Teoria dell'informazione e della trasmissione

Indice

Introduzione **3**

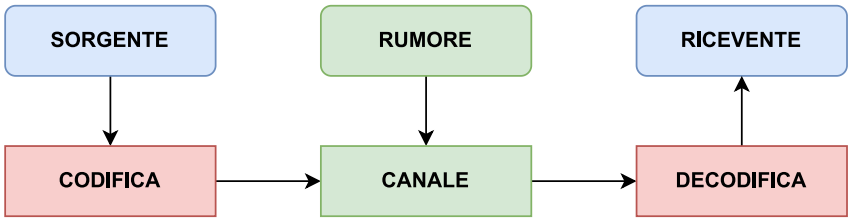
Parte I — Teoria dell'Informazione **4**

1. Codici **5**

2. Disuguaglianza di Kraft **8**

Introduzione

Lo schema di riferimento che andremo ad usare durante tutto il corso è il seguente:



La prima persona che lavorò alla teoria dell'informazione fu **Claude Shannon**, un impiegato della TNT (*Telecom americana*) al quale è stato commissionato un lavoro: massimizzare la quantità di dati che potevano essere trasmessi sul canale minimizzando il numero di errori che potevano accadere durante la trasmissione.

Nel 1948 Shannon pubblica un lavoro intitolato *«A mathematical theory of comunication»*, un risultato molto teorico nel quale modella in maniera astratta il canale e capisce come l'informazione può essere spedita *«meglio»* se rispetta certe caratteristiche. Ci troviamo quindi di fronte ad un risultato che non ci dice cosa fare nel caso specifico o che codice è meglio, ma è probabilistico, ti dice nel caso medio cosa succede.

Questo approccio è sicuramente ottimale, ma rappresenta un problema: va bene il caso medio, ma a me piacerebbe sapere cosa succede nel caso reale. Questo approccio più reale è quello invece seguito dal russo **Kolmogorov**, un accademico che vuole capire cosa succede nei singoli casi senza usare la probabilità. A metà degli anni '60 propone la sua idea di teoria dell'informazione, focalizzandosi quindi sui casi reali e non sui casi medi.

Questi due mostri sacri della teoria dell'informazione, nel nostro schema di lavoro, si posizionano nei rettangoli di sorgente e codifica, mentre la teoria della trasmissione si concentra sui rettangoli sottostanti, quindi nella parte di canale.

Altri due personaggi che hanno lavorato allo stesso problema di Kolmogorov sono due russi, che lavoravano uno in America e uno nell'est del mondo, ma non sono ricordati perché Kolmogorov era il più famoso dei tre.

Un altro personaggio importante è **Richard Hamming**, un ricercatore di Bell Lab che doveva risolvere un problema: i job mandati in esecuzione dalle code batch delle macchine aziendali se si piantavano durante il weekend potevano far perdere un sacco di tempo. La domanda che si poneva Hamming era *«che maroni, perché se le schede forate hanno errori, e le macchine lo sanno, io non posso essere in grado di correggerli?»*

Lui sarà il primo che, per risolvere un problema pratico, costruisce il primo codice di rilevazione e correzione degli errori, il famosissimo **codice di Hamming**.

Vediamo alcune date che rivelano quanto è stata importante la teoria dell'informazione:

- 1965: prima foto di Marte in bianco e nero grande $\approx 240K$ bit, inviata con velocità 6 bit al secondo, ci metteva ore per arrivare a destinazione;
- 1969: stessa foto ma compressa, inviata con velocità $16K$ bit al secondo, ci metteva pochi secondi;
- 1976: prima foto di Marte a colori da parte di Viking;
- 1979: prima foto di Giove e delle sue lune a colori da parte di Voyager;
- anni '80: prima foto di Saturno e delle sue lune da parte di Voyager.

