Complementi di Algoritmi

Massimo Perego

11 febbraio 2026[1]

[1]Questi appunti sono basati esclusivamente sulle dispense fornite dal professor Cesa-Bianchi (disponibili `a questo indirizzo`, aggiornate alla data della scrittura), unite a chiarimenti scritti da me (o almeno, ciò che io penso siano chiarimenti rispetto al materiale originale). Con *basati* si intende *copiati parola per parola, poi estesi*.

Indice

Capitolo 1

Complessità Computazionale

1.1 Riducibilità polinomiale tra problemi di decisione

Un **problema di decisione** X su $\{0,1\}$ è una coppia $(\mathcal{I}, q)$ dove

- $\mathcal{I} \subseteq \{0,1\}^*$ è l'insieme delle descrizioni (come stringhe binarie) delle **istanze valide del problema**
- $q : \mathcal{I} \to \{0,1\}$ è la **funzione di decisione**, che associa ogni istanza $I \in \mathcal{I}$ al suo valore, ovvero $q(I) = 1$ se positiva o $q(I) = 0$ se negativa

Un algoritmo A risolve un problema $X = (\mathcal{I}, q)$ se restituisce il valore corretto della funzione q su ogni istanza $I \in \mathcal{I}$.

Indichiamo con $|I|$ la **lunghezza della codifica** in bit di un'istanza $I \in \mathcal{I}$. In genere, non specifichiamo il modo con il quale un'istanza viene rappresentata come una stringa di bit, ma resta inteso che le istanze vengano rappresentate secondo rappresentazioni "ragionevoli" (e.g. lista di adiacenza per un grafo).

Se A è un algoritmo che risolve il problema $X = (\mathcal{I}, q)$, indichiamo con $T_A(I)$ il tempo di calcolo di A quando l'input è $I \in \mathcal{I}$. Definiamo ora

$$T_A(n) = \max \{T_A(I) \mid |I| \leq n\}$$

ovvero il massimo dei tempi di calcolo per istanze $I \in \mathcal{I}$ di lunghezza al più n. Diciamo che A risolve X in tempo polinomiale se esiste un intero k tale che $T_A(n) = \mathcal{O}\left(n^k\right)$.

Un algoritmo ha accesso a un **oracolo** per un problema $X = (\mathcal{I}, q)$ quando può ottenere in tempo unitario il valore $q(I)$ per istanze arbitrarie $I \in \mathcal{I}$.

Un problema Y è **polinomialmente riducibile** a un altro problema X, denotato con $Y \preceq_{\mathrm{p}} X$ quando

1. Esiste un algoritmo che risolve Y in tempo polinomiale con accesso a un oracolo per X
2. L'oracolo viene interrogato una sola volta e l'algoritmo termina fornendo la risposta dell'oracolo

Questa nozione di riducibilità polinomiale è anche nota come *riducibilità secondo Karp*. Esiste anche una nozione più generale, nota come riducibilità secondo Turing, dove l'algoritmo può accedere all'oracolo un numero polinomiale di volte.

Equivalentemente, la riducibilità polinomiale (secondo Karp) può essere vista come "l'input di Y può essere trasformato in tempo polinomiale in input di X" (per poi risolvere il problema in tempo unitario grazie all'oracolo, coincide con la prima definizione).

Se $Y \preceq_{\mathrm{p}} X$ ed esiste un algoritmo che risolve X in tempo polinomiale, allora esiste un algoritmo che risolve Y in tempo polinomiale (in quanto la polinomialità è robusta rispetto alla composizione). Viceversa, se $Y \preceq_{\mathrm{p}} X$ e non esiste un algoritmo che risolve Y in tempo polinomiale, allora non esiste un algoritmo che risolve X in tempo polinomiale. Intuitivamente, $Y \preceq_{\mathrm{p}} X$ significa che il problema Y non è *più difficile* del problema X. Se $X \preceq_{\mathrm{p}} Y$ e $Y \preceq_{\mathrm{p}} X$, ovvero X e Y sono polinomialmente riducibili l'uno con l'altro, allora scriviamo $X \equiv_{\mathrm{p}} Y$.

1.1.1 Esempi di riducibilità tra problemi

Sia $G = (V, E)$ un grafo semplice (non diretto, non pesato, senza loop e archi multipli). Un **insieme indipendente** (Independent Set) in G è un sottoinsieme $S \subseteq V$ di vertici non adiacenti, ovvero $\forall i, j \in S$, $(i, j) \notin E$. Una **copertura** (Vertex Cover) di G è un sottoinsieme $S' \subseteq V$ di vertici tali che ogni arco di G ha almeno un estremo in S', ovvero $\forall (i, j) \in E$ esiste

$k \in S'$ tale che $k = i$ o $k = j$.

Fatto 1.1.1. *Sia $G = (V, E)$ un grafo semplice. Allora S è un insieme indipendente (soluzione per Independent Set) se e solo se $S' = V \setminus S$ è una copertura (soluzione per Vertex Cover).*

Dimostrazione. Sia S un insieme indipendente e sia $S' = V \setminus S$. Allora dato un qualsiasi $(i, j) \in E$, deve valere $i \notin S$ oppure $j \notin S$. Quindi $i \in S'$ oppure $j \in S'$, da cui ne segue che S' è una copertura di G.

Viceversa, sia S' una copertura di G e $S = V \setminus S'$. Allora dati $i, j \in S$ arbitrari $(i, j) \notin E$, altrimenti allora S' non sarebbe una copertura; da questo otteniamo che S è un Independent Set. □

Sia INDEPENDENT SET il problema di decisione le cui istanze sono coppie $I = (G, k)$ dove G è un grafo semplice e k è un intero. La funzione di decisione q_{IS} è tale che $q_{IS}(I) = 1$ se e solo se G contiene un insieme indipendente di taglia almeno k.

Le istanze del problema VERTEX COVER sono le stesse di Independent Set, ma la funzione di decisione q_{VC} è tale che $q_{VC}(I) = 1$ se e solo se G contiene una copertura di taglia al più k.

Conseguenza immediata del Fatto 1.1.1 è che Independent Set e Vertex Cover sono polinomialmente riducibili l'uno con l'altro.

Corollario 1.1.1. *Independent Set $\equiv_p$ Vertex Cover*

La relazione $\equiv_{\mathrm{p}}$ gode di transitività.

Fatto 1.1.2. *Se $Z \preceq_p Y$ e $Y \preceq_p X$, allora $Z \preceq_p X$.*

Dimostrazione. Dato un oracolo per X possiamo risolvere un'istanza di Z nel modo seguente: eseguiamo l'algoritmo per risolvere Z usando l'oracolo per Y, ma ogni volta che dovrebbe essere invocato l'oracolo di Y, questo viene simulato eseguendo l'algoritmo per risolvere Y, il quale usa l'oracolo per X (ancora una volta, la polinomialità è robusta rispetto alla composizione). □

Introduciamo ora il problema di decisione SET COVER: le istanze sono della forma $I = (U, \mathcal{S}, k)$ dove U è un insieme finito, $\mathcal{S} \equiv \{S_1, \ldots, S_m\}$ è una collezione di sottoinsiemi di U e k è un intero. Allora $q_{SC}(I) = 1$ se e solo

se esistono al più k sottoinsiemi in $\mathcal{S}$ tali che la loro unione sia tutto U.

Fatto 1.1.3. *Vertex Cover* $\preceq_p$ *Set Cover*

Dimostrazione. Supponiamo di avere un oracolo per Set Cover e definiamo un algoritmo polinomiale per Vertex Cover.

Data un'istanza $I = (G, k)$ di Vertex Cover con $G = (V, E)$ si può costruire un'istanza I' di Set Cover dove $U \equiv E$ e $\mathcal{S} \equiv \{S_i \mid i \in V\}$, dove S_i è l'insieme degli archi di G incidenti sul vertice i (ottenibile in tempo lineare partendo dalla descrizione di G). Quindi ogni $u \in U$ è contenuto in esattamente due elementi di $\mathcal{S}$ (un lato ha due estremità).

Verifichiamo ora che U è l'unione di al più k insiemi $S_1, \ldots, S_{|V|}$ se e solo se G ha una copertura con vertici di taglia al più k.

Supponiamo che l'unione di $S_{i_1}, \ldots, S_{i_r}$ sia U, con $r \leq k$. Allora, ogni arco in G è incidente a uno dei vertici $i_1, \ldots, i_r$. Quindi $\{i_1, \ldots, i_r\}$ è una copertura con vertici di taglia al più k. Viceversa, se $\{i_1, \ldots, i_r\}$ è una copertura con vertici di taglia al più k allora la gli insiemi corrispondenti $S_{i_1}, \ldots, S_{i_r}$ hanno U come unione.

Quindi possiamo definire un algoritmo che implementa la funzione di decisione q_{VC} per Vertex Cover nel modo seguente: data un'istanza di I Vertex Cover, l'algoritmo costruisce in tempo polinomiale un'istanza I' di Set Cover nel modo descritto sopra. Quindi chiama l'oracolo un'unica volta per ottenere $q_{SC}(I')$ che viene restituita in output. □

L'universo di Set Cover corrisponde agli archi da coprire, ogni insieme corrisponde agli archi che incidono su un nodo, quindi selezionare uno degli insiemi nella collezione corrisponde a scegliere un nodo come parte della copertura.

Così come possiamo vedere Set Cover come una generalizzazione di Vertex Cover, introduciamo ora Set Packing come generalizzazione di Independent Set.

Le istanze di Set Packing sono le stesse di Set Cover, ovvero $I = (U, \mathcal{S}, k)$ dove U è un insieme finito, $\mathcal{S} \equiv \{S_1, \ldots, S_m\}$ è una collezione di sottoinsiemi di U e k è un intero. la funzione di decisione q_{SP} è tale che $q_{SP} = 1$ se e solo se esistono almeno k sottoinsiemi in $\mathcal{S}$ i quali sono a due a due disgiunti. In altre parole, nessun elemento dell'universo compare in più di uno degli insiemi scelti.

Anche se Set Packing è apparentemente più generale di Independent Set, si può dimostrare che i due problemi sono equivalenti.

Fatto 1.1.4. *Independent Set* $\equiv_p$ *Set Packing.*

Dimostrazione. Data un'istanza $\mathcal{S} \equiv \{S_1, \ldots, S_m\}$ di Set Packing costruiamo in tempo polinomiale un grafo $G = (V, E)$ dove $V \equiv \{v_S \mid S \in \mathcal{S}\}$ e $(v_S, v_T) \in E$ se e solo se $S \cap T \not\equiv \emptyset$ (un grafo con un vertice per ogni insieme del problema originale, ed è presente un arco tra due vertici solo se i relativi insiemi non sono disgiunti, ovvero se hanno elementi in comune). Allora ogni insieme indipendente in G corrisponde a un packing della stessa taglia.

Viceversa, dato un grafo $G = (V, E)$ possiamo costruire in tempo polinomiale la collezione $\mathcal{S} \equiv \{S_i \mid i \in V\}$ dove S_i è l'insieme degli archi di G incidenti su i. Allora ogni packing in $\mathcal{S}$ corrisponde a un insieme indipendente in G della stessa taglia. □

In molte discipline bisogna spesso risolvere problemi di ottimizzazione combinatoria vincolata. In questi problemi si cerca un assegnamento di valori per un insieme di variabili discrete in modo da soddisfare un dato insieme di vincoli. In astratto, problemi di questo tipo sono formulati come problemi di soddisfacibilità su variabili booleane.

Sia $\mathcal{X}$ un insieme di variabili booleane $x_1, \ldots, x_n$. Un assegnamento di valori di verità su $\mathcal{X}$ è una funzione $\pi : \mathcal{X} \to \{0, 1\}$. Un letterale ℓ_i è la variabile x_i o la sua negazione $\bar{x}_i$. Una clausola $C = \ell_{i_1} \vee \cdots \vee \ell_{i_k}$ è una disgiunzione di letterali. Un assegnamento π soddisfa una clausola C se e solo se c'è almeno un letterale della forma $\ell_i = x_i$ e $\pi(x_i) = 1$ oppure $\ell_j = \bar{x}_j$ e $\pi(x_j) = 0$ (in altre parole, `una formula in CNF`).

Sia **SAT** il problema di decisione le cui istanze I sono insiemi di clausole $\mathcal{C}$ su un insieme $\mathcal{X}$ di variabili booleane. Allora $q(I) = 1$ se e solo se esiste un assegnamento $\pi : \mathcal{X} \to \{0, 1\}$ tale che soddisfi tutte le clausole in $\mathcal{C}$ (in realtà sarebbe CNFSAT, SAT è la versione su formule generiche, ma ogni formula è riducibile a una CNF equisoddisfacibile, ovvero soddisfacibile se e solo se lo era anche la formula originaria, in tempo polinomiale, senza però mantenere l'equivalenza semantica).

Una versione ridotta di SAT è **3-SAT**, le cui istanze sono insiemi di clausole ciascuna contenente esattamente 3 letterali. Esempio di istanza di 3-SAT:

- $\mathcal{X} \equiv \{x_1, x_2, x_3, x_4\}$
- $\mathcal{C} \equiv \{(\bar{x}_1 \vee x_2 \vee \bar{x}_3), (\bar{x}_2 \vee x_3 \vee x_4), (x_1 \vee x_2 \vee \bar{x}_4)\}$

Un assegnamento che soddisfa tutte le clausole è $\pi(x_i) = 0$ per $i = 1, \ldots, 4$.

Teorema 1.1.1. *3-SAT $\preceq_p$ Independent Set.*

Dimostrazione. Una definizione equivalente di 3-SAT: $\mathcal{C}$ è soddisfacibile se e solo se è possibile scegliere esattamente un letterale in ciascuna clausola in modo tale che tra i letterali scelti non compaiano simultaneamente x_i e $\bar{x}_i$ per nessuna delle variabili $x_i \in \mathcal{X}$.

Data un'istanza $I = \mathcal{C}$ di 3-SAT, costruiamo un'istanza $I' = (G, |\mathcal{C}|)$ di Independent Set tale che $q_{3SAT}(I) = q_{IS}(I')$. Sia $k = |\mathcal{C}|$. Il grafo $G = (V, E)$ ha $3k$ vertici, uno per ogni letterale in $\mathcal{C}$. I tre vertici corrispondenti ai letterali di ciascuna clausola formano una clique (ognuno è connesso agli altri due, per fare in modo che un solo letterale per clausola venga scelto). Inoltre, per ogni $x_i \in \mathcal{X}$, se x_i e $\bar{x}_i$ sono letterali in clausole distinte allora c'è un arco in G fra i vertici corrispondenti (per fare in modo che solo un letterale o la sua negazione venga scelto). Si noti che possiamo costruire G a partire da $\mathcal{C}$ in tempo polinomiale rispetto a $|\mathcal{C}|$.

Per costruzione di G, l'unico modo per ottenere un insieme indipendente di taglia k nel grafo G è quello di scegliere un nodo in ciascuna delle k clique in modo che non ci siano archi fra il nodo scelto in una clique e nodi scelti appartenenti a un'altra clique. Per costruzione, questo avviene se e solo se è possibile scegliere esattamente un letterale in ciascuna clausola in modo tale che tra i letterali scelti non compaiano simultaneamente x_i e $\bar{x}_i$ per nessuna delle variabili $x_i \in \mathcal{X}$.

Possiamo quindi costruire un algoritmo che risolve 3-SAT in tempo polinomiale usando un oracolo per Independent Set: l'algoritmo riceve un'istanza I di 3-SAT, costruisce l'istanza corrispondente I' di Independent Set e accede all'oracolo ottenendo $q_{IS}(I')$, per poi restituire $q_{IS}(I')$. □

1.2 Classi $\mathcal{P}$ e $\mathcal{NP}$

Sia $\mathcal{P}$ la classe dei problemi di decisione X risolvibili in tempo polinomiale. Ovvero, per ogni problema $X \in \mathcal{P}$ esiste un algoritmo che lo risolve in tempo polinomiale rispetto alla lunghezza delle istanze.

La funzione di decisione q di un problema X caratterizza una determinata proprietà delle sue istanze (per esempio, quei grafi che contengono un insieme indipendente abbastanza grande). Un **certificatore polinomiale** per X è un algoritmo $B : \mathcal{I} \times \{0,1\}^* \to \{0,1\}$ tale che

1. Esiste un polinomio $p(\cdot)$ tale che, per ogni istanza $I \in \mathcal{I}$, $q(I) = 1$ se e solo se esiste una stringa $z \in \{0,1\}^{p(|I|)}$ tale che $B(I,z) = 1$
2. B termina in tempo polinomiale in $|I|$ e $|z|$

Possiamo pensare alla stringa z come un certificato del fatto che $q(I) = 1$. Per esempio, nel problema Independent Set la stringa z denota il sottoinsieme di vertici che costituisce un insieme indipendente di cardinalità almeno k. Nel problema SAT, la stringa z denota un assegnamento che soddisfa tutte le clausole. Si può vedere come dire che "è possibile verificare una soluzione in tempo polinomiale".

L'accesso a un certificatore polinomiale permette di verificare rapidamente se una stringa z è un certificato valido per un'istanza I di un problema. D'altra parte, se volessimo usare il certificatore per trovare un certificato qualora esso esista, ovvero stabilire il valore di $q(I)$, saremmo obbligati a eseguire $B(I,z)$ esaustivamente su tutte le $2^{p(|I|)}$ stringhe z tali che $|z| \leq p(|I|)$.

Introduciamo ora la classe $\mathcal{NP}$ di problemi di decisione X che posseggono un certificatore polinomiale.

Si noti che $\mathcal{P} \subseteq \mathcal{NP}$. Infatti, se $X \in \mathcal{P}$ allora esiste un algoritmo che calcola la funzione di decisione q in tempo polinomiale in $|I|$. Possiamo usare questo algoritmo per implementare un certificatore polinomiale B come segue: dati $I, z \in \mathcal{I} \times \{0,1\}^*$, B restituisce $q(I)$ ignorando z.

Quindi se $q(I) = 1$, abbiamo che $B(I,z) = 1$ per ogni $z \in \{0,1\}^*$ (in particolare per z limitati in lunghezza da un polinomio in $|I|$). Invece, se $q(I) = 0$, allora $B(I,z) = 0$ per ogni $z \in \{0,1\}^*$.

Anche se risolvere un'istanza di un problema in $\mathcal{P}$ non significa necessariamente trovare un certificato z, possiamo comunque interpretare $\mathcal{NP}$ come la classe che contiene quei problemi la cui soluzione (o certificato) è *verificabile* in tempo polinomiale, in contrasto con i problemi in $\mathcal{P}$, i quali problemi sono *risolvibili* in tempo polinomiale. $\mathcal{P} \equiv \mathcal{NP}$ è un problema aperto.

1.3 Problemi $\mathcal{NP}$-completi

Un problema di decisione X è $\mathcal{NP}$-completo se $x \in \mathcal{NP}$ e per qualunque $Y \in \mathcal{NP}$ vale $Y \preceq_{\mathrm{p}} X$; ogni problema in $\mathcal{NP}$ è polinomialmente riducibile a un problema $\mathcal{NP}$-completo. Intuitivamente, i problemi di decisione $\mathcal{NP}$-completi sono i "più difficili" in $\mathcal{NP}$, infatti vale il seguente.

Fatto 1.3.1. *Sia X un qualunque problema $\mathcal{NP}$-completo. Allora $X \in \mathcal{P}$ se e solo $P \equiv \mathcal{NP}$.*

Dimostrazione. Se $\mathcal{P} \equiv \mathcal{NP}$, allora $X \in \mathcal{P}$. D'altra parte, se X è $\mathcal{NP}$-completo, allora dato un qualunque $Y \in \mathcal{NP}$ vale $Y \preceq_{\mathrm{p}} X$. Per ipotesi, X è risolvibile in tempo polinomiale, di conseguenza lo è Y, quindi $Y \in \mathcal{P}$, da cui otteniamo $\mathcal{NP} \subseteq \mathcal{P}$. Abbiamo già dimostrato come $\mathcal{P} \subseteq \mathcal{NP}$, quindi $\mathcal{P} \equiv \mathcal{NP}$. □

Non è chiaro che esistano problemi $\mathcal{NP}$-completi. Potrebbero esserci due problemi $X', X'' \in \mathcal{NP}$ tali che non esiste nessun altro $X \in \mathcal{NP}$ tale che $X' \preceq_{\mathrm{p}} X$ e $X'' \preceq_{\mathrm{p}} X$. Oppure, potrebbe esserci una sequenza infinita di problemi in $\mathcal{NP}$ del tipo $X_1 \preceq_{\mathrm{p}} X_2 \preceq_{\mathrm{p}} \dots$ tale che non esista un problema più difficile degli altri.

Un **circuito** è un grafo diretto aciclico (senza loop, archi multipli e pesi) avente un unico nodo senza archi uscenti chiamato nodo di output. I nodi senza archi entranti possono avere un valore di verità pre-assegnato. I nodi senza archi entranti e senza valori preassegnati sono chiamati nodi di input. I nodi rimanenti hanno uno o due archi entranti e sono etichettati da un operatore booleano $\wedge$, $\vee$, $\neg$ in modo tale che nodi $\wedge$ e $\vee$ abbiano esattamente due archi entranti e nodi $\neg$ abbiamo esattamente un arco entrante.

Un circuito calcola una funzione booleana $f : \{0,1\}^n \to \{0,1\}$ dove n è il numero dei nodi di input. Dato un assegnamento $(b_1, \dots, b_n) \in \{0,1\}^n$ di valori di verità ai nodi di input, calcoliamo $f(b_1, \dots, b_n)$ come il valore di verità del nodo di output ottenuto valutando in cascata i valori di verità di ciascun nodo. La valutazione di un nodo avviene applicando l'operatore logico che lo etichetta ai valori di verità dei nodi all'altro capo degli archi entranti. Se la valutazione di ogni nodo avviene in tempo costante, allora l'intero circuito viene valutato in tempo lineare nel numero dei nodi.

