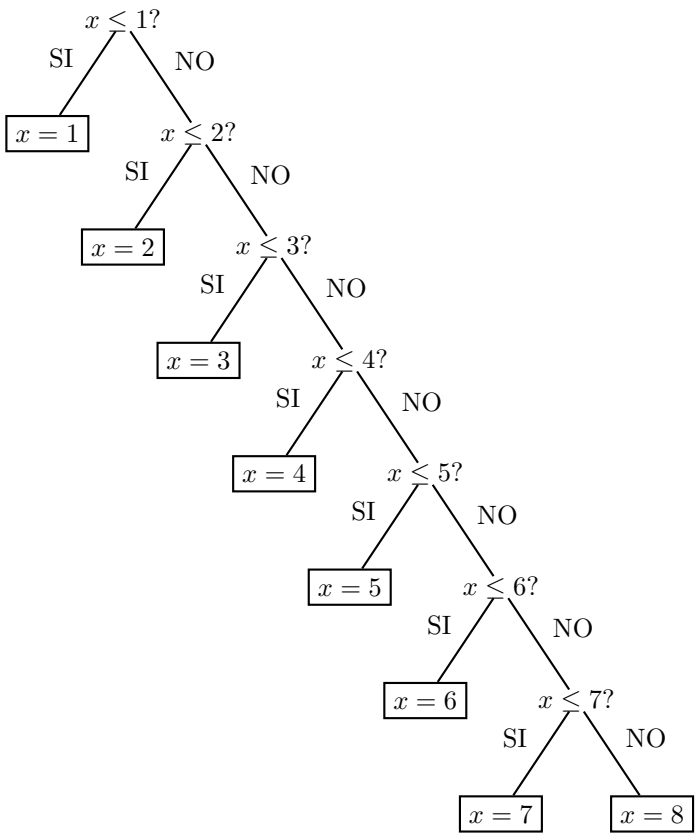
Lezione 6

Ugo Vaccaro

Data una variabile casuale $X = \begin{pmatrix} 1 & 2 & \cdots & n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}$ che assume valori i, con probabilità p_i, per $i = 1, \ldots, n$, vogliamo studiare quanto è facile (o difficile) scoprire il valore assunto da X, mediante domande del tipo "è il valore di X minore o al più uguale a j?", dove j è un numero in $\{1, \ldots, n\}$. La misura della difficoltà che intendiamo stimare sarà pari al numero di domande necessarie (e sufficienti) per determinare il valore assunto dalla variabile casuale X.