Parte I — Teoria dell'Informazione

1. Codici

Sia X un insieme di **simboli** finito. Un/a **messaggio/parola**

$$\overline{x} = x_1 \cdot ... \cdot x_n \in X^n$$

è una concatenazione di n caratteri x_i di X. Dato $d > 1$, definiamo

$$D = \{0, ..., d-1\}$$

l'insieme delle **parole di codice**. Definiamo infine

$$c : X \longrightarrow D^+$$

la **funzione di codifica**, la quale associa ad ogni carattere di X una parola di codice.

Non useremo tanto D, ci sarà più utile

$$D^+ = \bigcup_{n \geq 1}^{\infty} \{0, ..., d-1\}^n$$

insieme di tutte le parole ottenute concatenando parole di D.

Voglio **massimizzare la compressione**: sia $l_c(x)$ la lunghezza della parola $x \in X$ nel codice c. Ci servirà anche la **probabilità** $p(x)$ di estrarre un simbolo: questo perché alle parole estratte spesso andremo a dare una lunghezza breve, mentre alle parole estratte di rado andremo a dare una lunghezza maggiore. Un codice che segue queste convenzioni è ad esempio il **codice morse**.

Definiamo il **modello sorgente** come la coppia $\langle X, p \rangle$.

Il nostro modello è quasi completo, perché ci interessa la probabilità di ottenere una parola di codice di D^+, non la probabilità dei simboli presenti in D. Definiamo quindi

$$P_n(\overline{x} = x_1 \cdot ... \cdot x_n) = \prod_{i=1}^{n} p(x_i).$$

Qua vediamo una prima enorme semplificazione di Shannon: stiamo assumendo l'**indipendenza** tra più estrazioni consecutive, cosa che nelle lingue parlate ad esempio non è vera.

Dati il modello $\langle X, p \rangle$, la base $d > 1$ e il codice $c : X \longrightarrow D^+$ il modello deve essere tale che

$$\mathbb{E}[l_c] = \sum_{x \in X} l_c(x) p(x)$$

sia minimo.

Il primo problema che incontriamo sta nella fase di decodifica: se codifico due simboli con la stessa parola di codice come faccio in fase di decodifica a capire quale sia il simbolo di partenza?

Definizione 1.1 (*Codice non singolare*)**:** Un codice c è non singolare se c è una funzione iniettiva.

Imponiamo che il codice c sia **non singolare**. Stiamo imponendo quindi che il codominio sia abbastanza capiente da tenere almeno tutti i simboli di X.

Definiamo $C : X^+ \longrightarrow D^+$ l'**estensione del codice** c che permette di codificare una parola intera. Se il codice c è non singolare, cosa possiamo dire di C?

Purtroppo, la non singolarità **non** si trasmette nell'estensione del codice.

Definizione 1.2 (*Codice univocamente decodificabile*): Un codice c è univocamente decodificabile se la sua estensione C è non singolare.

Restringiamo l'insieme dei codici solo a quelli UD. Per dimostrare che un codice è UD esiste l'algoritmo di **Sardinas-Patterson**, che lavora in tempo

$$O(mL) \quad | \quad m = |X| \wedge L = \sum_{x \in X} l_c(x).$$

Con gli UD abbiamo un altro piccolo problema: non sono **stream**. In poche parole, un codice UD non ci garantisce di decodificare istantaneamente una parola di codice in un carattere di X quando mi arrivano un po' di bit, ma dovrei prima ricevere tutta la stringa e poi decodificare. Sono ottimi codici eh, però ogni tanto aspetteremo tutta la codifica prima di poterla decodificare. Eh, ma non va bene: in una stream non posso permettermi tutto ciò, e inoltre, se la codifica è veramente grande, potrei non riuscire a tenerla tutta in memoria.

Una prima soluzione sono i **codici a virgola**, ovvero codici che hanno un carattere di terminazione per dire quando una parola è finita, e quindi risolvere il problema stream negli UD, ma noi faremo altro.

Definizione 1.3 (*Codici istantanei*): Un codice c è istantaneo se ogni parola di codice non è prefissa di altre parole di codice.