Sostanzialmente, si tratta del parsing tree di una funzione booleana.

Il problema di decisione Circuit Satisfiability (CS) ha istanze che rappresentano circuiti. La funzione di decisione q è tale che $q(I) = 1$ se e solo se esiste un assegnamento ai nodi di input del circuito I tale che il nodo di output assume valore 1. In altre parole, $q(I) = 1$ se e solo se la funzione f calcolata dal circuito è tale che $f(b_1, \ldots, b_n) = 1$ per un qualche $(b_1, \ldots, b_n) \in \{0,1\}^n$.

Teorema 1.3.1 (Cook-Levin)**.** *CS è $\mathcal{NP}$-completo.*

Dimostrazione. La dimostrazione completa è omessa in quanto lunga e tecnicamente complessa. Si basa sul fatto che ogni algoritmo con un tempo di esecuzione polinomiale nella lunghezza dell'input può essere implementato da una famiglia di circuiti $C_1, C_2, \ldots$ dove C_n ha n nodi di input e un numero polinomiale di altri nodi. Per simulare l'algoritmo su un input I di lunghezza n, calcoliamo l'output del circuito C_n con input I.

Per dimostrare che un problema $X \in \mathcal{NP}$ è polinomialmente riducibile a CS consideriamo un certificatore polinomiale B per X, il quale esiste in quanto $X \in \mathcal{NP}$. Costruiamo la famiglia di circuiti $C'_1, C'_2, \ldots$ tale che C'_n ha $n + p(n)$ nodi senza archi entranti e un numero polinomiale in n di altri nodi, dove $p(\cdot)$ è il polinomio che limita la lunghezza dei certificati di B. Per ogni n, C'_n simula B su istanze I di lunghezza n. Data un'istanza $I \in \mathcal{I}$ di lunghezza n, sia $C'_n(I, \cdot)$ il circuito C'_n dove i valori dei primi n nodi senza archi entranti sono preassegnati ai bit di I, mentre i rimanenti $p(n)$ nodi sono di input. Allora $C'_n(I, \cdot)$ simula il certificatore polinomiale $B(I, \cdot)$. Ovvero, $C'_n(I, \cdot)$ è soddisfacibile se e solo se esiste $z \in \{0,1\}^*$ con $|z| \leq p(|I|)$ tale che $B(I, z) = 1$. □

Dimostriamo ora un caso particolare del teorema di Cook-Levin, ovvero che un particolare problema di decisione è polinomialmente riducibile a CS. Il problema è 2-IS, con istanze I che rappresentano grafi semplici, mentre la funzione di decisione q è tale che $q(I) = 1$ se e solo se il grafo I contiene un insieme indipendente di taglia almeno 2.

Teorema 1.3.2. *2-IS $\preceq_p$ CS.*

Dimostrazione. Sia $G = (V, E)$ un grafo su n vertici. Gli archi del grafo G possono essere codificati con una stringa $I_G \in \{0,1\}^N$ dove $N = \binom{n}{2}$, ogni bit rappresenta una coppia di vertici e un arco fra una coppia di vertici è

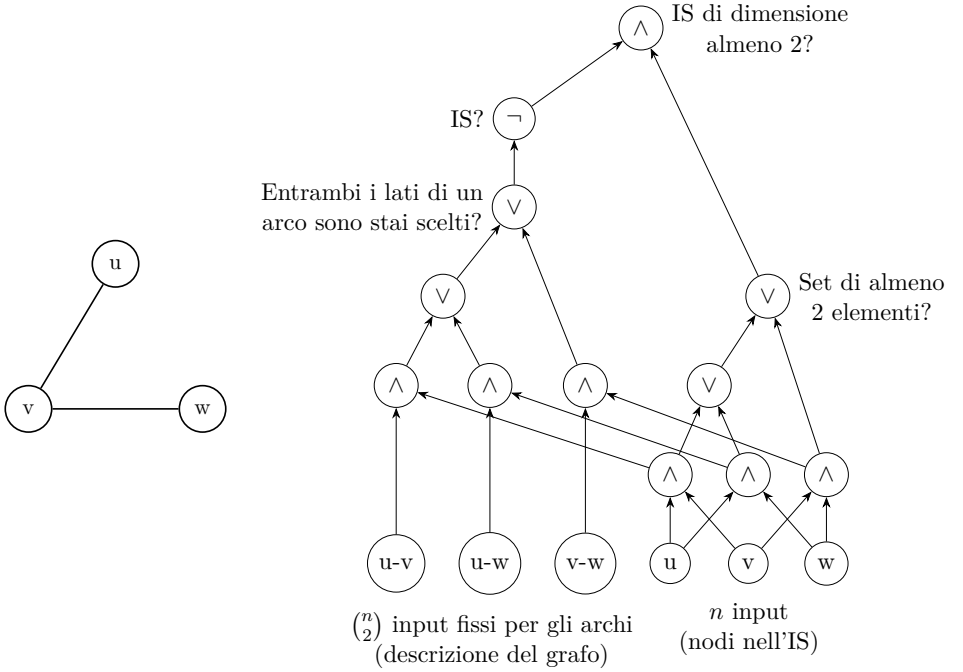


Figura 1.1: Riduzione da 2-IS a CS

codificato ponendo a 1 il bit corrispondente. Indichiamo con B il certificatore polinomiale per 2-IS, il quale esiste in quanto 2-IS $\in \mathcal{NP}$.

Dimostriamo ora come costruire in tempo polinomiale nella descrizione di G un circuito C con $\binom{n}{2} + n$ nodi senza archi entranti tale che $C(I_G, \cdot)$ è soddisfacibile se e solo se esiste una stringa z di lunghezza n tale che $B(I_G, z) = 1$. Questo completa la riduzione, in quanto per calcolare la funzione di decisione di 2-IS su un'istanza I_G è sufficiente costruire C, invocare l'oracolo per CS e produrre in output la risposta dell'oracolo.

Notiamo che i certificati per 2-IS possono essere codificati con una stringa $z \in \{0,1\}^n$ che ha almeno due occorrenze di 1 nelle posizioni corrispondenti ai vertici che formano un insieme indipendente nel grafo.

Costruiamo quindi un circuito con $\binom{n}{2} + n$ nodi di input tale che i primi $\binom{n}{2}$ vengono utilizzati per codificare I_G e i successivi n nodi vengono utilizzati per codificare un certificato z. A questo punto, usiamo $2\binom{n}{2} - 1$ per verificare che z contenga almeno due occorrenze di 1 e usiamo $2\binom{n}{2}$ nodi interni per verificare che non ci sia un arco fra i nodi scelti dal certificato. □

In Figura 1.1 si può vedere un esempio:

- Al di sopra dei nodi di input per z, ogni nodo $\wedge$ chiede "sono stati scelti questi due nodi?"; la restante parte destra dell'albero controlla solamente che siano stati scelti almeno due nodi
- Il risultato della "scelta" viene messo in $\wedge$ con la codifica degli archi presenti
- I risultati vengono messi in $\vee$, quindi risulta 1 se e solo se sono stati scelti due nodi che hanno un arco presente tra loro; viene invertito per avere 1 nel caso di Independent Set
- La radice si occupa di controllare che l'input rappresenti un Independent set (parte sinistra) di dimensione almeno 2 (parte destra)

Una volta che abbiamo stabilito che un certo problema è $\mathcal{NP}$-completo, possiamo trovarne molti altri usando la seguente osservazione.

Fatto 1.3.2. *Se Y è $\mathcal{NP}$-completo e $X \in \mathcal{NP}$ è tale che $Y \preceq_p X$, allora anche X è $\mathcal{NP}$-completo.*

Dimostrazione. Sia $Z \in \mathcal{NP}$ qualunque. Allora $Z \preceq_{\mathrm{p}} Y$. Ma dato che $Y \preceq_{\mathrm{p}} X$, allora $Z \preceq_{\mathrm{p}} X$, il che implica che X è $\mathcal{NP}$-completo. □

Possiamo subito applicare questa osservazione dimostrando quanto segue.

Teorema 1.3.3. *CS $\preceq_p$ 3-SAT.*

Dimostrazione. Omessa. □

Dato che CS è $\mathcal{NP}$-completo, ne segue che anche 3-SAT è $\mathcal{NP}$-completo. Ricordando inoltre che 3-SAT $\preceq_{\mathrm{p}}$ Independent Set $\preceq_{\mathrm{p}}$ Vertex Cover $\preceq_{\mathrm{p}}$ Set Cover, ne deduciamo che tutti questi problemi sono $\mathcal{NP}$-completi. Al contrario di 3-SAT che è $\mathcal{NP}$-completo, 2-SAT (ovvero usando clausole da esattamente due letterali) è un problema risolvibile in tempo lineare.

L'idea dietro la dimostrazione che 2-SAT $\in \mathcal{P}$ è: ogni clausola è della forma $a \vee b$, questa si può vedere come un'implicazione $\neg a \to b$. Costruiamo un grafo orientato con due nodi per ogni letterale, uno per il letterale stesso e uno per la negazione. Per ogni clausola si aggiungono due archi che rappresentano $\neg a \to b$ e $\neg b \to a$. La formula non è soddisfacibile se e solo se, per almeno una variabile, si ha un percorso da x a $\neg x$ e da $\neg x$ a x; questa

condizione è verificabile in tempo lineare, controllando, per ogni variabile x, che x e la sua negazione non siano presenti nella stessa componente fortemente connessa (ad esempio `algoritmo di Tarjan`, lineare in $O(N + M)$, dove N è il numero di nodi, 2 volte i letterali, e M è il numero di archi, ovvero 2 volte il numero di clausole). *[Questa "dimostrazione" non è parte del corso]*

Si noti che ogni formula booleana può essere equivalentemente rappresentata come un circuito (ovvero un'istanza di CS) o come una formula CNF (ovvero un'istanza di SAT) in modo che tali istanze abbiano lunghezza polinomiale nella lunghezza della formula. Inoltre, ogni istanza I di SAT può essere rappresentata come un'istanza di 3-SAT di lunghezza polinomiale in $|I|$ (queste dimostrazioni sono il processo per la dimostrazione omessa precedentemente).

Al contrario, non è sempre possibile rappresentare una formula booleana in DNF in modo che la DNF abbia lunghezza polinomiale nella lunghezza della formula. Se questo fosse possibile avremmo che $\mathcal{P} \equiv \mathcal{NP}$. La soddisfacibilità di una DNF è decidibile in tempo lineare controllando che esista almeno una congiunzione della formula che è soddisfacibile da un qualche assegnamento.

Si noti che la definizione di $\mathcal{NP}$ è asimmetrica: dato un problema $X = (\mathcal{I}, q) \in \mathcal{NP}$, se $q(I) = 1$ allora esiste un certificato polinomiale z tale che $B(I, z) = 1$ dove B denota il certificatore polinomiale per X. Se invece $q(I) = 0$ allora la definizione ci garantisce solo che $B(I, z) = 0$ per un numero esponenziale di certificati z. Questa asimmetria si riscontra quanto consideriamo problemi $\bar{X} = (\mathcal{I}, \bar{q})$ che sono complementari di problemi $X = (\mathcal{I}, q)$. Per *complementare* di X si intende che $\bar{q}(I) = 0$, ma non sappiamo se esista un certificato polinomiale che certifichi $\bar{q}(I) = 1$. Per esempio, se X è Independent Set, allora per certificare $\bar{q}(I) = 1$ dovremmo trovare un certificato polinomiale che attesti che il grafo **non** contenga un insieme indipendente di taglia k. Quindi non è chiaro se $\mathcal{NP} \equiv$ co-$\mathcal{NP}$, dove co-$\mathcal{NP}$ indica la classe dei problemi di decisione complementari di problemi in $\mathcal{NP}$ (se $\mathcal{NP}$ è l'insieme dei problemi di cui posso verificare una soluzione/certificato in tempo polinomiale, per i problemi in co-$\mathcal{NP}$ non è possibile avere una soluzione verificabilemente corretta in tempo polinomiale, è invece possibile verificare in tempo polinomiale che una soluzione restituisce 0; alternativamente, si può verificare in tempo polinomiale che una soluzione sia "sbagliata"). Lo scenario è differente per $\mathcal{P}$: $X \in \mathcal{P}$ se e solo se $\bar{X} \in \mathcal{P}$. Infatti $X = (\mathcal{I}, q) \in \mathcal{P}$ implica che esiste un algoritmo

per calcolare q in tempo polinomiale. Ma allora si può anche calcolare $\bar{q}$ in tempo polinomiale semplicemente calcolando q e complementando l'output. Quindi $\mathcal{P} \equiv \text{co-}\mathcal{P}$.

Dimostrare che co-$\mathcal{NP}$ $\not\equiv \mathcal{NP}$ sarebbe un progresso ancora maggiore che dimostrare $\mathcal{P} \not\equiv \mathcal{NP}$. Vale infatti la seguente cosa.

Fatto 1.3.3. *Se co-$\mathcal{NP} \not\equiv \mathcal{NP}$ allora $\mathcal{P} \not\equiv \mathcal{NP}$.*

Dimostrazione. Dimostriamo la contrapositiva, ovvero che $\mathcal{P} \equiv \mathcal{NP}$ implica co-$\mathcal{NP} \equiv \mathcal{NP}$. Intuitivamente, dato che $\mathcal{P}$ è chiuso rispetto all'operazione di complemento, se $\mathcal{P} \equiv \mathcal{NP}$ allora deve essere che co-$\mathcal{NP} \equiv \mathcal{NP}$. Formalmente

$$X \in \mathcal{NP} \iff X \in \mathcal{P} \iff \bar{X} \in \mathcal{P} \iff \bar{X} \in \mathcal{NP} \iff X \in \text{co-}\mathcal{NP}$$

Il che conclude la dimostrazione. □

Possiamo caratterizzare i problemi $X = (\mathcal{I}, q) \in$ co-$\mathcal{NP}$ tramite l'esistenza di un polinomio $p(\cdot)$ e di un certificatore polinomiale B, calcolabile in tempo polinomiale, tale che $q(I) = 0$ se e solo se esiste una stringa $z \in \{0,1\}^{p(|I|)}$ tale che $B(I, z) = 0$.

Si noti che $\mathcal{P} \subseteq$ co-$\mathcal{NP}$, infatti, potendo implementare q in tempo polinomiale, si può calcolare $B(I, z)$ ignorando z e restituendo $q(I)$.

Particolarmente interessante la classe dei problemi in $\mathcal{NP} \cap$ co-$\mathcal{NP}$. Sia $X = (\mathcal{I}, q)$ un tale problema

- Dato che $X \in \mathcal{NP}$ esiste un polinomio $p(\cdot)$ e un certificatore polinomiale B tale che $q(I) = 1$ se e solo se esiste una stringa $z \in \{0,1\}^{p(|I|)}$ tale che $B(I, z) = 1$
- Dato che $X \in$ co-$\mathcal{NP}$ esiste un polinomio $p'(\cdot)$ e un certificatore polinomiale B' tale che $q(I) = 0$ se e solo se esiste una stringa $z' \in \{0,1\}^{p'(|I|)}$ tale che $B'(I, z') = 0$

Quindi i problemi in $\mathcal{NP} \cap$ co-$\mathcal{NP}$ sono tali che per ogni istanza I esiste un certificato polinomiale, sia quando $q(I) = 1$ sia quando $q(I) = 0$.

Si noti che se $X \in \mathcal{P}$ allora $X \in \mathcal{NP}$ ed anche $X \in$ co-$\mathcal{NP}$. Quindi $\mathcal{P} \subseteq \mathcal{NP} \cap$ co-$\mathcal{NP}$. D'altra parte non si sa se $\mathcal{P} \not\equiv \mathcal{NP} \cap$ co-$\mathcal{NP}$. Ovvero non si sa se esistono problemi le cui istanze hanno sempre certificati brevi ma tuttavia non sono risolubili in tempo polinomiale.

Capitolo 2

Algoritmi Probabilistici

2.1 Algoritmi Montecarlo e Las Vegas

Un algoritmo probabilistico è un algoritmo che ha accesso a un oracolo il quale, a ogni chiamata, restituisce in tempo unitario un bit casuale, ovvero una variabile casuale Y tale che $\mathbb{P}(Y = 1) = \mathbb{P}(Y = 0) = 1/2$. Inoltre, i bit restituiti in una sequenza di chiamate all'oracolo sono indipendenti.

Indicheremo con $Z \in \{0,1\}^*$ la stringa di bit casuali indipendenti che l'oracolo restituisce in una sequenza di chiamate. Indicheremo anche con $A(I, Z) \in \{0,1\}$ la variabile casuale che rappresenta l'output dell'algoritmo probabilistico A per un problema di decisione $X = (\mathcal{I}, q)$ e avente come input l'istanza $I \in \mathcal{I}$ e i bit casuali Z dell'oracolo. Similmente, indichiamo con $T_A(I, Z)$ la variabile casuale che rappresenta il tempo di esecuzione di A con input $I \in \mathcal{I}$ e bit casuali Z forniti dall'oracolo.

Esistono due principali tipi di algoritmi probabilistici.

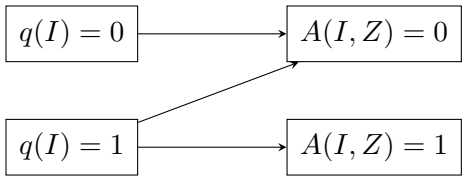
Algoritmi Montecarlo. Sono algoritmi A tali che

- Per ogni $I \in \mathcal{I}$, $T_A(I, Z)$ dipende solo da I, ovvero il tempo di esecuzione è deterministico
- Esiste $I \in \mathcal{I}$ per cui $\mathbb{P}\left(A(I, Z) \neq q(I)\right) > 0$, ovvero l'output non è sempre corretto

Gli algoritmi Montecarlo si dividono ulteriormente in:

- Algoritmi con errore **one-sided**; un algoritmo ha errore one-sided quando è sempre corretto almeno su uno dei suoi due possibili output. Convenzionalmente, assumeremo che l'algoritmo sia sempre corretto per output 1. In altre parole, A è Montecarlo one-sided quando, per ogni $I \in \mathcal{I}$, $\mathbb{P}\left(A(I,Z) = q(I) \mid A(I,Z) = 1\right) = 1$ e $\mathbb{P}\left(A(I,Z) = q(I) \mid A(I,Z) = 0\right) > 0$
- Algoritmi con errore **two-sided**. Possono sbagliare su entrambi i possibili output. Ovvero, A è Montecarlo two-sided quando $\mathbb{P}\left(A(I,Z) = q(I)\right) > 1/2$ per ogni $I \in \mathcal{I}$

Una rappresentazione grafica della relazione fra $q(I)$ e $A(I,Z)$ per un algoritmo Montecarlo one-sided è la seguente



Questo mostra come $A(I,Z) = 1$ può solo corrispondere a $q(I) = 1$, mentre $A(I,Z) = 0$ lascia incertezza sul valore di $q(I)$. Si noti anche che quando $q(I) = 0$ l'algoritmo è sempre corretto.

Algoritmi Las Vegas. Sono algoritmi che producono sempre l'output corretto ma che hanno un tempo di esecuzione probabilistico (ovvero che dipende dai bit forniti dall'oracolo). Ovvero, un algoritmo A è Las Vegas quando $\mathbb{P}\left(A(I,Z) = q(I)\right) = 1$ per ogni $I \in \mathcal{I}$ ma il tempo di calcolo di A su una qualunque istanza $I \in \mathcal{I}$ è una variabile casuale $T_A(I,Z)$ tale che $\mathbb{E}\left[T_A(I,Z)\right] < \infty$.

Amplificazione. Un algoritmo Montecarlo one-sided può essere facilmente trasformato in un algoritmo con probabilità di errore arbitrariamente piccola attraverso un meccanismo di amplificazione

Sia

$$\mathbb{P}\left(A(I,Z) \neq q(I) \mid A(I,Z) = 0\right) \leq 1 - p_n, \quad \forall I \in \mathcal{I} \text{ con } |I| = n$$

Se su una data istanza l'algoritmo produce 0 in output possiamo eseguirlo nuovamente per amplificare la probabilità di avere l'output corretto. Se k esecuzioni indipendenti producono sistematicamente la risposta 0, allora la probabilità che la risposta corretta sia 1 è al più $(1-p_n)^k \leq e^{-p_n k}$. Perché

questa probabilità sia al più un $\epsilon > 0$ piccolo a piacere è sufficiente scegliere $k \geq \frac{1}{p_n} \ln \frac{1}{\epsilon}$.

Un meccanismo di amplificazione simile ma leggermente più complesso esiste anche per gli algoritmi Montecarlo two-sided. Supponiamo che su istanze di taglia n l'algoritmo fornisca la risposta errata con probabilità al più $\frac{1}{2} - p_n < \frac{1}{2}$. Per amplificare la probabilità di ottenere la risposta corretta possiamo ripetere l'esecuzione k volte e utilizzare un voto di maggioranza sui k output prodotti (supponiamo, per semplicità, che k sia dispari).

Per analizzare il voto di maggioranza utilizzeremo il seguente lemma.

Lemma 2.1.1 (Chernoff-Hoeffding)**.** *Siano $Y_1, \ldots, Y_k$ variabili casuali Bernoulliane (cioè con valori in $\{0,1\}$), indipendenti e tali che $\mathbb{P}(Y_t = 1) \leq \mu$ per $t = 1, \ldots, k$. Allora, per ogni $\epsilon > 0$ fissato*

$$\mathbb{P}\left(\frac{1}{k}\sum_{t=1}^{k} Y_t > \mu + \epsilon\right) \leq e^{-2\epsilon^2 k}$$

Dimostrazione. Omessa. □

In altre parole, date delle variabili Bernoulliane indipendenti, con limite μ per il valore atteso, la disuguaglianza stabilisce la probabilità che la media si discosti di una certa quantità (ϵ) dal valore atteso μ. Intuitivamente, rappresenta la probabilità di allontanarsi dal valore atteso all'aumentare del numero di "tentativi" (variabili).