Innanzitutto, studiamo più in dettaglio il problema della ricerca di elementi in un insieme finito. Supponiamo di avere dato un insieme finito $S = \{1, \ldots, n\}$ ed un numero incognito $x \in \{1, \ldots, n\}$. Vogliamo determinare il valore di x mediante domande del tipo "è $x \leq j$?", dove $j \in S$. Ad esempio, un algoritmo potrebbe effettuare le seguenti domande, in sequenza. Innanzitutto chiedere "è $x \leq 1$?". Se la risposta è SI, allora avremmo scoperto che $x = 1$. Se la risposta è NO; allora l'algoritmo effettuerà la domanda "è $x \leq 2$?". Se la risposta è SI, abbiamo scoperto che $x = 2$. Se la risposta è NO, l'algoritmo effettuerà la domanda "è $x \leq 3$?", e così via continuando. Il problema che ci poniamo è di stabilire quante domande l'algoritmo effettuerà per scoprire il valore incognito x (e capire che c'entra l'entropia con questo problema). Ovviamente, per il particolare algoritmo che stiamo esaminando vale che il numero di domande che esso eseguirà è esattamente pari ad x (tranne quando $x = n$, in cui eseguirà $n - 1$ domande). Per cui l'algoritmo eseguirà, nel caso peggiore, $n - 1$ domande. É possibile far meglio? Consideriamo quest'altro algoritmo. Esso inizia chiedendo "è $x \leq \lceil n/2 \rceil$?". Se la risposta è SI, allora l'algoritmo ha appreso che il numero incognito x **non** può essere nessuno dei numeri $\lceil n/2 \rceil + 1, \lceil n/2 \rceil + 2, \ldots, n$ e quindi prosegue la ricerca di x nell'insieme $\{1, \ldots, \lceil n/2 \rceil\}$. E come? Effettuando la domanda "è $x \leq \lceil (1/2)\lceil n/2 \rceil \rceil$?", e così via. Analogamente, se la risposta alla prima domada fosse stata NO, allora l'algoritmo saprebbe che il numero incognito x **non** poteva essere nessuno dei numeri in $\{1, \ldots, \lceil n/2 \rceil\}$, pertanto avrebbe proseguito la ricerca di x nell'altra metà dell'insieme, ovvero in $\{\lceil n/2 \rceil + 1, \ldots, n\}$. E come? Effettuando la domanda "è $x \leq \lceil 3n/2 \rceil$?", e così via. Come si vede, ad ogni domanda (e relativa risposta ricevuta) l'algoritmo è in grado di (circa) dimezzare la cardinalità dell'insieme in cui l'elemento incognito x può trovarsi. Di conseguenza, dopo un numero generico i di domande l'algoritmo ha appreso che l'elemento incognito x si trova in un insieme di cardinalità al più $\lceil n/2^i \rceil$. Quando $\lceil n/2^i \rceil = 1$ avremo ovviamente scoperto chi è il valore di x. Ne segue che il numero di domande effettuate dall'algoritmo per scoprire il valore incognito x sarà pari, al più, a $\lceil \log n \rceil$, qualunque sia il valore di $x \in \{1, \ldots, n\}$.

In molte applicazioni pratiche è noto non solo l'insieme S cui l'elemento x incognito appartiene, ma è anche data una distribuzione di probabilità su S che specifica, appunto, la probabilità con cui l'elemento x è uno dei valori di S. Supponiamo quindi che sia nota la distribuzione di probabilità $\mathbf{p} = (p_1, \ldots, p_n)$, dove p_i=probabilità che l'elemento incognito x sia pari a $i \in S$. La conoscenza di $\mathbf{p}$ può facilitare la ricerca del valore x. Consideriamo il seguente esempio. Sia $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$ e $p_1 = 1/2, p_2, = 1/4, p_3 = 1/8, p_4 = 1/16, p_5 = 1/32, p_6 = 1/64, p_7 = 1/128, p_8 = 1/128$. Vediamo come si comportano i due algoritmi di prima, quando la distribuzione di probabilità sui possibili valori dell'incognita x è quella sopra data. Il primo algoritmo procedeva ponendo le domande "è $x \leq 1$?", "è $x \leq 2$?", ... Per semplificare l'analisi dell'algoritmo, diamone una rappresentazione grafica mediante un albero, nel modo seguente.

