Teoria dell'informazione e della trasmissione

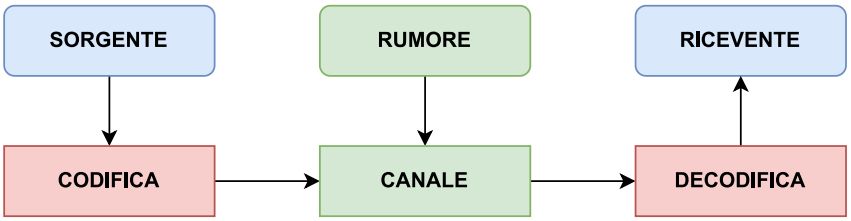
Indice

1. Lezione 01 [24/09]

2. Lezione 02 [01/10]

2.1. Introduzione storica

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 poteva accadere durante la trasmissione.

Nel 1948 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.

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 famoso e usatissimo **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.

2.2. Cosa faremo

Nel corso vedremo due operazioni fondamentali per spedire al meglio un messaggio sul canale:

- **compressione**: dobbiamo ottimizzare l’accesso al canale, ovvero se possiamo mandare meno bit ma questi mi danno le stesse informazioni dei bit totali ben venga;
- **aggiunta di ridondanza**: dobbiamo aggiungere dei bit per permettere il controllo, da parte del ricevente, dell’integrità del messaggio.

La compressione riguarda la **source coding**, e viene rappresentata nel **primo teorema di Shannon**, mentre la ridondanza riguarda la **channel coding**, e viene rappresentata nel **secondo teorema di Shannon**.

Voglio massimizzare l’informazione da spedire sul canale ma alla quale devo aggiungere ridondanza.

Quello che fa Shannon è modellare il canale secondo una matrice stocastica, che quindi permette di sapere in media cosa succede evitando tutti i casi precisi.

IN/OUT	a	b	c	d	e
a	0.7	0.0	0.1	0.1	0.1
b	0.2	0.8	0.0	0.0	0.0
c	0.1	0.0	0.6	0.2	0.1
d	0.0	0.0	0.2	0.5	0.3
e	0.0	0.0	0.0	0.0	1.0

Questa matrice indica, per ogni carattere del nostro alfabeto, quale è la probabilità di spedire tale carattere e ottenere i vari caratteri presenti nell’alfabeto.

2.3. Esempio: informazione di un messaggio

Quando un messaggio contiene più informazioni di un altro?

Lancio n volte due monete, una truccata e una non truccata.

Quale delle due monete mi dà più informazioni? Sicuramente quella non truccata: il lancio della moneta “classica” è un evento randomico, non so mai cosa aspettarmi, mentre il lancio della moneta truccata è prevedibile, so già cosa succederà.

Possiamo quindi dire che un evento prevedibile porta pochissima informazione, mentre un evento imprevedibile porta tantissima informazione.

2.4. Esempio: codice ZIP

Uno dei codici di compressione più famosi è il codice **ZIP**.

Come funziona:

1. creo un dizionario, inizialmente vuoto, che contiene le coppie (stringa-sorgente,codifica), dove “codifica” è un numero;
2. usando un indice che scorre la stringa carattere per carattere, e partendo con numero di codifica uguale a 1, fino all’ultimo carattere eseguo iterativamente:
    1. parti da una stringa vuota che useremo come accumulatore;

2. aggiungi il carattere corrente all'accumulatore;
3. se l'accumulatore non è presente come chiave nel dizionario la aggiungo a quest'ultimo con codifica il numero corrente di codifica; riparto poi dal punto 2.1 con numero di codifica aumentato di 1;
4. se l'accumulatore è presente come chiave nel dizionario riparto dal punto 2.2.

Ad esempio, la stringa $AABACDABDCAABBA$ viene codificata con:

- $A \longrightarrow 1$;
- $A \longrightarrow AB \longrightarrow 2$;
- $A \longrightarrow AC \longrightarrow 3$;
- $D \longrightarrow 4$;
- $A \longrightarrow AB \longrightarrow ABD \longrightarrow 5$;
- $C \longrightarrow 6$;
- $A \longrightarrow AA \longrightarrow 7$;
- $B \longrightarrow 8$;
- $B \longrightarrow BA \longrightarrow 9$.