Sia $p_E = \mathbb{P}(A(I, Z) \neq q(I))$ la probabilità di errore dell'algoritmo su un'istanza I del problema di decisione $(\mathcal{I}, q)$. Siano $X_1, \ldots, X_k \in \{0,1\}$ le variabili casuali indipendenti che denotano gli output prodotti dalle k esecuzioni dell'algoritmo. Sia $M_k \in \{0,1\}$ la variabile casuale che denota il voto di maggioranza su $X_1, \ldots, X_k$ (ovvero $M_k = 1$ se e solo se $\sum_{t=1}^{k} X_t > \frac{k}{2}$).

Allora

$$M_k = q(I) \iff \sum_{t=1}^{k} Y_t < \frac{k}{2}$$

dove $Y_t = 1$ se e solo se $X_t \neq q(I)$, per $t = 1, \ldots, k$ (ovvero, se il risultato della relativa esecuzione dell'algoritmo è sbagliato). In altre parole, il voto di maggioranza M_k è corretto se e solo se l'algoritmo genera output errato in non più di $\lfloor \frac{k}{2} \rfloor$ delle k esecuzioni.

Ora, $Y_1, \ldots, Y_k$ sono variabili casuali indipendenti (perché le esecuzioni dell'algoritmo sono indipendenti), identicamente distribuite, con valori in $\{0, 1\}$ e tali che $\mathbb{P}(Y_t = 1) = p_E \leq \frac{1}{2} - p_n$ per ogni $t = 1, \ldots, k$.

Applicando il lemma di Chernoff-Hoeffding (2.1.1) otteniamo che la probabilità che il voto di maggioranza sia sbagliato è limitata da

$$\begin{aligned}
\mathbb{P}\left(\sum_{t=1}^{k} Y_t > \frac{k}{2}\right) &= \mathbb{P}\left(\frac{1}{k}\sum_{t=1}^{k} Y_t > \frac{1}{2}\right) \\
&= \mathbb{P}\left(\frac{1}{k}\sum_{t=1}^{k} Y_t > \left(\frac{1}{2} - p_n\right) + p_n\right) \\
&\leq \mathbb{P}\left(\frac{1}{k}\sum_{t=1}^{k} Y_t > p_E + p_n\right) \\
&\leq e^{-2p_n^2 k}
\end{aligned}$$

Perché la probabilità $e^{-2p_n^2 k}$ sia al più un $\epsilon > 0$ piccolo a piacere è sufficiente scegliere

$$k \geq \frac{1}{2p_n^2} \ln \frac{1}{\epsilon}$$

Si noti che nel caso one-sided possiamo usare l'amplificazione per ridurre qualsiasi probabilità di errore strettamente minore di 1, mentre nel caso two-sided la stessa cosa vale per qualsiasi probabilità strettamente minore di $\frac{1}{2}$.

Da Las Vegas a Montecarlo one-sided. Un algoritmo Las Vegas per un problema di decisione può essere trasformato in un algoritmo Montecarlo one-sided. Per fare ciò utilizziamo la disuguaglianza di Markov.

Lemma 2.1.2 (Markov)**.** *Sia Z una variabile casuale non negativa tale che $\mathbb{E}[Z] < \infty$. Allora per ogni $c > 0$*

$$\mathbb{P}(Z > c) \leq \frac{\mathbb{E}[Z]}{c}$$

Dimostrazione. Sia $\mathcal{A}$ l'insieme di numeri non negativi tali che $Z \in \mathcal{A}$. Allora

$$\begin{aligned}\mathbb{E}[Z] &= \sum_{a\in\mathcal{A}} a\mathbb{P}(Z=a) = \overbrace{\sum_{a\in\mathcal{A}:a\leq c} a\mathbb{P}(Z=a)}^{\geq 0} + \sum_{a\in\mathcal{A}:a>c} a\mathbb{P}(Z=a) \\ &\geq c \sum_{a\in\mathcal{A}:a>c} \mathbb{P}(Z=a) \\ &= c\mathbb{P}(Z=c)\end{aligned}$$

$$\mathbb{E}[Z] \geq c\mathbb{P}(Z>c) \implies \mathbb{P}(Z>c) \leq \frac{\mathbb{E}[Z]}{c}$$

concludendo la dimostrazione. □

Sia A un algoritmo Las Vegas per un problema di decisione $(\mathcal{I}, q)$. Sia $f : \mathbb{N} \to \mathbb{R}$ la funzione tale che

$$f(n) = \max\left\{\mathbb{E}\left[T_A(I,Z)\right] \mid I \in \mathcal{I},\ |I| = n\right\}$$

In altre parole, la funzione rappresenta il massimo del valore atteso per il tempo di esecuzione dell'algoritmo A per istanze di dimensione n.

Dato che A è Las Vegas, $f(n) < \infty$ per ogni $n \in \mathbb{N}$. Possiamo quindi trovare una funzione $t : \mathbb{N} \to \mathbb{N}$ tale che

$$t(n) \geq \frac{3}{2} f(n), \quad n \in \mathbb{N}$$

Posso quindi costruire un algoritmo A' che simula A sull'istanza I arrestando la simulazione dopo $t(n)$ passi. Se A non ha terminato, allora A' produce 0 in output. Dato che A è Las Vegas, A' sbaglia solo quando A non termina entro $t(n)$ passi. Per la disuguaglianza di Markov, la probabilità che ciò accada è al più

$$\mathbb{P}\left(T_A(I,Z) > t(n)\right) \leq \frac{\mathbb{E}\left[T_A(I,Z)\right]}{t(n)} \leq \frac{f(n)}{t(n)} \leq \frac{2}{3}$$

Inoltre, dato che quando A non termina l'output di A' è 0, A' è one-sided dato che l'output 1 è sempre corretto. Quindi A' è un algoritmo Montecarlo one-sided con probabilità di errore al più 2/3. Infine, si noti che il tempo di esecuzione di A' soddisfa $T_{A'}(I) \leq t(|I|) = \mathcal{O}\left(f(|I|)\right)$.

Riassumendo: un algoritmo Las Vegas può diventare Montecarlo one-sided con i seguenti passi

- Trova una funzione $t(n)$ che maggiora la funzione $f(n)$, la quale rappresenta il tempo di esecuzione atteso per input di taglia n, ovvero $t(n) = m \cdot f(n)$
- L'algoritmo Montecarlo usa l'algoritmo Las Vegas, bloccando quest'ultimo se non termina entro $t(n)$ passi; restituisce sempre il risultato dell'algoritmo Las Vegas, oppure 0 se questo non termina
- La probabilità di errore è solo nel caso in cui esca 0; per la disuguaglianza di Markov, la probabilità che ciò accada è $\leq \frac{1}{m}$; quando esce 1, l'algoritmo è sempre corretto (quindi one-sided)

2.2 Classi di complessità probabilistiche

Un algoritmo probabilistico per un problema di decisione $X = (\mathcal{I}, q)$ può essere visto come un algoritmo deterministico che ha accesso a una stringa Z di bit casuali. L'algoritmo calcola una funzione $B : \mathcal{I} \times \{0,1\}^* \to \{0,1\}$ tale che, per ogni $I \in \mathcal{I}$, $B(I, Z) = q(I)$ con una certa probabilità rispetto all'estrazione della stringa Z.

In analogia con la definizione di $\mathcal{NP}$, possiamo definire le classi di problemi di decisione solubili in tempo polinomiale da diversi tipi di algoritmi probabilistici rivisitando la nozione di certificatore polinomiale.

$\mathcal{BPP}$: La classe di problemi di decisione risolti in modo efficiente da algoritmi Montecarlo two-sided è la classe $\mathcal{BPP}$. Un problema di decisione $X = (\mathcal{I}, q)$ appartiene alla classe $\mathcal{BPP}$ se esiste una funzione $B : \mathcal{I} \times \{0,1\}^* \to \{0,1\}$ calcolabile in tempo polinomiale e un polinomio $p(\cdot)$ tali che, per ogni istanza $I \in \mathcal{I}$ essi soddisfano

$$\mathbb{P}\left(B(I, Z) \neq q(I)\right) \leq \frac{1}{3}$$

dove la probabilità è calcolata rispetto all'estrazione di Z con probabilità uniforme da $\{0,1\}^{p(|I|)}$.

La costante $1/3$ è arbitraria, dato che, come già visto, si può ridurre a piacimento la probabilità di errore di un algoritmo two-sided tramite amplificazione. Una definizione equivalente di $\mathcal{BPP}$ sostituisce la diseguaglianza sopra con

$$\mathbb{P}\left(B(I, Z) \neq q(I)\right) \leq \frac{1}{2} - \frac{1}{p'(|I|)}$$

dove $p'(\cdot)$ è un polinomio. Il meccanismo di amplificazione tramite Lemma di Chernoff-Hoeffding (2.1.1) implica che è sufficiente eseguire l'algoritmo un numero di volte pari a ordine di $p'(|I|)^2$ per ottenere una probabilità di errore limitata da $1/3$. Dato che $p'(\cdot)$ è un polinomio, l'algoritmo risultante è ancora polinomiale in $|I|$.

In altre parole, $\mathcal{BPP}$ è la classe di algoritmi risolti in modo efficiente da algoritmi Montecarlo two-sided più di metà delle volte, dove la probabilità si può stabilire a piacimento tramite amplificazione rimanendo efficiente.

Si noti che $\mathcal{P} \subseteq \mathcal{BPP}$, dato che avendo un algoritmo polinomiale per calcolare la funzione di decisione q possiamo implementare il certificatore B in tempo polinomiale con probabilità di errore pari a zero. Non è invece noto se $\mathcal{P} \equiv \mathcal{BPP}$, ovvero se ogni algoritmo Montecarlo two-sided possa essere "derandomizzato" in modo da ottenere un algoritmo deterministico polinomiale per lo stesso problema. Non è neanche noto se $\mathcal{BPP} \subseteq \mathcal{NP}$. D'altra parte, dato che la condizione che definisce $\mathcal{BPP}$ è simmetrica rispetto al valore di $q(I)$, ne deduciamo che $\mathcal{BPP}$ è chiusa rispetto al complemento, ovvero $\mathcal{BPP} \equiv$ co-$\mathcal{BPP}$.

$\mathcal{RP}$: La classe di problemi di decisione risolti in modo efficiente da algoritmi Montecarlo one-sided è la classe $\mathcal{RP}$. Un problema di decisione $X = (\mathcal{I}, q)$ appartiene alla classe $\mathcal{RP}$ se esiste una funzione $B : \mathcal{I} \times \{0,1\}^* \to \{0,1\}$ calcolabile in tempo polinomiale e un polinomio $p(\cdot)$ tali che, per ogni istanza $I \in \mathcal{I}$

$$\mathbb{P}\left(B(I,Z) = 1\right) \geq \frac{2}{3}, \quad \text{se } q(I) = 1$$
$$\mathbb{P}\left(B(I,Z) = 0\right) = 1, \quad \text{se } q(I) = 0$$

dove la probabilità è calcolata rispetto all'estrazione di Z con probabilità uniforme da $\{0,1\}^{p(|I|)}$.

Si noti che questa definizione corrisponde all'osservazione precedentemente fatta che un algoritmo Montecarlo one-sided è sempre corretto quando $q(I) = 0$. Quando $q(I) = 1$, l'algoritmo è corretto con probabilità almeno $2/3$. Quindi l'algoritmo è sempre corretto su output 1, mentre sbaglia con probabilità al più $1/3$ su output 0. Anche in questo caso la costante $1/3$ è arbitraria grazie al meccanismo di amplificazione.

Possiamo dare una definizione equivalente di $\mathcal{RP}$ sostituendo la prima delle due condizioni con

$$\mathbb{P}\left(B(I,Z) = 1\right) \geq \frac{1}{p'(|I|)}, \quad \text{se } q(I) = 1$$

dove $p'(\cdot)$ è un polinomio. Il meccanismo di amplificazione implica che è sufficiente eseguire l'algoritmo un numero di volte pari a ordine di $p'(|I|)$ per ottenere una probabilità di errore limitata da $1/3$. Dato che $p'(\cdot)$ è un polinomio, l'algoritmo risultante è ancora polinomiale in $|I|$.

Con un ragionamento simile a quello che ci ha portato a concludere che $\mathcal{P} \subseteq \mathcal{BPP}$, possiamo anche dimostrare che $\mathcal{P} \subseteq \mathcal{RP}$. Ma, a differenza di $\mathcal{BPP}$, questa volta possiamo stabilire una relazione tra $\mathcal{RP}$ e $\mathcal{NP}$. Infatti, la definizione di $\mathcal{NP}$ può essere equivalentemente riscritta nel modo seguente. Un problema di decisione $X = (\mathcal{I}, q)$ appartiene alla classe $\mathcal{NP}$ se esiste una funzione $B : \mathcal{I} \times \{0,1\}^* \to \{0,1\}$ calcolabile in tempo polinomiale e un polinomio $p(\cdot)$ tali che, per ogni istanza $I \in \mathcal{I}$, essi soddisfano

$$\mathbb{P}\left(B(I,Z) = 1\right) > 0, \quad \text{se } q(I) = 1$$

$$\mathbb{P}\left(B(I,Z) = 0\right) = 1, \quad \text{se } q(I) = 0$$

dove le probabilità sono calcolate rispetto all'estrazione di Z con probabilità uniforme da $\{0,1\}^{p(|I|)}$.

Dato che per la prima condizione di $\mathcal{RP}$, $\mathcal{P}\left(B(I,Z) = 1\right) \geq \frac{2}{3}$ implica $\mathcal{P}\left(B(I,Z) = 1\right) > 0$, mentre la seconda condizione è uguale nelle due definizioni. Di conseguenza, concludiamo che $\mathcal{RP} \subseteq \mathcal{NP}$. In altre parole, interpretiamo i bit casuali Z nella definizione di $\mathcal{RP}$ come un certificato del fatto che $q(I) = 1$.

La classe co-$\mathcal{RP}$ contiene i problemi che sono complementi di problemi in $\mathcal{RP}$. La definizione di co-$\mathcal{RP}$ è semplicemente ottenuta invertendo $q(I) = 0$ e $q(I) = 1$ nella definizione di $\mathcal{RP}$. Con una dimostrazione simile a quella di $\mathcal{RP} \subseteq \mathcal{NP}$ possiamo dimostrare che co-$\mathcal{RP} \subseteq$ co-$\mathcal{NP}$. Come vale $\mathcal{P} \subseteq \mathcal{RP}$ così possiamo dimostrare che $\mathcal{P} \subseteq$ co-$\mathcal{RP}$.

Possiamo mettere in relazione $\mathcal{RP}$ e co-$\mathcal{RP}$ con $\mathcal{BPP}$ riscrivendo la definizione di quest'ultima come

$$\mathbb{P}\left(B(I,Z) = 1\right) \geq \frac{2}{3}, \quad \text{se } q(I) = 1$$

$$\mathbb{P}\left(B(I,Z) = 0\right) \geq \frac{2}{3}, \quad \text{se } q(I) = 0$$

Arrivando così alla conclusione $\mathcal{RP} \subseteq \mathcal{BPP}$ e co-$\mathcal{RP} \subseteq \mathcal{BPP}$.

Introduciamo ora la classe $\mathcal{ZPP} \equiv \mathcal{RP} \cap$ co-$\mathcal{RP}$. Un problema di decisione $X = (\mathcal{I}, q)$ appartiene alla classe $\mathcal{ZPP}$ se esistono due funzioni B, B' :

$\mathcal{I} \times \{0,1\}^* \to \{0,1\}$ calcolabili in tempo polinomiale e due polinomi $p(\cdot)$, $p'(\cdot)$ tali che, per ogni istanza $I \in \mathcal{I}$, essi soddisfano

$$\mathbb{P}\left(B(I,Z)=1\right) \geq \frac{2}{3} \text{ e } \mathbb{P}\left(B'(I,Z')=1\right)=1 \text{ se } q(I)=1$$

$$\mathbb{P}\left(B(I,Z)=0\right) \geq \frac{2}{3} \text{ e } \mathbb{P}\left(B'(I,Z')=0\right)=1 \text{ se } q(I)=0$$

dove le probabilità sono calcolate rispetto all'estrazione di Z con probabilità uniforme da $\{0,1\}^{p(|I|)}$ e di Z' con probabilità uniforme da $\{0,1\}^{p'(|I|)}$.

Non è difficile vedere che la classe $\mathcal{ZPP}$ è la classe dei problemi risolti da algoritmi Las Vegas che terminano in tempo atteso limitato da un polinomio nella lunghezza dell'istanza. Per farlo, abbiamo bisogno del lemma seguente.

Lemma 2.2.1 (Valore atteso distribuzione Geometrica)**.** *Siano $Z_1, Z_2, \ldots$ variabili casuali Bernoulliane, indipendenti e tali che $\mathbb{P}(Z_t = 1) = p$ per $t \geq 1$. Sia $G = \min\{k \mid Z_k = 1\}$. Allora $\mathbb{E}[G] = \frac{1}{p}$.*

Dimostrazione.

$$\begin{aligned}
\mathbb{E}[G] &= \sum_{k=1}^{\infty} k(1-p)^{k-1}p = p\sum_{k=1}^{\infty} k(1-p)^{k-1} \\
&= -p\sum_{k=1}^{\infty} \frac{d}{dp}(1-p)^k = -p\frac{d}{dp}\sum_{k=1}^{\infty}(1-p)^k \\
&= -p\frac{d}{dp}\left(\frac{1}{1-(1-p)} - 1\right) = -p\frac{d}{dp}\frac{1-p}{p} \\
&= -p\frac{-1}{p^2} = \frac{1}{p}
\end{aligned}$$

□

In altre parole, date delle variabili Bernoulliane indipendenti che restituiscono 1 con probabilità p, il valore atteso del numero di variabili da "estrarre" prima del primo 1 è $1/p$.

Ora, se $\mathcal{X} \in \mathcal{ZPP}$ allora posso costruire un algoritmo probabilistico A che, su input $I \in \mathcal{I}$, esegue B e B' arrestandosi non appena $B(I,Z) = 1$ oppure $B'(I,Z') = 0$. In entrambi i casi sappiamo che l'output è corretto, quindi

A si arresta sempre con la soluzione corretta. La probabilità che su una particolare istanza I si verifichi $B(I,Z)=0$ e $B'(I,Z')=1$ è

$$\begin{aligned}\mathbb{P}\left(B(I,Z)=0 \wedge B'(I,Z')=1\right) &= \mathbb{P}\left(B(I,Z)=0\right)\mathbb{P}\left(B'(I,Z')=1\right) \\ &\leq \begin{cases}\frac{1}{3}\cdot 1 & \text{se } q(I)=1 \\ 1\cdot\frac{1}{3} & \text{se } q(I)=0\end{cases}\end{aligned}$$

ovvero al più $1/3$ indipendentemente dal valore di $q(I)$. Quindi la probabilità che A si arresti con la soluzione corretta è almeno $2/3$ in ogni esecuzione di B e B'. Usando il lemma sul valore atteso della Geometrica (2.2.1), il numero atteso di ripetizioni è quindi al più $\frac{3}{2}<2$. Dato che per ipotesi B e B' terminano entrambe in tempo polinomiale, il tempo atteso di calcolo di A è polinomiale anch'esso.

D'altra parte, sia A un algoritmo Las Vegas per $(\mathcal{I},q)$ e sia $\mu(I)<p(|I|)$ il valore atteso del tempo di calcolo $T_A(I,Z)$ di A su input I. Per la disuguaglianza di Markov (2.1.2)

$$\mathbb{P}\left(T_A(I,Z)\geq\lceil 3\mu(I)\rceil\right)\leq\frac{1}{3}$$

Quindi se su input I arresto A dopo $\lceil 3\mu(I)\rceil$ passi, la probabilità che A non abbia terminato è al più $1/3$. Viceversa, quando A termina l'output è sempre corretto.

Possiamo quindi implementare le funzioni B e B' come segue:

- B esegue A e produce 0 se A non termina
- Quando $q(I)=0$ l'output di $B(I,Z)$ è deterministicamente 0
- Quando $q(I)=1$, l'output di $B(I,Z)$ è 1 con probabilità almeno $\frac{2}{3}$
- In modo simile possiamo implementare B'

Dato che $\mu(I)<p(|I|)$, B e B' terminano entrambi in tempo deterministico polinomiale.

Quindi, in particolare, $\mathcal{ZPP}\subseteq\mathcal{RP}$ come avevamo già osservato trasformando un algoritmo Las Vegas in uno Montecarlo one-sided. Ciò implica che risolvere un problema di decisione con un algoritmo Las Vegas è un risultato più forte che risolverlo con un algoritmo Montecarlo (one-sided o two-sided). Infine, dato che $\mathcal{P}$ è incluso sia in $\mathcal{RP}$ che in co-$\mathcal{RP}$, abbiamo che $\mathcal{P}\subseteq\mathcal{ZPP}$.

2.3 Estrattore di von Neumann

Un estrattore di causalità è una funzione che trasforma una sorgente non perfettamente casuale in una completamente casuale. Il più semplice estrattore è quello ideato da John von Neumann e risponde alla domanda: *come è possibile usare una moneta truccata per simulare dei lanci di moneta non truccata?*

Più precisamente, avendo una moneta con probabilità sconosciuta $0 < p < 1$ di restituire testa ogni volta che viene lanciata, la si vuole usare per simulare una sequenza di lanci di una moneta equa, ovvero con probabilità $1/2$ di restituire testa.