I CI hanno la proprietà che stiamo cercando, ovvero permettono una decodifica stream, quindi non dobbiamo aspettare tutta la codifica prima di passare alla decodifica, ma possiamo farla appena riconosciamo una parola di codice. Inoltre, per verificare se un codice è CI devo solo controllare se vale la regola dei prefissi, e questa è molto più veloce rispetto al controllo che dovevo fare negli UD.

Ho quindi la seguente gerarchia:

$$\text{CI} \subset \text{UD} \subset \text{NS} \subset \text{codici} \ .$$

Ritorniamo all'obiettivo di prima: minimizzare la quantità

$$\mathbb{E}[l_c] = \sum_{x \in X} l_c(x) p(x).$$

Vediamo come, mettendo ogni volta dei nuovi vincoli sul codice, questo valore atteso aumenta, visto che vogliamo dei codici con buone proprietà ma questo va sprecato in termini di bit utilizzati.

Teorema 1.1: Se c è un CI allora c è un UD.

Dimostrazione 1.1: Dimostro il contrario, ovvero che se un codice non è UD allora non è CI.

Sia $c : X \longrightarrow D^+$ e sia C la sua estensione. Assumiamo che c sia non singolare. Se c non è UD allora esistono due messaggi x_1 e x_2 diversi che hanno la stessa codifica $C(x_1) = C(x_2)$.

Per mantenere i due messaggi diversi possono succede due cose:

- un messaggio è prefisso dell'altro: se x_1 è formato da x_2 e altri m caratteri, vuol dire che i restanti m caratteri di x_1 devono essere codificati con la parola vuota, che per definizione di codice non è possibile, e, soprattutto, la parola vuota sarebbe prefissa di ogni altra parola di codice, quindi il codice c non è istantaneo;
- esiste almeno una posizione in cui i due messaggi differiscono: sia i la prima posizione dove i due messaggi differiscono, ovvero $x_1[i] \neq x_2[i]$ e $x_1[j] = x_2[j]$ per $1 \leq j \leq i-1$, ma allora $c(x_1[i]) \neq c(x_2[i])$ e $c(x_1[j]) = c(x_2[j])$ perché c è non singolare, quindi per avere la stessa codifica devo tornare al punto 1 e avere x_1 come prefisso di x_2. Come visto poco fa, otteniamo che c non è istantaneo.

Ma allora c non è istantaneo. ■

2. Disuguaglianza di Kraft

Cosa possiamo dire sui CI? Sono molto graditi perché, oltre a identificare un CI in maniera molto semplice guardando i prefissi, possiamo capire **se esiste** un codice istantaneo senza vedere il codice, ovvero guardando solo le lunghezze delle parole di codice.

Questa è una differenza abissale: prima guardavamo le parole di codice e controllavo i prefissi, ora non ho le parole, posso guardare solo le lunghezze delle parole di codice e, in base a queste, posso dire se esiste un CI con quelle lunghezze. Il problema principale è che non so che codice è quello istantaneo con quelle lunghezze, sto solo dicendo che sono sicuro della sua esistenza.

Come mai seguire questa via e non quelle dei prefissi? Se il codice è molto grande, guardare questa proprietà è meno oneroso rispetto al guardare i prefissi.

La proprietà che stiamo blaterando da un po' è detta **disuguaglianza di Kraft**.

Teorema 2.1 (*Disuguaglianza di Kraft*): Dati una sorgente $X = \{x_1, ..., x_m\}$ di m simboli, $d > 1$ base del codice e m interi $l_1, ..., l_m > 0$ che mi rappresentano le lunghezze dei simboli del codice, esiste un CI