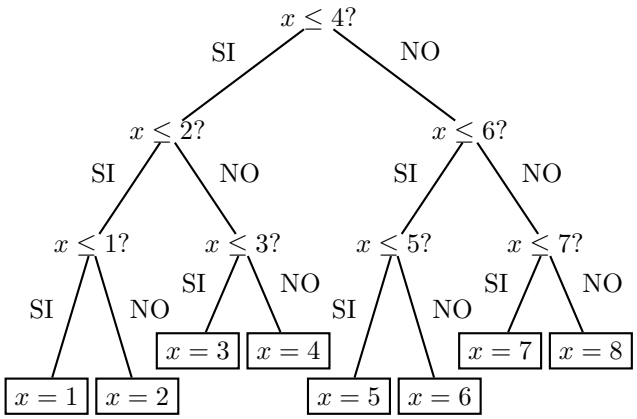


Algoritmo 1

Come abbiamo visto, l'algoritmo eseguirà una sola domanda se $x = 1$ (e ciò sappiamo accade con probabilità pari a $1/2$), eseguirà due domande se $x = 2$ (e ciò sappiamo accade con probabilità pari a $1/4$), eseguirà tre domande se $x = 3$ (e ciò sappiamo accade con probabilità pari a $1/8$), eseguirà quattro domande se $x = 4$ (e ciò sappiamo accade con probabilità pari a $1/16$), eseguirà cinque domande se $x = 5$ (e ciò sappiamo accade con probabilità pari a $1/32$), eseguirà sei domande se $x = 6$ (e ciò sappiamo accade con probabilità pari a $1/64$), eseguirà sette domande se $x = 7$ (e ciò sappiamo accade con probabilità pari a $1/128$), eseguirà sette domande se $x = 8$ (e ciò sappiamo accade con probabilità pari a $1/128$). In questo caso, quindi, non ha senso parlare del numero di domande nel caso peggiore, ma è più sensato calcolare il numero **medio** di domande che l'algoritmo esegue, che in questo caso è pari a

$$1 \times \frac{1}{2} + 2 \times \frac{1}{4} + 3 \times \frac{1}{8} + 4 \times \frac{1}{16} + 5 \times \frac{1}{32} + 6 \times \frac{1}{64} + 7 \times \frac{1}{128} + 7 \times \frac{1}{128} \approx 1.984\ldots$$

Il secondo algoritmo, invece, che corrisponde al seguente albero



Algoritmo 2

esegue sempre tre domande, senza tener di conto delle diverse probabilità con cui x assume valori in $\{1, 2, 3, 4, 5, 6, 7, 8\}$. Pertanto, il numero medio di domande che l'algoritmo eseguirà sarà pari a

$$3 \times \frac{1}{2} + 3 \times \frac{1}{4} + 3 \times \frac{1}{8} + 3 \times \frac{1}{16} + 3 \times \frac{1}{32} + 3 \times \frac{1}{64} + 3 \times \frac{1}{128} + 3 \times \frac{1}{128} = 3 > 1.984\ldots$$

Ovvero, in questa situazione si sono ribaltati i ruoli ed il miglior algoritmo è il primo. Prima di procedere con la nostra analisi, calcoliamo la entropia della distribuzione di probabilità $\mathbf{p} = (p_1, \ldots, p_8)$, con $p_1 = 1/2, p_2, = 1/4, p_3 = 1/8, p_4 = 1/16, p_5 = 1/32, p_6 = 1/64, p_7 = 1/128, p_8 = 1/128$, e osserviamo che

$$\begin{aligned} H(\mathbf{p}) &= \frac{1}{2}\log 2 + \frac{1}{4}\log 4 + \frac{1}{8}\log 8 + \frac{1}{16}\log 16 + \frac{1}{32}\log 32 + \frac{1}{64}\log 64 + \frac{1}{128}\log 128 + \frac{1}{128}\log 128 \\ &= 1 \times \frac{1}{2} + 2 \times \frac{1}{4} + 3 \times \frac{1}{8} + 4 \times \frac{1}{16} + 5 \times \frac{1}{32} + 6 \times \frac{1}{64} + 7 \times \frac{1}{128} + 7 \times \frac{1}{128} \approx 1.984\ldots \end{aligned}$$

Notiamo che l'entropia di $\mathbf{p} = (p_1, \ldots, p_8)$ è esattamente pari al *numero medio* di domande che il primo algoritmo esegue. É un caso? Intendiamo scoprirlo.