Siano $X_1, X_2, \ldots$ le variabili casuali Bernoulliane indipendenti con $\mathbb{P}(X_t = 1) = p$ che modellano i lanci della moneta truccata. Consideriamo le coppe $(X_1, X_2), (X_3, X_4), \ldots$ e notiamo che i valori possibili per ogni coppia sono:

$(0,0)$	con probabilità $(1-p)^2$
$(1,1)$	con probabilità p^2
$(0,1)$ e $(1,0)$	con probabilità $p(1-p)$

Quindi, per ogni coppia (X_{2k-1}, X_{2k}) gli eventi $(X_{2k-1}, X_{2k}) = (0,1)$ e $(X_{2k-1}, X_{2k}) = (1,0)$ sono equiprobabili e forniscono la sequenza di lanci desiderata.

Algoritmo 1: Estrattore di von Neumann

Input: Sequenza di lanci $X_1, X_2, \ldots$

```
1 for k = 1, 2, ... do
2     if X_{2k-1} ≠ X_{2k}          // Controlla se è una coppia utile
3     then
4         if X_{2k-1} = 1 then
5             Print "Testa"
6         else
7             Print "Croce"
```

Praticamente, lancia la moneta truccata finché non ottieni due valori diversi di seguito: se il primo dei due è testa, il lancio "equo" è testa, croce altrimenti. Questo funziona in quanto la probabilità che esca $(0,1)$ o $(1,0)$ è la stessa.

Possiamo ora calcolare quanti lanci di moneta truccata servono in media per simulare un lancio di moneta non truccata. Data una sequenza $Z_1, Z_2, \ldots$ di variabili Bernoulliane indipendenti tali che $\mathbb{P}(Z_k = 1) = q$ per $k \geq 1$, la variabile casuale geometrica G è definita come $G = \min\{k = 1, 2, \cdots \mid Z_k = 1\}$.

Chiaramente $\mathbb{P}(G = 1) = q$ e $\mathbb{P}(G = n) = (1-q)^{n-1}q$ per ogni $n > 1$. Non è difficile dimostrare che $\mathbb{E}[G] = \frac{1}{q}$ (Lemma 2.2.1).

Consideriamo ora la sequenza $Z_1, Z_2, \ldots$ di variabili Bernoulliane indipendenti tali che

$$Z_k = \begin{cases} 1 & X_{2k-1} \neq X_{2k} \\ 0 & \text{altrimenti} \end{cases}$$

Per quanto detto prima, $\mathbb{P}(Z_k = 1) = 2p(1-p)$. Sia G la variabile geometrica associata alla sequenza delle Z_k. Quindi il numero medio di lanci che mi servono è

$$2\mathbb{E}[G] = \frac{1}{p(1-p)}$$

2.4 Il problema del Coupon Collector

Il problema del Coupon Collector è definito come segue: sia $X_1, X_2, \ldots$ una sequenza di variabili casuali indipendenti e uniformemente distribuite su n valori distinti $a_1, \ldots, a_n$

$$\mathbb{P}(X_t = a_i) = \frac{1}{n}, \quad i = 1, \ldots, n, \quad t \geq 1$$

Calcolare $\mathbb{E}[N]$, dove $N = \min\{k \mid (\forall i \leq n)\,(\exists t \leq k)\, X_t = a_i\}$. In altre parole, N è il minimo numero di realizzazioni $x_1, \ldots, x_k$ sufficienti a osservare ciascun a_i almeno una volta.

Il nome *coupon collector* deriva dal problema di collezionare tutti gli n possibili coupon contenuti in prodotti da acquistare (per esempio, scatole di cereali), dove ogni scatola contiene uno qualsiasi dei buoni premio con probabilità uniforme.

Un problema equivalente è il seguente: supponiamo che a ogni lancio, una pallina cade con probabilità uniforme in una fra n possibili scatole. Quante palline devo lanciare in media affinché ce ne sia almeno una in ogni scatola?

Un'applicazione concreta del coupon collector è la seguente: supponiamo di voler sapere gli identificativi degli n router attraversati da una sequenza di pacchetti. Mentre non c'è abbastanza spazio in un pacchetto per memorizzare tutti gli n identificativi, è facile memorizzare in un pacchetto l'identificativo di un router a caso tra quelli attraversati. Ci si chiede allora quanti pacchetti servono in media per ottenere gli identificativi di tutti gli n router.

Per analizzare il problema, suddividiamo $X_1, X_2, \ldots$ in n blocchi di lunghezze $N_1, \ldots, N_n$, dove N_i è il numero di estrazioni aggiuntive che servono per ottenere l'i-esimo valore distinto avendone già osservati $i-1$ ("ho $i-1$ coupon, quante estrazioni mi servono per trovarne uno nuovo?"). Quindi

$$N = \sum_{i=1}^{n} N_i$$

Ovvero il numero totale di estrazioni. Le variabili casuali $N_1, \ldots, N_n$ sono tutte Geometriche. In particolare, quando $i-1$ valori distinti sono già stati osservati, la probabilità di osservarne uno nuovo, sapendo che la probabilità di trovare un coupon già visto è $(i-1)/n$, è

$$p_i = 1 - \frac{i-1}{n} = \frac{n-i+1}{n}$$

Infatti, $p_1 = 1$ e questo implica $N_1 = 1$ deterministicamente.

Ricordando che il valore atteso di una Geometrica di parametro p_i è $\mathbb{E}[N_i] = \frac{1}{p_i}$ (Lemma 2.2.1), per linearità del valore atteso abbiamo

$$\mathbb{E}[N] = \sum_{i=1}^{n} \mathbb{E}[N_i] = \sum_{i=1}^{n} \frac{1}{p_i} = \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{i=1}^{n} \frac{1}{i} = n \ln n + \Theta(n)$$

dove l'ultima uguaglianza vale perché la somma armonica $1 + \frac{1}{2} + \cdots + \frac{1}{n}$ è asintotica a $\ln n + \Theta(1)$.

2.5 Reservoir Sampling

Si consideri il problema di mantenere una struttura dati che, a ogni istante di tempo, contenga k elementi estratti a caso con probabilità uniforme da uno stream di elementi in ingresso. In particolare, vogliamo sviluppare un algoritmo che soddisfi il seguente invariante: per ogni $t \geq k$, ognuno dei primi

t elementi dello stream è contenuto nella struttura dati con probabilità pari a $\frac{k}{t}$ (il campionamento rappresenta in modo "equo" lo stream originale).

Per esempio, vogliamo stimare le percentuali delle varie tipologie di oggetti (libri, elettronica, abbigliamento, …) venduti su Amazon in un dato lasso di tempo. Se ogni oggetto venduto è campionato con la stessa probabilità, allora la distribuzione delle tipologie nel campionamento sarà tendenzialmente uguale a quella nello stream.

Studiamo il problema nel modello streaming: a ogni istante di tempo $t = 1, 2, \ldots$ l'algoritmo può accedere soltanto al t-esimo elemento x_t dello stream. Chiediamo inoltre che l'algoritmo lavori in spazio $\Theta(k)$.

Il seguente algoritmo soddisfa tutte le proprietà richieste.

Algoritmo 2: Reservoir Sampling

Input: Intero k

```
1 R = ∅                                          // Inizializza la riserva
2 for t = 1, 2, ... do
3 |   Leggi il prossimo elemento x_t nello stream
4 |   if t ≤ k then
5 |   |   Aggiungi x_t a R
6 |   else
7 |   |   Con probabilità k/t, sostituisci un elemento a caso in R con x_t
```

Nel caso in cui lo stream avesse lunghezza nota N, potremmo aggiungere alla riserva ogni elemento dello stream in modo indipendente con probabilità $\frac{k}{N}$. Questo garantirebbe la proprietà che ogni elemento dello stream è contenuto nella riserva con la stessa probabilità, ma il numero di elementi effettivamente inseriti nella riserva potrebbe essere maggiore o minore di k.

Teorema 2.5.1. *Sia R_t il contenuto della riserva dopo che sono stati osservati i primi t elementi dello stream. Per ogni $t \geq k$ vale: $\mathbb{P}(x_i \in R_t) = \frac{k}{t}$ per ogni $i \leq t$.*

Per la dimostrazione useremo più volte il fatto che, per ogni coppia di eventi A, B tale che $\mathbb{P}(B) > 0$ vale $\mathbb{P}(A \cap B) = \mathbb{P}(B)\mathbb{P}(A \mid B)$.

Dimostrazione. La dimostrazione è per induzione su $t \geq k$:

- **Base:** $t = k$. Allora $\mathbb{P}(x_i \in R_t) = 1 = \frac{k}{t}$ dato che $t = k$

- **Step:** Fissato $t \geq k$, assumiamo l'ipotesi induttiva

$$P\left(x_i \in R_t\right) = \frac{k}{t}, \quad \forall i \leq t$$

e dimostriamo

$$\mathbb{P}\left(x_i \in R_{t+1}\right) = \frac{k}{t+1}, \quad \forall i \leq t+1$$

Se $i = t+1$, allora vale la tesi per costruzione (riga 7 dell'algoritmo).

Se invece $i \leq t$, dato che $x_i \in R_{t+1}$ implica $x_i \in R_t$, abbiamo che $\mathbb{P}\left(x_i \in R_{t+1}\right) = \mathbb{P}\left(x_i \in R_{t+1},\ x_i \in R_t\right)$. Possiamo quindi scrivere

$$\begin{aligned}
\mathbb{P}\left(x_i \in R_{t+1}\right) &= \mathbb{P}\left(x_i \in R_{t+1} \cap x_i \in R_t\right) \\
&= \mathbb{P}\left(x_i \in R_t\right) \mathbb{P}\left(x_i \in R_{t+1} \mid x_i \in R_t\right) \\
&= \frac{k}{t} \cdot \mathbb{P}\left(x_i \in R_{t+1} \mid x_i \in R_t\right) \qquad \text{(per IH)}
\end{aligned}$$

Ora si osservi che, dato $x_i \in R_t$, abbiamo che $x_i \notin R_{t+1}$ implica $x_{t+1} \in R_{t+1}$ (se non c'è l'elemento i, vuol dire che abbiamo aggiunto $t+1$). Quindi possiamo scrivere

$$\begin{aligned}
\mathbb{P}\left(x_i \in R_{t+1} \mid x_i \in R_t\right) &= 1 - \mathbb{P}\left(x_i \notin R_{t+1} \mid x_i \in R_t\right) \\
&= 1 - \mathbb{P}\left(x_i \notin R_{t+1} \cap x_{t+1} \in R_{t+1} \mid x_i \in R_t\right) \\
&= 1 - \mathbb{P}\left(x_{t+1} \in R_{t+1} \mid x_i \in R_t\right) \cdot \\
&\quad \cdot \mathbb{P}\left(x_i \notin R_{t+1} \mid x_{t+1} \in R_{t+1},\ x_i \in R_t\right) \\
&= 1 - \frac{k}{t+1}\frac{1}{k} \\
&= \frac{t}{t+1}
\end{aligned}$$

dove

$$\mathbb{P}\left(x_{t+1} \in R_{t+1} \mid x_i \in R_t\right) = \mathbb{P}\left(x_{t+1} \in R_{t+1}\right) = \frac{k}{t+1}$$

per costruzione dell'algoritmo e

$$\mathbb{P}\left(x_i \notin R_{t+1} \mid x_{t+1} \in R_{t+1},\ x_i \in R_t\right) = \frac{1}{k}$$

dato che x_i ha probabilità uniforme di essere selezionato dalla riserva per far posto a x_{t+1}. Quindi

$$\begin{aligned}\mathbb{P}\left(x_i \in R_{t+1}\right) &= \mathbb{P}\left(x_i \in R_t\right)\mathbb{P}\left(x_i \in R_{t+1} \mid x_i \in R_t\right) \\ &= \frac{k}{t}\frac{t}{t+1} = \frac{k}{t+1}\end{aligned}$$

dove $\frac{k}{t}$ è la probabilità che l'elemento fosse già presente nella riserva al passo precedente, vale per IH. Questo conclude la dimostrazione.

□

2.6 Algoritmo di Karger per il taglio minimo

Si consideri un grafo non orientato $G = (V, E)$. Una partizione $S, T \subseteq V$ dei vertici di G (dunque $S \cap T \equiv \emptyset$ e $S \cup T \equiv V$) è detta *ammissibile* se $S \not\equiv \emptyset$ e $T \not\equiv \emptyset$. Un *taglio* in G è un insieme $\Gamma(S, T) \equiv \{(u, v) \in E \mid u \in S,\ v \in T\}$ per una partizione ammissibile S, T dei vertici di G (i lati da "tagliare" nel grafo per dividerlo nei due insiemi che formano la partizione). In un grafo non pesato, il costo di un taglio corrisponde alla sua cardinalità $|\Gamma(S, T)|$.

In quanto segue consideriamo grafi non pesati e senza cappi; tuttavia, ammettiamo la presenza di archi multipli fra coppie di nodi (*multigrafo*). In questo caso, E è un multinsieme di archi, poiché archi fra coppie di vertici distinti possono essere presenti con molteplicità diverse.

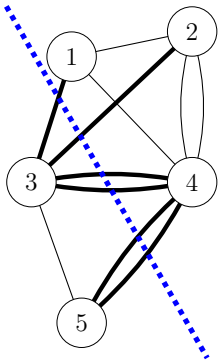


Figura 2.1: Un taglio in un multigrafo. Il taglio evidenziato è formato da 6 archi e corrisponde al multinsieme $\Gamma(S, T)$, dove $S = \{1, 2, 4\}$ e $T = \{3, 5\}$

Il problema del taglio minimo (MinCut) su un multigrafo è definito nel modo seguente.

Problema MinCut.
Istanza: Un multigrafo $G = (V, E)$.
Soluzione: Una partizione ammissibile S, T di V che minimizza il costo $|\Gamma(S, T)|$.

Il problema MinCut ha tantissime applicazioni. Per esempio, in un sistema distribuito dove i nodi rappresentano i processi e gli archi canali di comunicazione fra di essi, il taglio minimo corrisponde ad assegnare i processi a due CPU in modo che la comunicazione inter-processore – tipicamente lenta – sia minimizzata. Una seconda applicazione è la segmentazione di immagini. Qui i nodi rappresentano pixel e gli archi del grafo connettono pixel simili. Il taglio minimo corrisponde allora a una segmentazione dell'immagine in due parti tra loro il più dissimili possibile.

Il problema MinCut è facilmente risolvibile in tempo polinomiale deterministico, per esempio usando l'algoritmo di `Stoer-Wagner`, il quale ha un tempo di esecuzione dell'ordine $\mathcal{O}\left(|E||V| + |V|^2 \log |V|\right)$.

Mostriamo ora un semplice algoritmo probabilistico Monte Carlo, l'algoritmo di Karger, che trova il taglio minimo con probabilità almeno $1 - \epsilon$ in tempo pari a $\mathcal{O}\left(|E||V|^2 \log \frac{1}{\epsilon}\right)$.

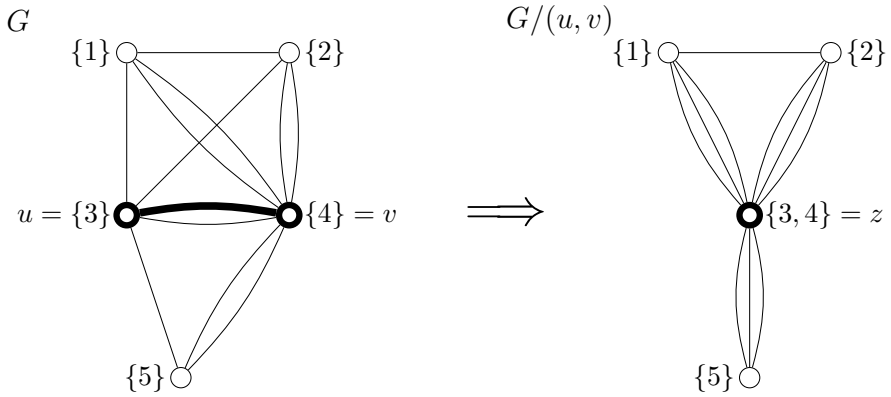


Figura 2.2: La contrazione di un arco in un multigrafo.

L'algoritmo di Karger è basato sull'operazione di contrazione di un arco (vedi Figura 2.2). La *contrazione* di un arco (u, v) in un multigrafo G produce un multigrafo $G \setminus (u, v)$ definito come risultato delle seguenti operazioni:

1. Un nuovo vertice z è aggiunto al grafo

2. Ogni arco $(w, u) \in E$ con $w \neq v$ è sostituito da un arco (w, z)
3. Ogni arco $(w, v) \in E$ con $w \neq u$ è sostituito da un arco (w, z)
4. Gli archi del tipo (u, v) e i vertici u, v sono rimossi

Nel seguito, diciamo che z è un supervertice che contiene u e v. Quando uno o entrambi i due vertici agli estremi di un arco contratto solo a loro volta supervertici, allora i nodi in essi contenuti diventano parte del nuovo supervertice.

Introduciamo ora l'algoritmo Karger "base", il quale contrae ripetutamente archi a caso del multigrafo fino a quando il numero di supervertici rimanenti è pari a due. Dato che la contrazione di un arco riduce di uno il numero di supervertici, l'algoritmo si fermerà dopo esattamente $|V| - 2$ passi. A questo punto, l'algoritmo produce il cut corrispondente all'unica partizione ammissibile dei due supervertici (si tagliano gli archi che collegano i vertici rimanenti). Dato che i supervertici del multigrafo finale corrispondono a una partizione dei vertici del multigrafo iniziale (i due supervertici dividono in due il grafo iniziale), abbiamo ottenuto un taglio del multigrafo iniziale.

Algoritmo 3: Karger-Base(G)

Input: Multigrafo $G = (V, E)$ con $|V| > 2$

```
1 while |V| > 2 do
2 |   Scegli un arco a caso (u, v) ∈ E
3 |   G ← G \ (u, v)
```

Output: l'unica partizione ammissibile S, T rimasta in G

Questo algoritmo può essere implementato in tempo $\mathcal{O}(|E|)$ rappresentando la sequenza di contrazioni tramite una permutazione causale degli archi di G (dettagli omessi).

Vediamo ora qual è la probabilità che questo algoritmo ritorni una partizione ammissibile S^*, T^* fissata che definisce un taglio $\Gamma^* = \Gamma(S^*, T^*)$ di costo minimo k in G; in particolare

$$k = |\Gamma^*| = \min_{(S,T)} |\Gamma(S, T)|$$

dove il minimo è su tutte le partizioni ammissibili S, T di V. Per prima cosa, si noti che l'algoritmo ritorna Γ^* se e solo se nessun arco in tale taglio viene contratto. Denotiamo ora con $X_1, \ldots, X_{|V|-2}$ la sequenza di archi contratti

dall'algoritmo. La probabilità che il primo arco X_1 contratto sia nel taglio corrisponde a

$$\mathbb{P}(X_1 \in \Gamma^*) = \frac{k}{|E|} \leq \frac{k}{|V|k/2} = \frac{2}{|V|}$$

perché la cardinalità di E è tale che

$$|E| = \frac{1}{2}\sum_{v \in V} d_v \geq \frac{1}{2}|V|d_{\min} \geq \frac{1}{2}|V|k$$

dove d_v è il grado di $v \in V$ in G, mentre $d_{\min}$ è il grado minimo dei nodi del grafo. In particolare, $d_{\min} \geq k$ è vero perché il costo k di un taglio minimo è sicuramente non superiore rispetto al costo d_v del taglio $\Gamma(\{v\}, V \setminus \{v\})$ per qualsiasi vertice $v \in V$. In altre parole, il numero di archi è sicuramente maggiore del numero di vertici moltiplicato per metà del grado minimo di qualsiasi vertice, e sappiamo che $d_{\min} \geq k$.

Quindi

$$\mathbb{P}\left(X_1 \notin \Gamma^*\right) \geq 1 - \frac{2}{|V|}$$

La probabilità che il secondo arco contratto X_2 non sia nel taglio Γ^*, dato che $X_1 \notin \Gamma^*$, è

$$\begin{aligned}
\mathbb{P}(X_2 \notin \Gamma^* \mid X_1 \notin \Gamma^*) &= 1 - \mathbb{P}(X_2 \in \Gamma^* \mid X_1 \notin \Gamma^*) \\
&\geq 1 - \frac{k}{(|V|-1)k/2} \\
&= 1 - \frac{2}{|V|-1}
\end{aligned}$$

In generale, denotando con A_i l'evento $X_i \notin \Gamma^*$ per $i \geq 1$, osserviamo che nessun arco di Γ^* è stato già contratto nel momento in cui dobbiamo scegliere X_i se condizioniamo sugli eventi $A_1, \ldots, A_{i-1}$. Il taglio Γ^* è dunque preservato sotto questo condizionamento. In aggiunta, al passo i-esimo, il grafo presenta $|V| - i + 1$ (super)vertici e un suo taglio minimo avrà ancora costo k condizionando su $A_1, \ldots, A_{i-1}$: le contrazioni si limitano a restringere le scelte di partizioni ammissibili dei vertici su cui valutare il costo dei tagli, e sappiamo che Γ^* è preservato.

Di conseguenza, abbiamo che

$$\begin{aligned}
\mathbb{P}\left(A_i \mid A_1, \ldots, A_{i-1}\right) &\geq 1 - \frac{k}{(|V| - i + 1)k/2} \\
&= 1 - \frac{2}{|V| - i + 1} \\
&= \frac{|V| - i + 1 - 2}{|V| - i + 1} \\
&= \frac{|V| - i - 1}{|V| - i + 1} \qquad (\dagger)
\end{aligned}$$

Dunque, indicando $\mathbb{P}(A_1 \mid A_0) = \mathbb{P}(A_1)$, dove A_0 corrisponde all'evento certo, possiamo scrivere

$$\begin{aligned}
\mathbb{P}(\text{restituisce } \Gamma^*) &= \mathbb{P}\left(X_i \notin \Gamma^*,\ \forall i \in [|V| - 2]\right) \\
&= \mathbb{P}\left(\bigcap_{i=1}^{|V|-2} A_i\right) \\
&= \prod_{i=1}^{|V|-2} \mathbb{P}\left(A_i \mid A_0, \ldots, A_{i-1}\right) \\
&\geq \prod_{i=0}^{|V|-3} \frac{|V| - i - 2}{|V| - i} \qquad \text{(per } (\dagger)\text{, sottraendo 1)} \\
&= \frac{\prod_{i=1}^{|V|-2} i}{\prod_{j=3}^{|V|} j} = \frac{(|V| - 2)!}{\frac{|V|!}{2!}} = \frac{1}{\binom{|V|}{2}} \qquad (\dagger\dagger)
\end{aligned}$$

Possiamo amplificare la probabilità di successo eseguendo più volte l'algoritmo e scegliendo la partiziona ammissibile che minimizza il costo del taglio tra tutte quelle ottenute. In particolare, se M è un numero sufficientemente grande, allora siamo in grado di ottenere un taglio di costo minimo in una delle M esecuzioni indipendenti dell'algoritmo con probabilità almeno $1 - \epsilon$, data una probabilità massima di errore $\epsilon \in (0, 1]$. Questa idea viene implementata nell'algoritmo seguente.