Prima avevo 15 caratteri, ora ne uso 9, letsgosky.

2.5. Richiami matematici

In questa parte abbiamo rivisto la definizione di monoide, gruppo, anello e campo, oltre alla definizione di generatore.

Abbiamo anche visto i **campi di Galois**, utilissimi per rendere il nostro calcolatore un campo algebrico, ovvero il campo $\mathbb{GF}(2^{64})$ dei polinomi di grado massimo 63 con coefficienti sul campo $\mathbb{Z}_2$.

3. Lezione 03 [04/10]

Dalla lezione scorsa abbiamo capito che dobbiamo fare due cose:

1. massimizzare l'informazione trasmessa ad ogni utilizzo del canale; il "ogni utilizzo" è importante perché ogni volta che utilizzo il canale devo essere top, adesso lo devo essere. Se devo mandare n informazioni lo posso fare in n accessi oppure in 1 accesso, in entrambi i casi massimizzo l'informazione trasmessa ma non uso sempre tutta la banda possibile;
2. minimizzare il numero di errori di trasmissione.

Shannon userà la tecnica del Divide Et Impera, ovvero risolverà le due task separatamente e poi unirà i due risultati parziali. Per puro culo, questa soluzione sarà ottima (*non sempre succede*).

Il punto 1 riguarda la sorgente, il punto 2 riguarda la codifica e il canale.

3.1. Codici

Prima di vedere un po' di teoria dei codici diamo alcune basi matematiche.

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 $D = \{0, ..., d-1\}$ insieme delle **parole di codice**.

Definiamo infine $c : X \longrightarrow D^+$ **funzione di codifica** che associa ad ogni carattere di X una parola di codice. Questa è una codifica in astratto: a prescindere da come è formato l'insieme X io uso le parole di codice che più mi piacciono (????). L'insieme D^+ è tale che

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

Voglio massimizzare la compressione: sia $l_c(x)$ la lunghezza della parola $x \in X$ nel codice c.

Quello che ho ora mi basta? NO: mi serve anche la **probabilità** $p(x)$ di estrarre un simbolo. Questo perché alle parole estratte molto 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 il codice morse.

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

Ora va bene? NON ANCORA: noi non vogliamo la probabilità di estrarre un simbolo perché noi lavoriamo con le parole, non con i simboli.

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 invece non è vera.

Il codice ZIP invece non assume l'indipendenza e lavora con la lingua che sta cercando di comprimere.

Dati il modello $\langle X, p \rangle$ e la base $d > 1$, dato 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?

Imponiamo che il codice sia un **codice non singolare**, ovvero che la funzione c sia iniettiva. 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 c è non singolare, come è C?

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

Andiamo a restringere l'insieme dei codici "buoni": chiamiamo codici **univocamente decodificabili** (*UD*) i codici che hanno C non singolare.

Per dimostrare che un codice è UD esista l'algoritmo di Sardinas-Patterson, che lavora in tempo $O(mL)$, con $m = |X|$ e $L = \sum_{x \in X} l_c(x)$.

Ora però abbiamo un altro problema: gli UD 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.

Potremmo utilizzare 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.

Restringiamo per l'ultima volta, definendo i **codici istantanei** (*CI*). Questi codici 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.

Per capire se un codice è CI devo guardare i prefissi: nessuna parola di codice deve essere prefissa di un'altra. Questo controllo è molto più veloce rispetto al controllo che dovevo fare nei codici 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. Inoltre, il codice che abbiamo sotto mano è il migliore?

A noi gli UD andavano bene (*stream esclusi*), quanto pago per andare nei codici istantanei?

Teorema: Se un codice è istantaneo allora è anche univocamente decodificabile.

Dimostrazione: 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 (o viceversa), ma, quindi otteniamo che c non è istantaneo.

Ma allora il codice non è istantaneo. ■