$$c : X \longrightarrow D^+$$

tale che

$$l_c(x_i) = l_i \quad \forall i = 1, ..., m$$

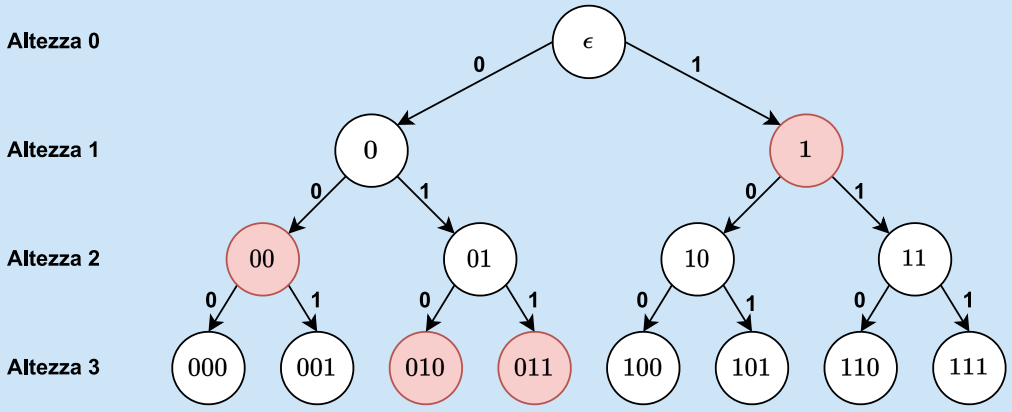
se e solo se

$$\sum_{i=1}^{m} d^{-l_i} \leq 1.$$

Dimostrazione 2.1: Dimostriamo la doppia implicazione.

($\Longrightarrow$) Esiste un CI con quelle proprietà, dimostro che vale la disuguaglianza di Kraft.

Sia c il nostro CI e d la base di c, dobbiamo definire la profondità del nostro codice. Possiamo usare un albero di codifica, ovvero un albero d-ario che rappresenta come sono codificate le varie parole di codice. Vediamo un breve esempio.



Vogliamo sapere

$$l_{\max} = \max_{i=1,\dots,m} l_c(x_i).$$

Costruiamo l'albero di codifica per il nostro CI e mettiamo dentro questo albero le parole del nostro codice. Come lo costruiamo? Creiamo l'albero d-ario alto $l_{\max}$ completo e scegliamo, in questo albero, delle parole ad altezza $l_i \quad \forall i = 1, \dots, m$.

Dividiamo il nostro albero in sotto-alberi grazie alle parole che abbiamo inserito: ogni sotto-albero ha come radice una delle parole che abbiamo scelto. Contiamo quante foglie sono coperte da ogni sotto-albero. Sono tutti alberi disgiunti, visto che non posso avere prefissi essendo c un CI. Il numero massimo di foglie è $d^{l_{\max}}$, ma noi potremmo non coprirle tutte, quindi

$$\sum_{i=1}^{m} |A_i| \leq d^{l_{\max}},$$

con A_i sotto-albero con radice la parola x_i. Notiamo che

$$\sum_{i=1}^{m} d^{l_{\max}-l_i} = \sum_{i=1}^{m} |A_i| \leq d^{l_{\max}}.$$

Ora possiamo dividere tutto per $d^{l_{\max}}$ e ottenere

$$\sum_{i=1}^{m} \frac{d^{l_{\max}-l_i}}{d^{l_{\max}}} = \sum_{i=1}^{m} d^{-l_i} \leq 1.$$

($\Longleftarrow$) Vale la disuguaglianza di Kraft, dimostro che esiste un CI con quelle proprietà.

Abbiamo le lunghezze che rendono vera la disuguaglianza di Kraft, quindi costruiamo l'albero di codifica: scegliamo un nodo ad altezza l_i e poi proibiamo a tutti gli altri nodi ancora da inserire di:

- inserirsi nel sotto-albero di nodi già selezionati;
- scegliere nodi presenti nel cammino da un nodo già selezionato fino alla radice.

Una volta costruito l'albero vedo che esso rappresenta un CI, perché tutti i nodi non hanno nodi nel loro sotto-albero, quindi non ho prefissi per costruzione, quindi questo è un CI. ■