Ripetendo l'algoritmo base per $M = \left\lceil \binom{|V|}{2} \ln \frac{1}{\epsilon} \right\rceil = \mathcal{O}\left(|V|^2 \ln \frac{1}{\epsilon}\right)$ volte e scegliendo un taglio di costo minimo tra quelli prodotti, la probabilità che

Algoritmo 4: Karger(G, ϵ)

Input: Multigrafo $G = (V, E)$ con $|V| > 2$, parametro di confidenza $\epsilon \in (0, 1]$

$M \leftarrow \left\lceil \binom{|V|}{2} \ln \frac{1}{\epsilon} \right\rceil$

for $i = 1, \ldots, M$ **do**

  $S_i, T_i \leftarrow$ Karger-Base(G)

$j \in \arg\min_{i=1,\ldots,M} |\Gamma(S_i, T_i)|$

questo non abbia costo ottimo è

$$\begin{aligned}
\mathbb{P}(\text{Karger}(G,\epsilon) \text{ fallisce}) &= \mathbb{P}(|\Gamma(S_i,T_i)| > k,\ \forall i \in [M]) && \text{(nessun taglio output è minimo)}\\
&\leq \mathbb{P}(\Gamma(S_i,T_i) \not\equiv \Gamma^*,\ \forall i \in [M]) && \text{(nessun taglio prodotto è } \Gamma^*)\\
&= \mathbb{P}(\text{Karger-Base}(G) \text{ non restituisce } \Gamma^*)^M && \text{(indipendenza)}\\
&= \left(1 - \frac{1}{\binom{|V|}{2}}\right)^M && \text{(per (††))}\\
&\leq e^{-\frac{1}{\binom{|V|}{2}}M} && (1 - x \leq e^{-x})\\
&= e^{-\ln\frac{1}{\epsilon}} = \epsilon
\end{aligned}$$

dove abbiamo usato la maggiorazione $1 - x \leq e^{-x}$ per ogni $x \in \mathbb{R}$.

Il tempo totale di esecuzione, considerando anche il costo di ciascuna contrazione, è quindi $\mathcal{O}\left(|E||V|^2 \ln \frac{1}{\epsilon}\right)$, dove

- tempo $\mathcal{O}(|E|)$ per ogni ripetizione
- numero di ripetizioni $M = \left\lceil \binom{|V|}{2} \ln \frac{1}{\epsilon} \right\rceil = \mathcal{O}\left(|V|^2 \ln \frac{1}{\epsilon}\right)$

Una versione più sofisticata, nota come `algoritmo di Karger-Stein`, trova un taglio di costo minimo con probabilità almeno $1-\epsilon$ in tempo $\mathcal{O}\left((|V| \ln |V|)^2 \ln \frac{1}{\epsilon}\right)$.

Capitolo 3

Hashing e Randomizzazione

3.1 Hashing Universale

Sia $\mathcal{U}$ un insieme detto *universo*, con $|\mathcal{U}| = m \gg 1$. Consideriamo una tabella hash T di dimensione $1 < n \ll m$ e una funzione di hash $h : \mathcal{U} \to \{0, \ldots, n-1\}$.

Supponiamo di memorizzare un arbitrario sottoinsieme $S \subseteq \mathcal{U}$ nella tabella di hash usando la funzione h. Ovvero, memorizziamo ogni $u \in S$ nella posizione $h(u)$ di T. Dato che la tabella di hash è molto più piccola della cardinalità di $\mathcal{U}$, potranno avvenire delle collisioni. Una *collisione* consiste nell'evento in cui due elementi distinti $u, v \in S$ sono tali che $h(u) = h(v)$. Ciò significa che u e v andrebbero memorizzati nella stessa posizione della tabella hash T.

Questo problema viene solitamente gestito utilizzando, per esempio, delle liste associate a ogni posizione di T. Una conseguenza di questa soluzione alle collisioni riguarda il costo delle operazioni sulla tabella di hash: ogni ulteriore operazione di ricerca o inserimento sulla tabella richiederà tempo proporzionale alla lunghezza della lista associata alla posizione della tabella dove viene fatta l'operazione.

In un'analisi di caso peggiore, il tempo per eseguire un'operazione di ricerca o inserimento di un elemento $v \in \mathcal{U}$ è quindi proporzionale al massimo numero $\ell_{\max}$ di elementi in S diversi da v che sono stati mappati da h nella

stessa posizione di v in T. Più precisamente

$$\ell_{\max} = \max_{v \in \mathcal{U}} |\{u \in S \mid h(u) = h(v),\ u \neq v\}|$$

Idealmente, vorremmo avere $\ell_{\max} = \mathcal{O}(|S|/n)$, in modo da minimizzare il tempo di esecuzione di un'operazione nel caso peggiore (vogliamo che gli elementi siano distribuiti il più equamente possibile nella tabella).

Possiamo ottenere questo risultato usando la randomizzazione. Supponiamo di avere a disposizione una famiglia $\mathcal{H}$ di funzioni $h : \mathcal{U} \to \{0, \ldots, n-1\}$, e di usare una funzione di hash h che sia la realizzazione di una variabile casuale H corrispondente all'estrazione casuale (con probabilità uniforme) di un elemento da $\mathcal{H}$.

Supponiamo ora che $\mathcal{H}$ soddisfi la condizione seguente

$$\mathbb{P}(H(u) = H(v)) = \frac{1}{n}, \quad \text{per ogni } u, v \in \mathcal{U},\ u \neq v \tag{$\dagger$}$$

dove la probabilità è calcolata rispetto all'estrazione casuale della funzione di hash H da $\mathcal{H}$. In altre parole, ancora una volta, vogliamo che gli elementi siano il più possibile equamente distribuiti.

La condizione ($\dagger$) è sufficiente a garantire che

$$\max_{v \in \mathcal{U}} \mathbb{E}\left[|\{u \in S \mid H(u) = H(v),\ u \neq v\}|\right] \leq \frac{|S|}{n}$$

dove il valore atteso è calcolato rispetto all'estrazione di H da $\mathcal{H}$ (il numero di collisioni atteso è minimizzato, la proprietà desiderata precedentemente).

Infatti, supponiamo di aver estratto H a caso da $\mathcal{H}$ e di averla usata per inserire $S = \{u_1, \ldots, u_s\}$ nella tabella. Sia $v \in \mathcal{U}$ un elemento arbitrario che vogliamo cercare o inserire nella tabella. Definiamo le variabili casuali $X_1, \ldots, X_s$ dove $X_i = \mathbb{I}\{H(u_i) = H(v)\}$ e $\mathbb{I}\{\cdot\}$ è la funzione indicatrice di un evento, definita come

$$\mathbb{I}\{A\} = \begin{cases} 1 & \text{se } A \text{ è vero} \\ 0 & \text{altrimenti} \end{cases}$$

Per come utilizzata sopra, indica che la variabile X_i è a 1 se risulta una collisione tra u_i e v. In altre parole, la variabile casuale X_i vale 1 se e solo

se c'è una collisione tra il valore della funzione di hash per l'elemento u_i e per l'elemento v da cercare/inserire.

Si ricorda che una proprietà della funzione indicatrice di un evento A è che il suo valore atteso equivale a $\mathbb{E}\left[\mathbb{I}\{A\}\right] = \mathbb{P}(A)$. Allora

$$|\{u \in S \mid H(u) = H(v),\ u \neq v\}| = \sum_{i=1,\ u_i \neq v}^{s} X_i$$

rappresenta il numero di collisioni atteso (quanti 1 compaiono nelle variabili, escluso il caso $u_i = v$).

Il valore atteso di questo numero è calcolato come

$$\begin{aligned}
\mathbb{E}\left[\sum_{i=1,\ u_i \neq v}^{s} X_i\right] &= \sum_{i=1,\ u_i \neq v}^{s} \mathbb{E}[X_i] && \text{(linearità di } \mathbb{E}) \\
&= \sum_{i=1,\ u_i \neq v}^{s} \mathbb{P}\left(H(u_i) = H(v)\right) && \text{(definizione di } X_i) \\
&= \sum_{i=1,\ u_i \neq v}^{s} \frac{1}{n} && \text{(ipotesi su } \mathcal{H}, \text{eq. } (\dagger)) \\
&\leq \frac{|S|}{n} && \text{(al più } |S| \text{ elementi)}
\end{aligned}$$

Non è difficile trovare famiglie $\mathcal{H}$ che soddisfano la condizione (†). Assumiamo per semplicità che $\mathcal{U} = \{0, \ldots, m-1\}$ e consideriamo la classe $\mathcal{H}$ di tutte le funzioni $h : \{0, \ldots, m-1\} \to \{0, \ldots, n-1\}$. Ognuna di queste funzioni di hash corrisponde a un vettore $\boldsymbol{h} \in (h_0, \ldots, h_{m-1}) \in \{0, \ldots, n-1\}^m$ in modo che $h(u) = h_u$. Allora, il numero di vettori $\boldsymbol{h} \in \{0, \ldots, n-1\}^m$ tali che $h_u = h_v$ è n^{m-1} (n possibili valori per h_u, lo stesso valore per h_v e sono "liberi" i restanti $m-2$ valori dell'universo, quindi $n \cdot 1 \cdot n^{m-2}$ in totale), per $u, v \in \mathcal{U}$ distinti. Quindi, se estraggo H uniformemente a caso da $\mathcal{H}$

$$\mathbb{P}\left(H(u) = H(v)\right) = \frac{|\{\boldsymbol{h} \in \mathcal{H} \mid h_u = h_v\}|}{|\mathcal{H}|} = \frac{n^{m-1}}{n^m} = \frac{1}{n}$$

e la condizione (†) è soddisfatta. D'altra parte, la classe $\mathcal{H}$ non è utilizzabile in pratica in quanto mi servono $\lceil \log_2 |\mathcal{H}| \rceil = \Theta(m \log n)$ bit per memorizzare ciascuna $h \in \mathcal{H}$, e stiamo assumendo che $m \gg 1$.

In altre parole, è facile creare una famiglia di hash perfettamente uniforme (†) prendendo tutte le n^m funzioni possibili su un universo grande $|\mathcal{U}| = m$, ma per memorizzare ogni funzione appartenente a tale famiglia è necessario memorizzare l'associazione esplicita di ogni chiave al suo valore di hash, infattibile nella pratica.

Per eliminare situazioni di questo genere, aggiungiamo alla condizione (†) una richiesta ulteriore, ovvero che ogni $h \in \mathcal{H}$ possa essere rappresentata con al più $\Theta(\log m)$ bit (che è anche lo spazio che occorre per rappresentare un elemento arbitrario di $\mathcal{U}$) e calcolata in modo efficiente (chiamiamo questa proprietà (‡)). Una famiglia $\mathcal{H}$ di funzioni $h : \mathcal{U} \to \{0, \ldots, n-1\}$ che soddisfa entrambe queste condizioni è detta una **famiglia universale di funzioni di hash**.

Dimostriamo ora l'esistenza di famiglie universali. Oltre a $\mathcal{U} = \{0, \ldots, m-1\}$ assumiamo anche $n = p$ per un qualche p primo. Rappresentiamo ciascun elemento $u \in \{0, \ldots, m-1\}$ come un numero $[u]_p$ in base p. Più precisamente, $[u]_p = x = (x_1, \ldots, x_r)$ dove $x_i \in \{0, \ldots, p-1\}$ e r è il più piccolo intero tale che $p^r \geq m$ (valore che permette di rappresentare in base p tutti i valori di $\mathcal{U}$). Ovvero

$$r = \left\lceil \frac{\log_2 m}{\log_2 p} \right\rceil$$

Sia $\mathcal{A} = \{0, \ldots, p-1\}^r$ l'insieme usato per rappresentare gli elementi di $\mathcal{U}$. Introduciamo ora la famiglia di funzioni di hash $\mathcal{H} = \{h_a \mid a \in \mathcal{A}\}$ di tipo $h_a : \mathcal{A} \to \{0, \ldots, p-1\}$ (funzioni che vanno da ogni possibile valore nella rappresentazione base p scelta a una singola cifra) e definite come

$$h_a(u) = \left(\sum_{i=1}^{r} a_i x_i \right) \mod p, \quad \text{dove } x = [u]_p$$

Ovvero, ogni cifra del valore su cui viene applicata la funzione è moltiplicata per la cifra nella posizione corrispondente definita all'interno della funzione, vengono tutte sommate e poi messe modulo p, ovvero la base della rappresentazione (definizione di prodotto scalare tra vettori; in binario, sarebbero due array di bit, ogni posizione di uno moltiplicata con l'altro, tutti i valori sommati e riportati in base 2).

Si noti che si può rappresentare ogni h_a con

$$\lceil \log_2 |\mathcal{A}| \rceil = \Theta(r \log p) = \Theta\left(\frac{\log m}{\log p} \log p \right) = \Theta(\log m)$$

bit e si può calcolare h_a in modo efficiente (proprietà (‡)). Intuitivamente, si può vedere come sia facile da calcolare e la funzione di hash consiste effettivamente di $\log m$ valori, dove m è il massimo valore che si vuole considerare in input (per definizione di $\mathcal{U}$).

Per verificare la condizione (†) faremo uso del lemma seguente.

Lemma 3.1.1. *Per ogni primo p, per α, β e z interi*

$$(z \not\equiv 0 \mod p) \ \wedge \ (\alpha z \equiv \beta z \mod p) \implies \alpha \equiv \beta \mod p$$

Dimostrazione. Chiaramente, $\alpha z \equiv \beta z \mod p$ se e solo se $z(\alpha - \beta) \equiv 0 \mod p$. Inoltre, $z \not\equiv 0 \mod p$ e $z(\alpha - \beta) \equiv 0 \mod p$ implicano $(\alpha - \beta) \equiv 0 \mod p$. Infatti, se p è primo e z non contiene p come fattore, allora $\alpha - \beta$ lo deve contenere (se invece p non fosse primo, allora z e $\alpha - \beta$ potrebbero spartirsi i fattori primi di p).

$$\begin{aligned} \alpha z \equiv \beta z \mod p &\iff \alpha z - \beta z \equiv 0 \mod p \\ &\iff z(\alpha - \beta) \equiv 0 \mod p \\ &\iff \alpha - \beta \equiv 0 \mod p \qquad (z \not\equiv 0 \mod p) \\ &\iff \alpha \equiv \beta \mod p \end{aligned}$$

□

In altre parole, si può cancellare z dai lati di un'equazione, come in aritmetica classica, a patto che p sia primo e che z non sia multiplo di p.

Siamo pronti ora a dimostrare che per $\mathcal{H}$ vale la proprietà (†). Dato che, come abbiamo già osservato, per la stessa classe valeva anche la proprietà (‡), concludiamo che $\mathcal{H}$ è una famiglia universale di funzioni di hash.

Teorema 3.1.1. *$\mathcal{H}$ è tale che per ogni $u, v \in \mathcal{U}$ distinti, la frazione di elementi $a \in \mathcal{A}$ tali che $h_a(u) = h_a(v)$ è al più $\frac{1}{p}$. Quindi, se H è estratta a caso da $\mathcal{H}$, allora*

$$\mathbb{P}\left(H(u) = H(v)\right) \leq \frac{1}{p}$$

Dimostrazione. Siano $x = [u]_p$ e $y = [v]_p$. Sappiamo che $x \neq y$ poiché $u \neq v$. Allora, esiste almeno una coordinata $j \in \{1, \ldots, r\}$ tale che $x_j \neq y_j$. Per estrarre H a caso in $\mathcal{H}$ prendiamo $a \in \mathcal{A}$ estraendo a caso ciascuna

coordinata $a_i \in \{0, \ldots, p-1\}$. Per qualunque estrazione dei valori a_i sulle coordinate $i \neq j$ abbiamo che

$$\begin{aligned}
h_a(u) = h_a(v) &\iff \left(\sum_{i=1}^{r} a_i x_i \equiv \sum_{i=1}^{r} a_i y_i\right) \mod p && \text{(def. } h_a\text{)} \\
&\iff a_j x_j + \sum_{i=1; i\neq j}^{r} a_i x_i \equiv a_j y_j + \sum_{i=1; i\neq j}^{r} a_i y_i \mod p \\
&\iff a_j y_j - a_j x_j \equiv \sum_{i=1; i\neq j} a_i x_i - \sum_{i=1; i\neq j} a_i y_i \mod p \\
&\iff a_j \underbrace{(y_j - x_j)}_{z} \equiv \underbrace{\sum_{i: i\neq j} a_i (x_i - y_i)}_{k} \mod p \\
&\iff a_j z \equiv k \mod p
\end{aligned}$$

Dimostriamo ora che questa congruenza può essere soddisfatta da non più di un valore di a_j. In altre parole, per qualunque scelta dei valori di $a_i \in \{0, \ldots, p-1\}$ con $i \neq j$, esiste al più un unico valore di $a_j \in \{0, \ldots, p-1\}$ tale che $a_j z \equiv k \mod p$ sia vera.

Per assurdo, supponiamo che esistano a_j, a'_j valori distinti per la coordinata j che soddisfino entrambi la congruenza. Allora avremmo anche che $a_j z \equiv a'_j z \mod p$. Dato che $z \not\equiv 0 \mod p$ per ipotesi (perché $x_j \neq y_j$ e $0 \leq x_j, y_j < p$), il Lemma 3.1.1 implica che $a_j \equiv a'_j \mod p$. Siccome $0 \leq a_j, a'_j < p$, ciò significa che $a_j = a'_j$, si ha quindi una contraddizione. Quindi, c'è al più un solo valore della coordinata j che può rendere $h_a(u) = h_a(v)$ vera e la probabilità di estrarre questo valore fra i p possibili per a_j è al più $\frac{1}{p}$. □

Nota a margine: per maggiore chiarezza, potremmo argomentare più formalmente per arrivare alla medesima conclusione del teorema precedente. Seguendo lo stesso ragionamento, sappiamo che

$$\{h_a \mid h_a(u) = h_a(v),\ a \in \mathcal{A}\} = \{h_a \mid a_j z \equiv k \mod p,\ a \in \mathcal{A}\}$$

per via dell'equivalenza tra le due condizioni, e che dunque

$$\begin{aligned}
|\{h_a \mid h_a(u) = h_a(v),\ a \in \mathcal{A}\}| &= |\{h_a \mid a_j z \equiv k \mod p,\ a \in \mathcal{A}\}| \\
&= \sum_{a \in \mathcal{A}} \mathbb{I}\{a_j z \equiv k \mod p\} \\
&= \sum_{a_i \in \{0,\ldots,p-1\},\ i \neq j} \underbrace{\sum_{a_j \in \{0,\ldots,p-1\}} \mathbb{I}\{a_j z \equiv k \mod p\}}_{\leq 1} \\
&\leq \sum_{a_i \in \{0,\ldots,p-1\},\ i \neq j} 1 \\
&= |\{(a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_r) \mid a_i \in \{0, \ldots, p-1\},\ \forall i \neq j\}| \\
&= p^{r-1}
\end{aligned}$$

dove la disuguaglianza è dovuta all'esistenza di al più un valore di a_j che soddisfa la congruenza. Quindi, ricordando che $|\mathcal{H}| = |\mathcal{A}| = p^r$, la frazione di elementi $a \in \mathcal{A}$ tali che $h_a(u) = h_a(v)$ è

$$\frac{|\{h_a \mid h_a(u) = h_a(v),\ a \in \mathcal{A}\}|}{|\mathcal{H}|} \leq \frac{p^{r-1}}{p^r} = \frac{1}{p}$$

In conclusione, la tecnica di hash basata sul prodotto scalare di vettori (un vettore per la rappresentazione dell'elemento $u \in \mathcal{U}$ e uno per la funzione di hash) modulo un numero primo soddisfa la definizione di hashing universale.

3.2 Conteggio approssimato

Supponiamo di dover trovare, in una tabella di grandi dimensioni, tutti gli elementi che si ripetono più di un certo numero di volte. Per esempio, vogliamo trovare i prodotti visualizzati più frequentemente su Amazon, oppure le parole cercate più frequentemente su Google. Questo problema prende il nome di ricerca degli *heavy hitters* e, in astratto, richiede di trovare in una tabella di n interi tutti gli interi che si ripetono almeno n/k volte, dove $n \gg k$. Si noti che ci possono essere al più k heavy hitters e potrebbe non essercene neanche uno.

Partiamo da una versione più semplice del problema: vogliamo trovare nella tabella un numero che si ripete almeno $n/2$ volte, sapendo che tale numero è presente. Chiaramente, questo valore deve corrispondere alla mediana

di tutti i valori nella tabella e posso trovarlo in tempo $\mathcal{O}(n)$ con un algoritmo deterministico. Vediamo ora un semplicissimo algoritmo ad hoc che trova tale valore scandendo l'array una sola volta dall'inizio alla fine e usando una memoria ausiliaria sublineare (algoritmi di questo tipo si chiamano *streaming*).