Per studiare questo fatto, osserviamo innanzitutto che ogni algoritmo di ricerca nell'insieme S definisce, implicitamente, una codifica degli elementi di S. Ad esempio, se osserviamo l'Algoritmo 2 e la sequenza di risposte che otteniamo nel caso in cui $x = 1$, otteniamo la sequenza SI SI SI, mentre l'Algoritmo 1 restituisce la sequenza di risposte SI, se $x = 2$, l'Algoritmo 2 restituisce la sequenza SI SI NO, mentre l'Algoritmo 1 restituisce la sequenza di risposte NO SI, se $x = 3$, l'Algoritmo 2 restituisce la sequenza SI NO SI, mentre l'Algoritmo 1 restituisce la sequenza di risposte NO NO SI, se $x = 4$, l'Algoritmo 2 restituisce la sequenza SI NO NO, mentre l'Algoritmo 1 restituisce la sequenza di risposte NO NO NO SI, se $x = 5$, l'Algoritmo 2 restituisce la sequenza NO SI SI, mentre l'Algoritmo 1 restituisce la sequenza di risposte NO NO NO NO SI, se $x = 6$, l'Algoritmo 2 restituisce la sequenza NO SI NO, mentre l'Algoritmo 1 restituisce la sequenza di risposte NO NO NO NO SI, se $x = 7$, l'Algoritmo 2 restituisce la sequenza NO NO SI, mentre l'Algoritmo 1 restituisce la sequenza di risposte NO NO NO NO NO SI, se $x = 8$, l'Algoritmo 2 restituisce la sequenza NO NO NO, mentre l'Algoritmo 1 restituisce la sequenza di risposte NO NO NO NO NO NO. D'altra parte, dire che una risposta ad una domanda è del tipo SI/NO è solo una convenzione, che potremmo rimpiazzare con quella per cui un SI corrisponde ad uno 0, ed un NO corrisponde ad un 1. In questo caso, otterremo che l'Algoritmo 1 definisce una codifica $f : \{1, 2, 3, 4, 5, 6, 7, 8\} \to \{0, 1\}^*$ per cui $f(1) = 0, f(2) = 10, f(3) = 110, f(4) = 1110, f(5) = 11110, f(6) = 111110, f(7) = 1111110, f(8) = 1111111$, mentre l'Algoritmo 2 definisce una codifica $g : \{1, 2, 3, 4, 5, 6, 7, 8\} \to \{0, 1\}^*$ per cui $g(1) = 000, f(2) = 001, g(3) = 010, g(4) = 011, g(5) = 100, g(6) = 101, g(7) = 110, g(8) = 111$.

É altresi evidente che il numero medio di domande che un dato algoritmo $\mathcal{A}$ esegue per scoprire il valore dell'elemento incognito in S è esattamente pari alla lunghezza media $\sum_{i=1}^{n} p_i \ell_i$ della codifica $f : S \to \{0,1\}^*$ definita dall'algoritmo $\mathcal{A}$, dove ℓ_i = lunghezza della codifica $f(i)$ dell'elemento $i \in S$.

Osserviamo ora che le codifiche ottenute dagli Algoritmi 1 e 2 hanno particolari caratteristiche. Ricordiamo innanzitutto le definizione di ordine lessicografico tra sequenze binarie.

Definizione 1 *Date sequenze binarie $\boldsymbol{v} = v_1 \dots v_n, \boldsymbol{w} = w_1 \dots w_m \in \{0,1\}^*$, diremo che $\boldsymbol{v}$ è lessicograficamente inferiore a $\boldsymbol{w}$ (e scriveremo $\boldsymbol{v} \prec \boldsymbol{w}$), se e solo se, detto i il più piccolo intero per cui $v_i \neq w_i$, vale che $v_i < w_i$ (ovvero $v_i = 0$ e $w_i = 1$).*

Ovviamente, un ordine lessicografico tra sequenze può essere definito per qualunque alfabeto su cui le sequenze sono costruite (non necessariamente binario, quindi) purchè sull'alfabeto sia definita una relazione d'ordine totale.

Definizione 2 *Una codifica $\mathsf{c} : \{1, \dots, n\} \to \{0,1\}^*$ è detta* alfabetica *se e solo se vale che per ogni $i, j \in \{1, \dots, n\}$, $i < j \Rightarrow f(i) \prec f(j)$.*

É agevole verificare che sia la codifica $f : \{1,2,3,4,5,6,7,8\} \to \{0,1\}^*$ ottenuta dall'Algoritmo 1 che la codifica $g : \{1,2,3,4,5,6,7,8\} \to \{0,1\}^*$ ottenuta dall'Algoritmo 2 sono alfabetiche. Questo fatto vale in generale.