Algoritmo 5: Boyer-Moore

Input: Array A

```
1  c ← 0                          // Inizializza contatore maggioranza
2  v ← NULL                       // Inizializza maggioranza corrente
3  for i = 1, ..., n do
4      if c = 0 then
           // Nessuna maggioranza
5          v ← A[i]
6          c ← c + 1
7      else
8          if A[i] = v then
               // Incremento maggioranza corrente
9              c ← c + 1
10         else
               // Decremento maggioranza corrente
11             c ← c − 1
```

Non è difficile verificare che quando l'algoritmo termina il valore corrente di v corrisponde al valore di maggioranza nella tabella. Ora ci chiediamo se esista una soluzione streaming anche per il problema di trovare gli heavy hitters. In realtà è possibile dimostrare che non esiste un algoritmo streaming che risolve il problema di ricerca degli heavy hitters con memoria ausiliaria sublineare (dimostrazione omessa).

Per riuscire a trovare una soluzione streaming, rilassiamo il problema originario introducendo una versione approssimata. Nel problema di ricerca di heavy hitters ϵ-approssimati (indicato ϵ-HH) abbiamo una tabella A di lunghezza n e due parametri k e ϵ con $\frac{1}{n} < \epsilon < \frac{1}{k}$. L'algoritmo deve produrre una lista di valori tali che:

1. Ogni valore che compare in A almeno $\frac{n}{k}$ volte è nella lista
2. Ogni valore nella lista compare almeno $\frac{n}{k} - \epsilon n$ volte in A

L'algoritmo che proponiamo è probabilistico e usa memoria ausiliaria $\Theta\left(\frac{\ln n}{\epsilon}\right)$.

Per risolvere il problema ϵ-HH utilizzeremo una struttura dati probabilistica chiamata count-min sketch. Questa struttura supporta due operazioni:

- INC(x) che incrementa il contatore associato a x
- COUNT(x) che ritorna il numero di volte che INC(x) è stato invocato

La struttura dati è composta da ℓ tabelle di hash, ciascuna di dimensione b. Siano $h_1, \ldots, h_\ell : \{1, \ldots, n\} \to \{0, \ldots, b-1\}$ le funzioni hash associate alle ℓ tabelle. Ogni tabella di hash comprime la tabella di n elementi in una di dimensione $b \ll n$. Le ℓ tabelle diverse servono a ridurre la probabilità di errore dovuto a collisione.

Il codice per le due operazioni e per la routine principale SELECTEL(A, k) è estremamente semplice.

Algoritmo 6: Count-Min Sketch

```
1  Crea matrice CMS[ℓ][b]
2  Procedure INC(x)
3      for i = 1, ..., ℓ do
4          Incrementa CMS[i][h_i(x)]
5  Procedure COUNT(x)
6      return min_{i=1,...,ℓ} CMS[i][h_i(x)]
7  Procedure SELECTEL(A, k)
8      Crea lista vuota
9      for t = 1, ..., n do
10         Leggi il prossimo elemento x_t = A[t]
11         Esegui INC(x_t)
12         if COUNT(x_t) ≥ n/k then
13             Aggiungi x_t alla lista (se non già presente)
14     Ritorna la lista
```

Sia x un valore che compare almeno una volta nella tabella A e sia n_x il numero di occorrenze di x in A. Dato che $b \ll n$ ci saranno delle collisioni, ovvero $h(x) = h(y)$ con $x \neq y$. Questo significa che

$$n_x \leq \text{CMS}[i][h_i(x)], \quad i = 1, \ldots, \ell$$

Infatti INC(x) verrà chiamata esattamente n_x volte, ma, a causa delle collisioni, due chiamate INC(x) e INC(y) tali che $h_i(x) = h_i(y)$ incrementeranno lo stesso contatore. Quindi, dato che ogni CMS$[i][h_i(x)]$ sovrastima n_x è sensato utilizzare come valore di COUNT(x) la più piccola di tali stime.

Analizziamo ora la probabilità di errore del count-min sketch quando le funzioni di hash $h_1, \ldots, h_\ell$ sono estratte a caso e in modo indipendente da una famiglia universale $\mathcal{H}$ di funzioni hash. Usiamo la notazione $H_1, \ldots, H_\ell$ per indicare che le funzioni sono variabili casuali opportunamente definite. Dato x, siano $Z_1, \ldots, Z_\ell$ le variabili casuali $Z_i = \text{CMS}[i][H_i(x)]$ dove la probabilità è rispetto all'estrazione di H_i da $\mathcal{H}$. Allora

$$Z_i = n_x + \sum_{y \neq x} n_y \mathbb{I}\{H_i(y) = H_i(x)\}$$

Ora, dato che $\mathcal{H}$ è una famiglia universale

$$\mathbb{P}(H_i(x) = H_i(y)) \leq \frac{1}{b}, \quad i = 1, \ldots, \ell$$

Quindi

$$\begin{aligned}
\mathbb{E}[Z_i] &= n_x + \sum_{y \neq x} n_y \mathbb{P}(H_i(y) = H_i(x)) && \text{(def. di } \mathbb{I}\text{)} \\
&\leq n_x + \sum_{y \neq x} \frac{n_y}{b} && (\mathcal{H} \text{ famiglia universale}) \\
&\leq n_x + \frac{n}{b}
\end{aligned}$$

Introduciamo le variabili casuali non negative $X_i = Z_i - n_x$. Scegliendo $b = \frac{e}{\epsilon}$ abbiamo che $\mathbb{E}[X_i] \leq \frac{\epsilon n}{e}$. Applicando la disuguaglianza di Markov (2.1.2) alle X_i otteniamo quindi

$$\mathbb{P}(Z_i \geq n_x + \epsilon n) = \mathbb{P}\left(X_i \geq e\frac{\epsilon n}{e}\right) \leq \frac{1}{e} \qquad (\dagger)$$

Ora, dato che le funzioni hash $H_1, \ldots, H_\ell$ sono indipendenti, anche le $Z_1, \ldots, Z_\ell$ sono indipendenti e quindi, in particolare, gli eventi $Z_i \geq n_x + \epsilon n$ ($i =$

$1, \ldots, \ell$) sono indipendenti. Questo implica che

$$\begin{aligned}
\mathbb{P}\left(\textsc{Count}(x) \geq n_x + \epsilon n\right) &= \mathbb{P}\left(\min_{i=1,\ldots,\ell} Z_i \geq n_x + \epsilon n\right) \\
&= \prod_{i=1}^{\ell} \mathbb{P}\left(Z_i \geq n_x + \epsilon n\right) \\
&= \mathbb{P}\left(\bigwedge_{i=1,\ldots,\ell} (Z_i \geq n_x + \epsilon n)\right) \\
&\leq e^{-\ell} \qquad \text{(per †)}
\end{aligned}$$

Per capire i prossimi passaggi, ricordiamo che, per qualsiasi insieme di eventi $A_1, \ldots, A_N$ vale che

$$\mathbb{P}\left(\exists i : A_i\right) = \mathbb{P}\left(A_1 \cup \cdots \cup A_N\right) \leq \sum_{i=1}^{N} \mathbb{P}(A_i)$$

Dato che vogliamo conteggi corretti con alta probabilità per ogni x nella tabella A di lunghezza n

$$\begin{aligned}
\mathbb{P}\left(\exists x \in A \mid \textsc{Count}(x) \geq n_x + \epsilon n\right) &= \mathbb{P}\left(\bigcup_{x \in A} \left(\textsc{Count}(x) \geq n_x + \epsilon n\right)\right) \\
&\leq \sum_{x \in A} \mathbb{P}\left(\textsc{Count}(x) \geq n_x + \epsilon n\right) \\
&\leq n e^{-\ell} \leq \delta
\end{aligned}$$

per $\ell \geq \ln \frac{n}{\delta}$.

Quindi, se fissiamo $\delta = 1/100$, abbiamo che $b = \Theta\left(\frac{1}{\epsilon}\right)$ e $\ell = \Theta\left(\log n\right)$. Quindi lo spazio totale utilizzato è $\Theta\left(\frac{1}{\epsilon} \log n\right)$, ovvero logaritmico nella taglia della tabella A se ϵ non dipende da n[1]. La routine $\textsc{SelectEl}(A, k)$ soddisfa le seguenti proprietà:

1. Ogni valore che compare almeno $\frac{n}{k}$ volte in A è nella lista
2. Con probabilità almeno del 99%, ogni valore nella lista compare almeno $\frac{n}{k} - \epsilon n$ volte in A

[1] In realtà dobbiamo anche contare lo spazio utilizzato dalla lista che contiene gli heavy hitters. Questo sarà di ordine $\mathcal{O}\left(k/(1 - \epsilon k)\right)$.

3.3 Proiezioni casuali

La tecnica del conteggio approssimato permette di stimare in modo efficiente la numerosità degli elementi più frequenti in una collezione. Vediamo ora come la tecnica delle proiezioni casuali permette di stimare in modo efficiente le distanze fra coppie di punti nello spazio Euclideo d-dimensionale quando d è grande.

Ricordiamo che la distanza Euclidea fra due punti $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$ è calcolata come

$$\|\boldsymbol{x} - \boldsymbol{x}'\| = \sqrt{\sum_{i=1}^{d} (x_i - x'_i)^2}$$

In molte applicazioni i dati possono essere rappresentati come vettori di numeri. Due esempi importanti sono le immagini (ogni coordinata è un pixel) e i testi (ogni coordinata è una parola del dizionario e il valore della coordinata è la frequenza con la quale la parola compare nel testo). Se consideriamo l'elenco dei film disponibili su Netflix come un dizionario, allora anche un utente di Netflix può essere visto come un vettore di numeri dove ogni coordinata è un film e il valore della coordinata rappresenta una valutazione del film da parte dell'utente.

In tutti questi casi, possiamo interpretare la vicinanza di due punti in $\mathbb{R}^d$ come una misura della similarità fra gli elementi (immagini, testi, utenti) che i punti rappresentano. Quindi la capacità di calcolare in modo efficiente qual è il punto in un insieme più vicino a un dato punto (*nearest neighbor*) diventa fondamentale per, ad esempio, suggerire film a nuovi utenti basandosi sui film apprezzati da utenti che hanno un profilo simile (ovvero, le loro codifiche in $\mathbb{R}^d$ sono vicine in termini di distanza Euclidea).

Problema Nearest Neighbor.
Istanza: Un insieme finito $S \subset \mathbb{R}^d$ e un punto $\boldsymbol{x} \in \mathbb{R}^d$.
Soluzione: $\arg\min_{\boldsymbol{x}' \in S} \|\boldsymbol{x} - \boldsymbol{x}'\|$.

Purtroppo, trovare il nearest neighbot in d dimensioni diventa computazionalmente costoso quando $d \gg 1$, come di solito succede nelle applicazioni interessanti. Per esempio, se $|S| = n$ e voglio risolvere il problema nearest neighbor calcolando le distanza fra $\boldsymbol{x}$ e i punti di S impiegherò un tempo dell'ordine di nd. Se devo risolvere il problema ogni volta che viene aggiunto un utente a S impiegherò tempo nell'ordine di $\sum_{t=1}^{n}(td) = \Theta(n^2 d)$.

Per ovviare a questo problema mostriamo che per ogni $0 < \epsilon,\ \delta < 1$ esiste $k = \mathcal{O}\left(\frac{1}{\epsilon^2} \ln \frac{|S|}{\delta}\right)$ ed esiste una classe $\mathcal{F}$ di funzioni $f : \mathbb{R}^d \to \mathbb{R}^k$ tale che

$$(1-\epsilon)\,\|\boldsymbol{x} - \boldsymbol{x}'\|^2 \leq \|f(\boldsymbol{x}) - f(\boldsymbol{x}')\|^2 \leq (1+\epsilon)\|\boldsymbol{x} - \boldsymbol{x}'\|^2, \quad \boldsymbol{x}, \boldsymbol{x}' \in S$$

con probabilità almeno $1 - \delta$ rispetto all'estrazione di $f \in \mathcal{F}$ (la distanza calcolata tramite le funzioni "spesso" è "abbastanza vicina" ai valori reali).

Per dimostrare questo risultato utilizziamo una tecnica simile al conteggio approssimato. Ovvero, usiamo k funzioni casuali analoghe alle funzioni hash del conteggio approssimato. Queste funzioni sono rappresentate da k vettori casuali $\boldsymbol{Z}_i, \ldots, \boldsymbol{Z}_k \in \mathbb{R}^d$ estratti in un modo che spiegheremo a breve. La funzione casuale associata al vettore $\boldsymbol{Z}_j$ è definita come

$$f_j(\boldsymbol{x}) = \boldsymbol{Z}_j^\top \boldsymbol{x} = \sum_{i=1}^{d} Z_{j,i} x_i$$

Il prodotto scalare $\boldsymbol{Z}_j^\top \boldsymbol{x}$ calcola la lunghezza della proiezione di $\boldsymbol{x}$ su $\boldsymbol{Z}_j$ moltiplicata per la lunghezza di $\boldsymbol{Z}_j$. L'idea è quella di calcolare una sorta di "impronta" molto più piccola del vettore originale (uno scalare al posto che d dimensioni), conservandone le proprietà essenziali, usata per fare i calcoli.

Quindi, usando la funzione f_j possiamo approssimare la distanza $\|\boldsymbol{x} - \boldsymbol{x}'\|$ fra i vettori $\boldsymbol{x}$ e $\boldsymbol{x}'$ con la differenza $|f_j(\boldsymbol{x}) - f_j(\boldsymbol{x}')|$ fra numeri reali $f_j(\boldsymbol{x})$ e $f_j(\boldsymbol{x}')$. Per ridurre l'errore di approssimazione utilizziamo k funzioni indipendenti invece di una sola.

I vettori $\boldsymbol{Z}_j$ sono ottenuti generando ciascuna componente $Z_{j,i}$ per $i = 1, \ldots, d$ con estrazioni indipendenti da una distribuzione di probabilità con media zero e varianza uno, cioè

$$\mathbb{E}[Z_{j,i}] = 0 \ \text{ e } \ \mathrm{Var}[Z_{j,i}] = 1, \qquad j = 1, \ldots, k, \ \ i = 1, \ldots, d$$

Quindi

$$\begin{aligned}
\mathbb{E}\left[\left(f_j(\boldsymbol{x}) - f_j(\boldsymbol{x}')\right)^2\right] &= \mathbb{E}\left[\left(\sum_{i=1}^{d}\left(x_i - x_i'\right) Z_{j,i}\right)^2\right] && \text{(def. } f_j\text{)} \\
&= \mathbb{E}\left[\sum_{r=1}^{d}\sum_{s=1}^{d}\left(x_r - x_r'\right)\left(x_s - x_s'\right) Z_{j,r} Z_{j,s}\right] \\
&= \mathbb{E}\left[\sum_{i=1}^{d}\left(x_i - x_i'\right)^2 Z_{j,i}^2\right] \\
&\quad + \mathbb{E}\left[\sum_{r,s:r\neq s}\left(x_r - x_r'\right)\left(x_s - x_s'\right) Z_{j,r} Z_{j,s}\right] \\
&= \sum_{i=1}^{d}\left(x_i - x_i'\right)^2 \mathbb{E}\left[Z_{j,i}^2\right] \\
&\quad + \sum_{r,s=1}^{d}\left(x_r - x_r'\right)\left(x_s - x_s'\right) \underbrace{\mathbb{E}\left[Z_{j,r}\right]\mathbb{E}[Z_{j,s}]}_{\text{media 0}} \\
&= \sum_{i=1}^{d}\left(x_i - x_i'\right)^2 \operatorname{Var}\left[Z_{j,i}\right] + 0 && (*) \\
&= \sum_{i=1}^{d}\left(x_i - x_i'\right)^2 && (\operatorname{Var}\left[Z_{j,i}\right] = 1) \\
&= \|\boldsymbol{x} - \boldsymbol{x}'\|^2
\end{aligned}$$

dove $(*)$ vale perché le $Z_{j,i}$ hanno media zero e inoltre

$$\operatorname{Var}\left[Z_{j,i}\right] = \mathbb{E}\left[\left(Z_{j,i} - \mathbb{E}\left[Z_{j,i}\right]\right)^2\right] = \mathbb{E}\left[Z_{j,i}^2\right]$$

Questo dimostra che posso usare $\left(f_j(\boldsymbol{x}) - f_j(\boldsymbol{x}')\right)^2$ per stimare la distanza quadrata $\|\boldsymbol{x} - \boldsymbol{x}'\|^2$ (abbiamo dimostrato che le approssimazioni create mantengono le misure relative dei vettori).

Definiamo ora la proiezione casuale $f : \mathbb{R}^d \to \mathbb{R}^k$

$$f(\boldsymbol{x}) = \left(\frac{f_1(\boldsymbol{x})}{\sqrt{k}}, \ldots, \frac{f_k(\boldsymbol{x})}{\sqrt{k}}\right)$$

Si noti che $f(\boldsymbol{x}) = M\boldsymbol{x}$ dove M è la matrice casuale $k \times d$ avente $\boldsymbol{Z}_1/\sqrt{k}, \ldots, \boldsymbol{Z}_k/\sqrt{k}$ come righe. Questo implica che f è una trasformazione lineare, ovvero $f(a\boldsymbol{x} + b\boldsymbol{x}') = af(\boldsymbol{x}) + bf(\boldsymbol{x}')$ per ogni $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$ e $a, b \in \mathbb{R}$. Quindi

$$\mathbb{E}\left[\|f(\boldsymbol{x}) - f(\boldsymbol{x}')\|^2\right] = \mathbb{E}\left[\|f(\boldsymbol{x} - \boldsymbol{x}')\|^2\right] = \frac{1}{k}\sum_{j=1}^{k}\mathbb{E}\left[f_j(\boldsymbol{x} - \boldsymbol{x}')^2\right] = \|\boldsymbol{x} - \boldsymbol{x}'\|^2$$

dato che ciascun f_j è uno stimatore della distanza quadrata (stiamo facendo la media delle k distanze calcolate, per ridurre l'errore).

Ora, detto $\boldsymbol{v} = \boldsymbol{x} - \boldsymbol{x}'$ e usando sempre il fatto che f è lineare

$$\frac{\|f(\boldsymbol{x}) - f(\boldsymbol{x}')\|^2}{\|\boldsymbol{x} - \boldsymbol{x}'\|^2} = \left\| f\left(\frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}\right)\right\|^2$$

Quindi, se vogliamo dimostrare che

$$(1-\epsilon)\|\boldsymbol{x} - \boldsymbol{x}'\|^2 \leq \|f(\boldsymbol{x}) - f(\boldsymbol{x}')\|^2 \leq (1+\epsilon)\|\boldsymbol{x} - \boldsymbol{x}'\|^2 \qquad \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$$

possiamo equivalentemente dimostrare che

$$1 - \epsilon \leq \|f(\boldsymbol{v})\|^2 \leq 1 + \epsilon$$

per ogni $\boldsymbol{v} \in \mathbb{R}^d$ tale che $\|\boldsymbol{v}\| = 1$.

A questo punto ci serve fare assunzioni sulla distribuzione delle variabili casuali $Z_{j,i}$. Assumiamo quindi che le $Z_{j,i}$ abbiano una distribuzione Normale (ovvero, Gaussiana con media zero e varianza uno). Per le proprietà della Normale, vale che per ogni ϵ, $\delta > 0$ fissati e per ogni $\boldsymbol{v} \in \mathbb{R}^d$ di norma unitaria

$$\mathbb{P}\left(\left|\|f(\boldsymbol{v})\|^2 - 1\right| > \epsilon\right) \leq \delta, \quad \text{per } k = \mathcal{O}\left(\frac{1}{\epsilon^2}\ln\frac{1}{\delta}\right) \tag{‡}$$

dove la probabilità è calcolata rispetto all'estrazione delle $\{Z_{j,i} \mid j = 1, \ldots, k, \quad i = 1, \ldots, d\}$.

Si noti che

$$\|f(\boldsymbol{v})\|^2 = \frac{1}{k}\sum_{j=1}^{k}\left(\boldsymbol{Z}_j^\top \boldsymbol{v}\right)^2$$

e inoltre

$$\mathbb{E}\left[\left(\boldsymbol{Z}_j^\top \boldsymbol{v}\right)^2\right] = \boldsymbol{v}^\top \mathbb{E}\left[\boldsymbol{Z}_j \boldsymbol{Z}_j^\top\right]\boldsymbol{v} = \boldsymbol{v}^\top I \boldsymbol{v} = \|\boldsymbol{v}\|^2 = 1$$

dove abbiamo usato il fatto che la matrice $M = \mathbb{E}\left[\boldsymbol{Z}_j \boldsymbol{Z}_j^\top\right]$ ha componenti $M_{r,s} = \mathbb{E}\left[Z_{j,r} Z_{j,s}\right]$ tali che

$$M_{r,s} = \begin{cases} 0 & \text{se } r \neq s \\ 1 & \text{altrimenti} \end{cases}$$

Quindi le variabili casuali $V_j = \left(\boldsymbol{Z}_j^\top\right)^2$ per $j = 1, \ldots, k$ sono i.i.d. (indipendenti e identicamente distribuite) con media $\mu = 1$ e (‡) può essere riscritta come

$$\mathbb{P}\left(\left|\frac{1}{k}\sum_{j=1}^{k} V_j - \mu\right| > \epsilon\right) \leq e^{-\mathcal{O}(k\epsilon^2)}$$

Questa diseguaglianza è analoga al Lemma di Chernoff-Hoeffding (2.1.1), con l'unica differenza che qui le V_j non hanno valori limitati. La formula (‡) ci dice quindi che un risultato analogo al Lemma di Chernoff-Hoeffding vale anche per variabili casuali del tipo $\left(\boldsymbol{Z}_j^\top \boldsymbol{v}\right)^2$ dove $\boldsymbol{Z}_j$ sono Normali multivariate e $\|\boldsymbol{v}\| = 1$.

Per capire i prossimi passaggi ricordiamo che, per qualsiasi insieme di eventi $A_1, \ldots, A_N$ vale che

$$\mathbb{P}\left(\exists i : A_i\right) = \mathbb{P}\left(A_1 \cup \cdots \cup A_N\right) \leq \sum_{i=1}^{N} \mathbb{P}\left(A_i\right) \qquad (*)$$

Nel nostro caso, ci interessano gli eventi

$$A_{\boldsymbol{x},\boldsymbol{x}'} = \left|\frac{\|f(\boldsymbol{x}) - f(\boldsymbol{x}')\|^2}{\|\boldsymbol{x} - \boldsymbol{x}'\|^2} - 1\right| > \epsilon$$

per ognuna delle $N = \binom{n}{2} \leq n^2$ coppie di punti distinti $\boldsymbol{x}, \boldsymbol{x}' \in S$. Allora, dato un qualunque insieme $S \subset \mathbb{R}^d$ di n punti

$$\begin{aligned} \mathbb{P}\left(\exists \boldsymbol{x}, \boldsymbol{x}' \in S : A_{\boldsymbol{x},\boldsymbol{x}'}\right) &= \mathbb{P}\left(\bigcup_{\boldsymbol{x},\boldsymbol{x}' \in S} A_{\boldsymbol{x},\boldsymbol{x}'}\right) \\ &\leq \sum_{\boldsymbol{x},\boldsymbol{x}' \in S} \mathbb{P}\left(A_{\boldsymbol{x},\boldsymbol{x}'}\right) \qquad \text{(per } (*)\text{)} \\ &\leq \sum_{\boldsymbol{x},\boldsymbol{x}' \in S} \delta \leq n^2 \delta \end{aligned}$$

per $k = \mathcal{O}\left(\frac{1}{\epsilon^2} \ln \frac{1}{\delta}\right)$.

Da questo ne deduciamo che, per $k = \mathcal{O}\left(\frac{1}{\epsilon^2} \ln \frac{n}{\delta}\right)$ vale

$$(1-\epsilon)\|\boldsymbol{x} - \boldsymbol{x}'\|^2 \leq \|f(\boldsymbol{x}) - f(\boldsymbol{x}')\|^2 \leq (1+\epsilon)\|\boldsymbol{x} - \boldsymbol{x}'\|^2, \qquad \forall \boldsymbol{x}, \boldsymbol{x}' \in S \ \ (\star)$$

con probabilità almeno $1 - \delta$ rispetto all'estrazione delle variabili casuali $\{Z_{j,i} \mid j = 1, \ldots, k, \ \ i = 1, \ldots, d\}$.

Se ci accontentiamo di un errore nella stima delle distanze del 10% con probabilità del 99% rispetto all'estrazione di tutte le $Z_{j,i}$, allora ϵ e δ sono costanti e quindi $k = \mathcal{O}(\log n)$. Il costo per mappare i punti di S in $\mathbb{R}^k$ è $ndk = nd \ln n$ e il costo per calcolare le coppie di distanze fra un $\boldsymbol{x}$ e i punti in S è $n \ln n$. Se devo risolvere il problema nearest neighbor approssimato n volte impiegherò quindi un tempo dell'ordine di $nd \ln n + n^2 \ln n \leq n^2 d$ quando $n = \mathcal{O}(2^d)$.

Se avessimo al più $s < k$ valori non nulli in ciascuna colonna di M, allora il costo per mappare un punto di S in $\mathbb{R}^k$ sarebbe ds. È possibile dimostrare che $(\star)$ vale per $k = \mathcal{O}\left(\frac{1}{\epsilon^2} \ln \frac{n}{\delta}\right)$ e $s = \mathcal{O}\left(\frac{1}{\epsilon} \ln \frac{n}{\delta}\right)$.

Capitolo 4

Clustering e Randomizzazione

4.1 Correlation Clustering

Il clustering è un problema fondamentale all'interno dell'apprendimento non supervisionato. Un problema di clustering è tipicamente rappresentato tramite un insieme di elementi e una misura di similarità (o dissomiglianza) definita sugli elementi dell'insieme. Quando gli elementi sono punti in uno spazio metrico, la dissomiglianza può essere misurata tramite una funzione distanza. In un contesto più generico, quando gli elementi su cui fare clustering sono membri di un insieme astratto V, la similarità è definita da una funzione simmetrica arbitraria σ, definita su coppie di elementi distinti in V.

Il Correlation Clustering (CC) è un caso speciale noto, nel quale σ è una funzione con valori in $\{-1, +1\}$, la quale stabilisce se due elementi distinti di V sono simili o meno. L'obiettivo del CC è di raggruppare i punti in V in modo da minimizzare il numero di errori, dove un errore è dato da ogni coppia di elementi con similarità -1 appartenenti allo stesso cluster, oppure ogni coppia di elementi con similarità $+1$ appartenenti a cluster diversi. Non ci sono limitazioni sul numero di cluster o la loro dimensione: tutte le partizioni di V, incluse quelle triviali, sono valide. Dati V e σ, l'errore dato da un clustering ottimale è chiamato *Correlation Clustering Index* (CCI), indicato con OPT.

Si noti che OPT $= 0$ indica che V può essere partizionato perfettamente: ogni coppia di elementi nello stesso cluster ha similarità $+1$ e ogni coppia di elementi appartenente a cluster differenti ha similarità -1. Sin dalla sua introduzione, CC ha attratto molto interesse e trova numerose applicazioni in ambiti come risoluzione delle entità, analisi delle immagini e analisi dei social media. Minimizzare l'errore del CC è difficile e il migliore algoritmo efficiente trovato finora raggiunge un fattore di approssimazione di 2 (soluzione al più $2 \cdot \text{OPT}$; in realtà appena meno).

Una semplice e elegante soluzione per approssimare CC è KwikCluster. Ad ogni iterazione KwikCluster:

- Sceglie un pivot casuale π_r da V
- Calcola tutte le similarità tra π_r e ogni altro nodo in V
- Crea un cluster C contenente π_r e tutti i punti u tali che $\sigma(\pi_r, u) = +1$

L'algoritmo poi chiama se stesso sull'insieme $V \setminus C$. Per ogni istanza di CC, KwikCluster ottiene un fattore di approssimazione al massimo di 3 (3OPT).

Algoritmo 7: KwikCluster(V_r, r)

Input: Insieme di nodi rimanenti V_r, indice di round r

if $|V_r| = 0$ **then**
  return
if $|V_r| = 1$ **then**
  output il cluster singleton V_r
  return
Scegli un pivot π_r unif. a caso da V_r
$C_r \leftarrow \{\pi_r\}$ `// Crea nuovo cluster con il pivot`
$C_r \leftarrow C_r \cup \{u \in V_r \mid \sigma(\pi_r, u) = +1\}$ `// Popola il cluster`
Output cluster C_r
KwikCluster($V_r \setminus C_r, r + 1$)

Chiamiamo $V \equiv \{1, \ldots, n\}$ il set di nodi in input, $\mathcal{E} \equiv \binom{V}{2}$ il set di tutte le coppie $\{u, v\}$ tali che $u, v \in V$ e $u \neq v$ (coppie di nodi distinti in V), $\sigma : \mathcal{E} \to \{-1, +1\}$ la funzione di similarità binaria. Un clustering $\mathcal{C}$ è una partizione di V in cluster disgiunti $C_i : i = 1, \ldots, k$. Dati $\mathcal{C}$ e σ, l'insieme $\Gamma_{\mathcal{C}}$ di lati sbagliati contiene tutte le coppie $\{u, v\}$ che causano un errore, ovvero tali che $\sigma(u, v) = -1$ e u, v appartengono allo stesso $C_i \in \mathcal{C}$, oppure

tali che $\sigma(u, v) = +1$ e u, v appartengono a due diversi cluster di $\mathcal{C}$. Il costo della partizione $\mathcal{C}$ è $|\Gamma_{\mathcal{C}}|$. Il CCI è OPT $= \min_{\mathcal{C}} |\Gamma_{\mathcal{C}}|$, ovvero la partizione con costo minimo possibile.

Un triangolo è una qualunque tripla non ordinata $T = \{u, v, w\} \subseteq V$. Chiamiamo $e = \{u, w\}$ un qualsiasi lato di un triangolo; scriviamo $e \subset T$ e $v = T \setminus e$. Diciamo che T è un *triangolo sgradevole* (in originale *bad triangle*, non sono sicuro di come tradurlo) se i segni dei valori dati da $\sigma(u, v)$, $\sigma(u, w)$ e $\sigma(v, w)$ sono $\{+, +, -\}$ (l'ordine è irrilevante).

Chiamiamo $\mathcal{T}$ l'insieme di tutti i triangoli sgradevoli presenti all'interno di V e definiamo $\mathcal{T}(e) \equiv \{T \in \mathcal{T} \mid e \subset T\}$. Si può facilmente vedere come il numero di triangoli sgradevoli senza lati in comune è un lower bound per OPT: indipendentemente da come vengono divisi i nodi, un triangolo sgradevole aumenta di almeno 1 il costo della partizione.

Il lemma seguente mostra come anche la somma pesata di tutti i triangoli sgradevoli è un lower bound per OPT, ammesso che la somma dei pesi di tutti i triangoli sgradevoli che incidono su qualunque singolo lato sia al più 1.

Lemma 4.1.1. *Se $\{\beta_T \geq 0 \mid T \in \mathcal{T}\}$ è un insieme di pesi sui triangoli sgradevoli tale che $\sum_{T \in \mathcal{T}(e)} \beta_T \leq 1$ per ogni $e \in \mathcal{E}$, allora $\sum_{T \in \mathcal{T}} \beta_T \leq$ OPT.*

Dimostrazione. Omessa. □

Cerchiamo ora un bound per l'errore atteso di KwikCluster. Chiamiamo V_r l'insieme dei nodi rimanenti all'inizio della r-esima chiamata ricorsiva.

Sia Γ_A l'insieme di lati sbagliati per il clustering emesso da KwikCluster e sia $|\Gamma_A|$ il costo di tale partizione.

Lemma 4.1.2. *Per ogni $e \in \mathcal{E}$, $e \in \Gamma_A$ se e solo se esiste una chiamata ricorsiva r e un $T \in \mathcal{T}$ tale che $T \subseteq V_r$, $T \in \mathcal{T}(e)$ e $\pi_r = T \setminus e$.*

In altre parole, una lato e causa un errore se e solo se al passo r dell'algoritmo c'è un triangolo sgradevole T formato dal pivot π_r e dai nodi di e.

Dimostrazione. Si scelga qualsiasi lato e e sia r l'iterazione in cui almeno uno dei due nodi di e viene rimosso da V_r. Allora, KwikCluster fa un errore su e se e solo se e forma un triangolo sgradevole con π_r e $\pi_r = T \setminus e$.

Quindi, se $e \in \Gamma_A$ allora esiste un'iterazione r e un triangolo $T \subseteq V_r$ tale che: $\pi_r = T \setminus e$ e $T \in \mathcal{T}(e)$. Proviamo l'implicazione inversa analizzando

caso per caso $e = \{u, w\}$. Assumiamo $T = \{u, \pi_r, w\} \subseteq V_r$, $T \in \mathcal{T}(e)$ e $\pi_r = T \setminus e$.

Caso 1: $\sigma(u, w) = +1$: dato che $T \in \mathcal{T}$, $\sigma(\pi_r, w) \neq \sigma(\pi_r, u)$. Ma allora u e w devono finire in cluster differenti per costruzione dell'algoritmo, di conseguenza e è un errore.

Caso 2: $\sigma(u, w) = -1$: dato che $T \in \mathcal{T}$, $\sigma(\pi_r, w) = \sigma(\pi_r, u) = +1$. Ma allora u e w finiscono nello stesso cluster, di conseguenza e è un errore. □

Il Lemma 4.1.2 implica che all'iterazione r viene fatto un errore su esattamente uno dei lati di ogni $T \in \mathcal{T}$ tale che $T \subseteq V_r$ e $\pi_r \in T$. Ricordiamo che ogni triangolo sgradevole può causare un solo errore, dato che poi il pivot $\pi_r \in T$ viene rimosso da V_r. Quindi, per una qualsiasi sequenza di pivot $\pi_1, \pi_2, \ldots$, abbiamo che

$$|\Gamma_A| = \sum_{T \in \mathcal{T}} \mathbb{I}\{(\exists r) \mid T \subseteq V_r \wedge \pi_r \in T\}$$

Per ogni $T \in \mathcal{T}$, sia A_T l'evento $\{(\exists r) \mid T \subseteq V_r \wedge \pi_r \in T\}$ che indica il contributo a un singolo errore da parte di T.

Si noti che per ogni $e \in \Gamma_A$ e per ogni coppia di $T, T' \in \mathcal{T}(e)$ con $T \neq T'$, A_T e A'_T non possono accadere assieme in quanto e può formare un triangolo sgradevole assieme a π_r in solo uno dei due casi. Quindi, per un qualsiasi lato e

$$\sum_{T \in \mathcal{T}(e)} \mathbb{I}\{A_T \wedge e \in \Gamma_A\} = 1$$

Prendendo in considerazione il valore atteso dato dalla sequenza casuale di pivot

$$1 = \sum_{T \in \mathcal{T}(e)} \mathbb{P}(A_T \wedge e \in \Gamma_A) = \sum_{T \in \mathcal{T}(e)} \mathbb{P}(e \in \Gamma_A \mid A_T)\, \mathbb{P}(A_T) = \sum_{T \in \mathcal{T}(e)} \frac{1}{3} \mathbb{P}(A_T)$$

dove $\mathbb{P}(e \in \Gamma_A | A_T) = \frac{1}{3}$ considerato che, dato r tale che $T \subseteq V_r$ e $\pi_r \in T$, e è sbagliato solo se $\pi_r \in T \setminus e$.

Applicando il Lemma 4.1.1 con $\beta_T = \frac{1}{3}\mathbb{P}(A_T)$ per ogni $T \in \mathcal{T}$ si ottiene

$$\mathbb{E}[|\Gamma_A|] = \sum_{T \in \mathcal{T}} \mathbb{P}(A_T) = 3 \sum_{T \in \mathcal{T}} \beta_T \leq 3\mathrm{OPT}$$

4.2 k-Means

Consideriamo il problema di partizionare un insieme finito $\mathcal{X} \subset \mathbb{R}^d$ di punti in $k > 1$ cluster. La similarità tra punti può essere misurata come distanza Euclidea, dato che siamo in $\mathbb{R}^d$. Identifichiamo ogni cluster $i \in \{1, \ldots, k\}$ tramite il relativo centro $\boldsymbol{c}_i \in \mathbb{R}^d$ (non è necessario che $\boldsymbol{c}_i \in \mathcal{X}$) e assegniamo ogni punto $\boldsymbol{x} \in \mathcal{X}$ al centro più vicino (rispetto alla distanza Euclidea).

Il costo di un punto all'interno di una partizione (clustering) $\mathcal{C} = \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k\}$ è $\phi(\mathcal{C}, \boldsymbol{x}) = \min_{i=1,\ldots,k} \|\boldsymbol{x} - \boldsymbol{c}_i\|^2$.

Il costo della partizione $\mathcal{C}$ è $\Phi(\mathcal{C}) = \sum_{\boldsymbol{x} \in \mathcal{X}} \phi(\mathcal{C}, \boldsymbol{x})$.

Si noti che ogni punto "costa" quanto il quadrato della distanza dal centro più vicino. Il k-clustering ottimale $\mathcal{C}^*$ è una qualsiasi scelta di centri (partizione) tale che minimizzi il costo, ovvero

$$\mathcal{C}^* = \arg \min_{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k \in \mathbb{R}^d} \Phi\left(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k\right)$$

Non è detto che l'insieme di centri ottimali sia unico. Denotiamo con $\mathrm{OPT}(\mathcal{X})$ il costo di $\mathcal{C}^*$.

> Problema k-MEANS.
> *Istanza:* insieme $\mathcal{X} \subset \mathbb{R}^d$, parametro $k > 1$.
> *Soluzione:* qualsiasi $\mathcal{C} \subset \mathbb{R}^d$ con $|C| = k$ tale che $\Phi(\mathcal{C}) = \mathrm{OPT}(\mathcal{X})$.

Il problema è triviale per $k = 1$, essendoci un unico centro $\boldsymbol{c}^*$ che minimizza il costo, il quale corrisponde al centroide dell'insieme $\mathcal{X}$

$$\boldsymbol{c}^* = \arg \min_{\boldsymbol{c} \in \mathbb{R}^d} \sum_{\boldsymbol{x} \in \mathcal{X}} \|\boldsymbol{x} - \boldsymbol{c}\|^2 = \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{x} \in \mathcal{X}} \boldsymbol{x}$$

Questo può essere dimostrato notando che $F(\boldsymbol{c}) = \sum_{\boldsymbol{x} \in \mathcal{X}} \|\boldsymbol{x} - \boldsymbol{c}\|^2$ è una funzione convessa con minimo nel punto in cui $\boldsymbol{c}$ è il centroide. Questo implica che $\mathcal{C}^*$ è formato dai centroidi dei cluster che lo compongono.

Il problema di k-means assume implicitamente che i punti in $\mathcal{X}$ siano campionati da k distribuzioni Gaussiane sferiche $\mathcal{N}(\boldsymbol{\mu}_i, \sigma_i^2 I)$ per $i = 1, \ldots, k$ le quali medie $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ sono i centri e le quali varianze $\sigma_1^2, \ldots, \sigma_k^2$ sono upper bound per il costo ottimale

$$\boldsymbol{\mu}_i = \arg \min_{\boldsymbol{c}} \mathbb{E}\left[\|\boldsymbol{X} - \boldsymbol{c}\|^2\right], \text{ dove } \boldsymbol{X} \sim \mathcal{N}\left(\boldsymbol{\mu}_i, \sigma_i^2 I\right) \text{ e } \mathbb{E}\left[\Phi\left(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\right)\right] \leq \sum_{i=1}^{k} \sigma_i^2$$

Risolvere il problema in $\mathbb{R}^d$ è $\mathcal{NP}$-hard anche per $k = 2$ (quando $d = 2n$). Di conseguenza, il migliore algoritmo esatto per risolvere k-means si basa su

1. Enumerare tutte le $k^{|\mathcal{X}|}$ partizioni di $\mathcal{X}$ in k componenti
2. Calcolare i centroidi $\mathcal{C} = \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k\}$ per i k elementi della partizione
3. Calcolare il costo $\Phi(\mathcal{C})$ della partizione

Indichiamo con $\mathcal{X}(i)$ il sottoinsieme di punti di $\mathcal{X}$ tali che hanno $\boldsymbol{c}_i$ come centro più vicino

$$\mathcal{X}(i) = \left\{ \boldsymbol{x} \in \mathcal{X} \mid \arg \min_{j=1,\ldots,k} \|\boldsymbol{x} - \boldsymbol{c}_j\|^2 = i \right\}$$

L'algoritmo seguente è l'euristica più comune per risolvere k-means.

Algoritmo 8: Algoritmo di Lloyd

Input: Insieme finito $X \subset \mathbb{R}^d$, parametro k t.c. $1 < k < |\mathcal{X}|$

Scegliere uniformemente a caso k punti da $\mathcal{X}$
repeat
  for $\boldsymbol{x} \in \mathcal{X}$ **do**
    Assegna $\boldsymbol{x}$ al cluster C_i, con $i = \arg\min_{j=1,\ldots,k} = \|\boldsymbol{x} - c_j\|^2$
  for $i = 1, \ldots, k$ **do**
    $\boldsymbol{c}_i \leftarrow \dfrac{1}{|C_i|} \displaystyle\sum_{\boldsymbol{x} \in \mathcal{X}(i)} \boldsymbol{x}$ `// c_i è il centroide di C_i`
until $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k$ *rimangono invariati*

Il tempo per ogni iterazione dell'algoritmo è nell'ordine di $\mathcal{O}(nkd)$. Si possono usare proiezioni casuali per mappare $\mathcal{X}$ a $\mathbb{R}^N$ con $N = \Theta(\ln n)$, risultando in un valore di OPT moltiplicato per una costante. Questo riduce il tempo per iterazione a $\mathcal{O}(nk \ln n)$. Il caso peggiore per quanto riguarda il numero di iterazioni dell'algoritmo, purtroppo, è $2^{\Omega(\sqrt{n})}$.

Anche se funziona bene in casi concreti, l'algoritmo di Lloyd non approssima OPT per nessuna costante.

Teorema 4.2.1. *Per ogni $a > 1$ esistono istanze 1-dimensionali $\mathcal{X} \subset \mathbb{R}$ di k-means con $k = 3$ per le quali l'algoritmo di Lloyd restituisce una partizione $\mathcal{C}$ tale che $\Phi(\mathcal{C}) \geq a \cdot \mathrm{OPT}$ con probabilità arbitrariamente vicina a 1.*

Dimostrazione. Scelto un $a > 1$ e sia $\mathcal{X}$ di dimensione n tale che $n-2$ punti sono spaziati equamente all'interno del segmento $[0, 1]$, mentre i due punti rimanenti (outliers) sono posizionati a $2\sqrt{an}$ e $3\sqrt{an}$

La probabilità che l'algoritmo di Lloyd non scelga entrambi gli outlier come centri iniziali è calcolata come segue: ci sono $\binom{n}{3}$ modi possibili per scegliere 3 punti su n e $n-2$ modi di scegliere 3 punti tali che tra questi ci siano i due outlier. Quindi la probabilità è

$$p_n = 1 - \frac{n-2}{\binom{n}{3}} = 1 - \frac{(n-3)!6(n-2)}{n!})1 - \frac{6}{n(n-1)}$$

Consideriamo quindi il caso sfavorevole in cui l'algoritmo sceglie inizialmente al più uno degli outlier. In questo caso, l'algoritmo termina con almeno due centri all'interno di $[0, 1]$ è al più un centro in $\frac{5}{2}\sqrt{an}$. Il costo $\Phi(\mathcal{C})$ di questa partizione $\mathcal{C}$ è almeno $\frac{an}{2}$, mentre il costo del cluster ottimo (due centri sugli outlier e il rimanente su $1/2$) è $\mathrm{OPT} = \frac{n-2}{4}$.