Lemma 1 *Sia $\mathcal{A}$ un algoritmo per la ricerca in $S = \{1, \dots, n\}$, che procede attraverso la formulazione di domande del tipo "è $x \leq j$?", per opportuni $j \in S$, e sia $\mathsf{c} : S \to \{0,1\}^*$ la codifica ottenuta a partire dall'algoritmo $\mathcal{A}$. Allora, vale che c è una codifica alfabetica degli elementi in S. Inoltre, c è anche una codifica prefisso degli elementi in S (ovvero, per ogni $a, b \in S$, nè $\mathsf{c}(a)$ è prefisso di $\mathsf{c}(b)$, nè $\mathsf{c}(b)$ è prefisso di $\mathsf{c}(a)$).*

Dimostrazione. Siano a, b arbitrari elementi di S, con $a < b$. Sia $\mathsf{c}(a) = z_1 \dots z_n$ la codifica di a e $\mathsf{c}(b) = y_1 \dots y_m$ la codifica di b, ottenute scrivendo le risposte 0/1 che otterremo eseguendo l'algoritmo $\mathcal{A}$, nell'ipotesi che l'elemento incognito x sia uguale ad a e b, rispettivamente.

Sia inoltre ℓ il più piccolo intero per cui $z_\ell \neq y_\ell$. Ciò vuol dire che $\ell =$ è anche l'indice della prima domanda formulata dall'algoritmo $\mathcal{A}$ che riceve risposta *diversa*, a seconda che l'elemento x che si cerca sia uguale ad a o uguale a b.

Poichè $a < b$ siamo in una situazione del genere

$$1 \dots i \dots a \dots k \dots b \dots h \dots n$$

dove con i abbiamo denotato un generico intero minore di a, con h un generico intero maggiore di b e con k un generico intero compreso tra a e b.

Ricordiamo che ℓ è il più piccolo intero per cui $(z_\ell \neq y_\ell)$, ovvero anche l'indice della prima domanda formulata dall'algoritmo $\mathcal{A}$ che riceve risposta *diversa*, a seconda che l'elemento x che si cerca è uguale ad a o uguale a b e sia tale domanda del tipo "è $x \leq \alpha$?, per qualche $\alpha \in \{1, 2, \dots, n\}$. Chiediamoci che valore può essere stato α, per aver provocato una risposta diversa dell'algoritmo $\mathcal{A}$, a seconda che x sia a o b.

- Se $(\alpha = i)$ la domanda "è $x \leq \alpha$? riceverebbe risposta NO in *entrambi* i casi (sia a che b sono $> \alpha$) quindi $\alpha = i$ *non* può accadere
- Se $(\alpha = h)$ la domanda "è $x \leq \alpha$? riceverebbe risposta SI in *entrambi* i casi (sia a che b sono $< \alpha$) quindi $\alpha = h$ *non* può accadere

- Se ($\alpha = k$) la domanda "è $x \leq \alpha$? riceve risposta SI se $x = a$ e riceve risposta NO se $x = b$ (e ciò sappiamo accade) da cui deduciamo che $z_\ell = 0$ e $y_\ell = 1$, ovvero $\mathsf{c}(a) \preceq \mathsf{c}(b)$. Abbiamo quindi provato che la codifica $\mathsf{c} : \{1, \ldots, n\} \to \{0,1\}^*$ è *alfabetica*.

La prova che la codifica c è una codifica prefisso è semplice. Infatti, se c non fosse prefisso, allora per qualche $a, b \in S$, $\mathsf{c}(a) = z_1 \ldots z_n$ sarebbe parte iniziale di $\mathsf{c}(b) = y_1 \ldots y_m$. Detto in altri termini, sia che l'elemento incognito x fosse a o fosse b, l'algoritmo $\mathcal{A}$ fornirebbe le stesse n risposte $z_1 \ldots z_n$ sulle prime n domande, e quindi non sarebbe in grado di decidere se l'elemento incognito x è pari ad a o a b, contro l'ipotesi che $\mathcal{A}$ è un corretto algoritmo per la ricerca in $S = \{1, \ldots, n\}$.

□