Quindi, $\Phi(\mathcal{C})/\mathrm{OPT} = \Omega(a)$. Al crescere di n, ovvero per $n \to \infty$, abbiamo che $p_n \to 1$, con la conseguenza che il caso sfavorevole accade con probabilità arbitrariamente alta. □

Ora mostriamo come, se i centri si muovono, il valore di Φ decresce strettamente e può farlo al più $\mathcal{O}(k^n)$ volte, ovvero il numero di possibili partizioni in k componenti di $\mathcal{X}$ con $|\mathcal{X}| = n$.

Lemma 4.2.1. *Se in una iterazione un qualsiasi centro viene mosso, allora* Φ *decresce strettamente.*

Dimostrazione. Faremo uso del fatto seguente. Per ogni $C \subset \mathbb{R}^d$ finito e per ogni $\boldsymbol{c} \in \mathbb{R}^d$

$$\sum_{\boldsymbol{x} \in C} \|\boldsymbol{x} - \boldsymbol{c}\|^2 = |C|\|\boldsymbol{c} - \boldsymbol{\mu}\|^2 + \sum_{\boldsymbol{x} \in C} \|\boldsymbol{x} - \boldsymbol{\mu}\|^2 \qquad (\dagger)$$

dove $\boldsymbol{\mu}$ è il centroide di C.

Siano $C_1, \ldots, C_k$ e $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k$ rispettivamente cluster e centri all'inizio di una iterazione (Linea 2) e siano $C'_1, \ldots, C'_k$ e $\boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_k$ cluster e centri al termine

dell'iterazione (Linea 7). Sia

$$\psi\left(C_1,\ldots,C_k,\boldsymbol{c}_1,\ldots,\boldsymbol{c}_k\right)=\sum_{i=1}^{k}\sum_{\boldsymbol{x}\in C_i}\|\boldsymbol{x}-\boldsymbol{c}_i\|^2$$

Si noti che $\psi\left(C_1,\ldots,C_k,\boldsymbol{c}_1,\ldots,\boldsymbol{c}_k\right)\geq\psi\left(C'_1,\ldots,C'_k,\boldsymbol{c}_1,\ldots,\boldsymbol{c}_k\right)$ dato che Linea 4 assegna ogni punto al centro più vicino.

Ora, se $\boldsymbol{c}'_i\neq\boldsymbol{c}_i$ per qualche i, allora

$$\psi\left(C'_1,\ldots,C'_k,\boldsymbol{c}_1,\ldots,\boldsymbol{c}_k\right)>\psi\left(C'_1,\ldots,C'_k,\boldsymbol{c}'_1,\ldots,\boldsymbol{c}'_k\right)$$

Per dimostrarlo, ricordando che $\boldsymbol{c}'_i$ è il centroide di C'_i,

$$\sum_{\boldsymbol{x}\in C'_i}\|\boldsymbol{x}-\boldsymbol{c}_i\|^2\overset{(\dagger)}{=}|C'|\|\boldsymbol{c}_i-\boldsymbol{c}'_i\|^2+\sum_{\boldsymbol{x}\in C'_i}\|\boldsymbol{x}-\boldsymbol{c}'_i\|^2>\sum_{\boldsymbol{x}\in C'_i}\|\boldsymbol{x}-\boldsymbol{c}'_i\|^2$$

usando (†) nel primo passaggio e $\boldsymbol{c}_i\neq\boldsymbol{c}'_i$ nel secondo. Di conseguenza

$$\begin{aligned}\Phi\left(C_1,\ldots,C_k\right)&=\psi\left(C_1,\ldots,C_k,\boldsymbol{c}_1,\ldots,\boldsymbol{c}_k\right)\\&>\psi\left(C'_1,\ldots,C'_k,\boldsymbol{c}'_1,\ldots,\boldsymbol{c}'_k\right)\qquad=\Phi\left(C'_1,\ldots,C'_k\right)\end{aligned}$$

□

Questo implica anche il seguente risultato.

Teorema 4.2.2. *L'algoritmo di Lloyd termina in al più $k^{|\mathcal{X}|}$ iterazioni per qualsiasi input $(\mathcal{X},k)$.*

Dimostrazione. Si può notare che Φ è una funzione applicata alla partizione corrente $\{C_1,\ldots,C_k\}$, tale partizione può assumere al più k^n valori distinti. Inoltre, l'algoritmo non termina solo se l'iterazione corrente ha modificato la partizione. Dato che Φ può solo decrescere quando la partizione viene modificata, l'algoritmo deve terminare in al più k^n iterazioni. □

4.3 k-Means++

Ricordando il problema di k-means (4.2): dato un insieme $\mathcal{X}\subset\mathbb{R}^d$ di dimensione n e $1<k<n$ trovare

$$\mathcal{C}^*\in\arg\min_{\boldsymbol{c}_1,\ldots,\boldsymbol{c}_k\in\mathbb{R}^d}\Phi\left(\boldsymbol{c}_1,\ldots,\boldsymbol{c}_k\right)$$

dove, per ogni $\mathcal{C} \subset \mathbb{R}^d$

$$\Phi(\mathcal{C}) = \sum_{\boldsymbol{x} \in \mathcal{X}} \phi\left(\mathcal{C}, \boldsymbol{x}\right) = \sum_{\boldsymbol{x} \in \mathcal{X}} \min_{\boldsymbol{c}_i \in \mathcal{C}} \|\boldsymbol{x} - \boldsymbol{c}_i\|^2$$

Sia $\mathrm{OPT} = \Phi(\mathcal{C}^*)$ e, per ogni $\mathcal{C} \subset \mathbb{R}^d$ e $A \subseteq \mathcal{X}$, sia

$$\phi\left(\mathcal{C}, A\right) = \sum_{\boldsymbol{x} \in A} \phi\left(\mathcal{C}, \boldsymbol{x}\right)$$

Si può identificare una partizione/clustering attraverso i suoi centri $\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k\}$ oppure attraverso i cluster che la compongono $\{C_1, \ldots, C_k\}$. Si noti che, per ogni partizione $\mathcal{C}$ emessa dall'algoritmo di Lloyd (Algoritmo 8), inclusa la soluzione ottima $\mathcal{C}^*$

$$\phi(\mathcal{C}, C) = \sum_{\boldsymbol{x} \in C} \|\boldsymbol{c} - \mu_C\|^2, \quad \forall C \in \mathcal{C}, \text{ con } \mu_C \text{ centroide di } C \qquad (\dagger)$$

Come già visto (Teorema 4.2.1) l'algoritmo di Lloyd non ha un fattore di approssimazione massimo in quanto gli outlier non vengono considerati nella fase di inizializzazione dell'algoritmo, nonostante possano rappresentare un costo significativo nella soluzione.

Algoritmo 9: k-means++

Input: Insieme finito di punti $\mathcal{X} \subset \mathbb{R}^d$, parametro $1 < k < |\mathcal{X}|$

1 Scegliere uniformemente a caso un centro $\boldsymbol{c}_1$ da $\mathcal{X}$ e sia $\mathcal{C}_1 = \{\boldsymbol{c}_1\}$

2 **for** $i = 2, \ldots, k$ **do**

3 Scegliere $\boldsymbol{c}_i$ da $\mathcal{X}$ secondo la distribuzione

$$\mathbb{P}\left(\boldsymbol{c}_i = \boldsymbol{x} \mid C_{i-1}\right) = \frac{\phi\left(\mathcal{C}_{i-1}, \boldsymbol{x}\right)}{\Phi\left(\mathcal{C}_{i-1}\right)}$$

4 $\mathcal{C}_i = \mathcal{C}_{i-1} \cup \{\boldsymbol{c}_i\}$

Output: L'output dell'algoritmo di Lloyd, inizializzato con centri $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k$

In altre parole, la probabilità di scegliere un punto $\boldsymbol{x}$ come nuovo centro è proporzionale alla distanza dai centri già scelti, più un punto è lontano dai centri attuali (valore al numeratore), più alta sarà la probabilità che venga scelto.

Proveremo una versione semplificata del teorema seguente.

Teorema 4.3.1. *La partizione $\mathcal{C}$ ottenuta da k-means++ soddisfa*

$$\mathbb{E}\left[\Phi\left(\mathcal{C}\right)\right] \leq 8\left(\ln k + 2\right)\mathrm{OPT}$$

Attualmente, l'algoritmo con il migliore fattore di approssimazione utilizza un approccio basato sulla programmazione lineare e produce clustering con costo $c \cdot \mathrm{OPT}$, dove $c \in [6, 7]$.

Si consideri una qualsiasi partizione ottimale $\mathcal{C}^* = (A_1, \ldots, A_k)$ e sia $\mathcal{C}_i$ la partizione data da k-means++ dopo aver scelto i primi i centri (Linea 3).

Lemma 4.3.1. *Per ogni $A \in \mathcal{C}^*$ e per ogni $i \in [k]$*

$$\mathbb{E}\left[\phi(\mathcal{C}_i, A) \mid \boldsymbol{c}_i \in A,\ \mathcal{C}_{i-1}\right] \leq 8\phi(\mathcal{C}^*, A)$$

Dimostrazione. Si consideri $i = 1$. Allora $\mathcal{C}_{i-1} = \mathcal{C}_0 = \emptyset$ e $\boldsymbol{c}_i$ scelto secondo una distribuzione uniforme su $\mathcal{X}$, quindi possiamo dire che

$$\begin{aligned}
\mathbb{E}\left[\phi\left(\mathcal{C}_1, A\right) \mid \boldsymbol{c}_1 \in A\right] &= \frac{1}{|A|}\sum_{\boldsymbol{a}\in A}\left(\sum_{\boldsymbol{x}\in A}\|\boldsymbol{x}-\boldsymbol{a}\|^2\right) && (\mathcal{C}_1 = \{\boldsymbol{c}_1\}) \\
&\leq \frac{1}{|A|}\sum_{\boldsymbol{a}\in A}\left(|A|\|\boldsymbol{a}-\boldsymbol{\mu}\|^2 + \sum_{\boldsymbol{x}\in A}\|\boldsymbol{x}-\boldsymbol{\mu}\|^2\right) && (\mu \text{ centroide di } A) \\
&= \sum_{\boldsymbol{x}\in A}\|\boldsymbol{x}-\boldsymbol{\mu}\|^2 + \sum_{\boldsymbol{a}\in A}\|\boldsymbol{a}-\boldsymbol{\mu}\|^2 \\
&= 2\sum_{\boldsymbol{x}\in A}\|\boldsymbol{x}-\boldsymbol{\mu}\|^2 = 2\phi\left(\mathcal{C}^*, A\right) && (\text{per } \dagger)
\end{aligned}$$

In particolare, si noti che

$$\frac{1}{|A|}\sum_{\boldsymbol{a}\in A}\sum_{\boldsymbol{x}\in A}\|\boldsymbol{x}-\boldsymbol{a}\|^2 \leq 2\phi\left(\mathcal{C}^*, A\right) \qquad (\ddagger)$$

Consideriamo ora $i > 1$. Allora, per definizione dell'algoritmo

$$\mathbb{P}\left(\boldsymbol{c}_i = \boldsymbol{a} \mid \boldsymbol{a} \in A,\ \mathcal{C}_{i-1}\right) = \frac{\phi\left(\mathcal{C}_{i-1}, \boldsymbol{a}\right)}{\sum_{\boldsymbol{x}\in A}\phi\left(\mathcal{C}_{i-1}, \boldsymbol{x}\right)}$$

Per ogni $\boldsymbol{x}, \boldsymbol{a} \in A$, sia $\boldsymbol{c}$ il centro di $\mathcal{C}_{i-1}$ più vicino a $\boldsymbol{x}$. Allora

$$\begin{aligned}
\min_{j=1,\ldots,i-1}\|\boldsymbol{a}-\boldsymbol{c}_j\| &\leq \|\boldsymbol{a}-\boldsymbol{c}\| \\
&\leq \|\boldsymbol{x}-\boldsymbol{c}\| + \|\boldsymbol{a}-\boldsymbol{x}\| \quad \text{(disuguaglianza triangolare)}
\end{aligned}$$

Sapendo che $(a+b)^2 \leq 2(a^2+b^2)$ per ogni $a, b \in \mathbb{R}$ e $\|\boldsymbol{x}-\boldsymbol{c}\|^2 = \phi(\mathcal{C}_{i-1}, \boldsymbol{x})$ otteniamo

$$\begin{aligned} \|\boldsymbol{a}-\boldsymbol{c}\|^2 &\leq 2\left(\|\boldsymbol{x}-\boldsymbol{c}\|^2 + \|\boldsymbol{a}-\boldsymbol{x}\|^2\right) \\ \implies \phi(\mathcal{C}_{i-1}, \boldsymbol{a}) &\leq 2\left(\phi(\mathcal{C}_{i-1}, \boldsymbol{x}) + \|\boldsymbol{a}-\boldsymbol{x}\|^2\right) \end{aligned}$$

Facendo la media di questa disuguaglianza su tutti $\boldsymbol{x} \in A$, otteniamo che

$$\phi(\mathcal{C}_{i-1}, \boldsymbol{a}) \leq \frac{2}{|A|} \sum_{\boldsymbol{a} \in A} \left(\phi(\mathcal{C}_{i-1}, \boldsymbol{x}) + \|\boldsymbol{a}-\boldsymbol{x}\|^2\right)$$

Inoltre, per ogni $\boldsymbol{x} \in \mathcal{X}$

$$\phi(\mathcal{C}_i, \boldsymbol{x}) = \min\left\{\phi(\mathcal{C}_{i-1}, \boldsymbol{x}),\ \|\boldsymbol{x}-\boldsymbol{c}_i\|^2\right\}$$

Di conseguenza, per $\mathcal{C}_i = \mathcal{C}_{i-1} \cup \{\boldsymbol{c}_i\}$

$$\begin{aligned} &\mathbb{E}\left[\phi(\mathcal{C}_i, A) \mid \boldsymbol{c}_i \in A,\ \mathcal{C}_{i-1}\right] = \sum_{\boldsymbol{a} \in A} \frac{\phi(\mathcal{C}_{i-1}, \boldsymbol{a})}{\sum_{\boldsymbol{x} \in A} \phi(\mathcal{C}_{i-1}, \boldsymbol{x})} \phi(\mathcal{C}_i, A) \\ &\quad \leq \frac{2}{|A|} \sum_{\boldsymbol{a} \in A} \sum_{\boldsymbol{x} \in A} \frac{\phi(\mathcal{C}_{i-1}, \boldsymbol{x}) + \|\boldsymbol{a}-\boldsymbol{x}\|^2}{\sum_{\boldsymbol{x}' \in A} \phi(\mathcal{C}_{i-1}, \boldsymbol{x}')} \sum_{\boldsymbol{a}' \in A} \min\left\{\phi\left(\mathcal{C}_{i-1}, \boldsymbol{a}'\right),\ \|\boldsymbol{a}'-\boldsymbol{a}\|^2\right\} \\ &\quad = \frac{2}{|A|} \sum_{\boldsymbol{a} \in A} \frac{\sum_{\boldsymbol{x} \in A} \phi(\mathcal{C}_{i-1}, \boldsymbol{x})}{\sum_{\boldsymbol{x}' \in A} \phi(\mathcal{C}_{i-1}, \boldsymbol{x}')} \sum_{\boldsymbol{a}' \in A} \min\left\{\phi\left(\mathcal{C}_{i-1}, \boldsymbol{a}'\right),\ \|\boldsymbol{a}'-\boldsymbol{a}\|^2\right\} \\ &\qquad + \frac{2}{|A|} \sum_{\boldsymbol{a} \in A} \sum_{\boldsymbol{x} \in A} \frac{\|\boldsymbol{a}-\boldsymbol{x}\|^2}{\sum_{\boldsymbol{x}' \in A} \phi(\mathcal{C}_{i-1}, \boldsymbol{x})} \sum_{\boldsymbol{a}' \in A} \min\left\{\phi\left(\mathcal{C}_{i-1}, \boldsymbol{a}'\right),\ \|\boldsymbol{a}'-\boldsymbol{a}\|^2\right\} \\ &\quad \leq \frac{2}{A|} \sum_{\boldsymbol{a} \in A} \sum_{\boldsymbol{a}' \in A} \|\boldsymbol{a}'-\boldsymbol{a}\|^2 + \frac{2}{|A|} \sum_{\boldsymbol{a} \in A} \sum_{\boldsymbol{x} \in A} \|\boldsymbol{a}-\boldsymbol{x}\|^2 \\ &\quad = \frac{4}{|A|} \sum_{\boldsymbol{a} \in A} \sum_{\boldsymbol{x} \in A} \|\boldsymbol{x}-\boldsymbol{a}\|^2 \\ &\quad \leq 8\phi(\mathcal{C}^*, A) \qquad \text{(per ‡)} \end{aligned}$$

Questo conclude la dimostrazione. □

Un cluster $A \in \mathcal{C}^*$ è scoperto in $\mathcal{C}_i$ se $A \cap \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_i\} = \emptyset$ (nessuno dei centri selezionati da $\mathcal{C}_i$ appartiene a A). Il Lemma 4.3.1 mostra che paghiamo $\mathcal{O}(\mathrm{OPT})$ per ogni cluster ottimale che copriamo. Questo giustifica le seguenti assunzioni, ponendo il costo di ogni cluster ottimale a 1, e paghiamo

1 per ogni cluster coperto (ovvero il costo è "basso") e L per ogni cluster ottimale che rimane scoperto (costo "alto").

Assunzione 4.3.1. *Per ogni $A \in \mathcal{C}^*$:*

1. $\phi(\mathcal{C}^*, A) = 1$
2. *Per ogni $i \in [k]$, se A è coperto in $\mathcal{C}_i$, allora $\phi(\mathcal{C}_i, A) = 1$, altrimenti $\phi(\mathcal{C}_i, A) = L$*

Lemma 4.3.2. *Considerando la semplificazione di cui sopra*

$$\mathbb{E}\left[\Phi(\mathcal{C})\right] \leq (2 + \ln k)\,\mathrm{OPT}$$

Dimostrazione. Sia $\mathcal{C}_i = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_i)$. Convenzionalmente, $\mathcal{C}_0 = \emptyset$ e $\Phi(\mathcal{C}_0) = kL$ (come se ci fosse un centro di default "molto lontano"). Ora osserviamo che $\mathcal{C} = \mathcal{C}_k$

$$\Phi(\mathcal{C}_k) = \Phi(\mathcal{C}_0) + \sum_{i=0}^{k-1} \left(\Phi\left(\mathcal{C}_{i+1}\right) - \Phi(\mathcal{C}_i)\right)$$

ovvero, il costo finale è la somma di costo iniziale e variazioni a ogni passo (si tratta di una somma telescopica).

Considerando il valore atteso

$$\begin{aligned}
\mathbb{E}\left[\Phi\left(\mathcal{C}_k\right)\right] &= \Phi\left(\mathcal{C}_0\right) + \sum_{i=0}^{k-1} \left(\mathbb{E}\left[\Phi\left(\mathcal{C}_{i+1}\right)\right] - \mathbb{E}\left[\Phi\left(\mathcal{C}_i\right)\right]\right) \\
&= kL + \sum_{i=0}^{k-1} \left(\mathbb{E}\left[\Phi\left(\mathcal{C}_{i+1}\right)\right] - \mathbb{E}\left[\Phi\left(\mathcal{C}_i\right)\right]\right) \\
&= k + k(L-1) + \sum_{i=0}^{k-1} \left(\mathbb{E}\left[\Phi\left(\mathcal{C}_{i+1}\right)\right] - \mathbb{E}\left[\Phi\left(\mathcal{C}_i\right)\right]\right) \\
&= k + \sum_{i=0}^{k-1} \left((L-1) + \mathbb{E}\left[\Phi\left(\mathcal{C}_{i+1}\right)\right] - \mathbb{E}\left[\Phi\left(\mathcal{C}_i\right)\right]\right)
\end{aligned}$$

Il costo atteso è pari a k (costo ottimo, tutti i cluster costano 1) sommato a quanto "male" l'algoritmo ha performato rispetto alla riduzione ideale $L-1$ (quanto bisogna togliere per passare da costo L a costo 1; riducendo di $L-1$ a ogni passo vuol dire ottenere il costo ottimo alla fine).

Sia N_i il numero di cluster scoperti in $\mathcal{C}_i$. Secondo le assunzioni, $\Phi(\mathcal{C}_i) = N_i L + (k - N_i)$. Vogliamo andare a dimostrare che, se ci sono cluster scoperti, è probabile che il prossimo algoritmo scelga da lì il prossimo centro.

Per ogni A scoperto, la probabilità che nell'iterazione $i+1$ venga scelto un centro all'interno di A è

$$\mathbb{P}\left(\boldsymbol{c}_{i+1} \in A \mid \mathcal{C}_i\right) = \frac{\phi\left(\mathcal{C}_i, A\right)}{\Phi\left(\mathcal{C}_i\right)} = \frac{N_i L}{N_i L + (k - N_i)}$$

Quindi la probabilità p_{i+1} di scegliere un centro da un cluster non ancora coperto è

$$\begin{aligned} p_{i+1} = \mathbb{P}\left(\exists A \in \mathcal{C}^* : \boldsymbol{c}_{i+1} \in A \wedge A \cap \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_i\} = \emptyset \mid \mathcal{C}_i\right) &= \frac{N_i L}{N_i L + (k - N_i)} \\ &\geq \frac{(k-i)L}{(k-i)L + i} \end{aligned}$$

dove abbiamo usato $N_i \geq k - i$.

Se $\boldsymbol{c}_{i+1}$ non copre nessun A precedentemente scoperto in $\mathcal{C}_i$ (che accade con probabilità $1 - p_{i+1}$), allora $\Phi\left(\mathcal{C}_{i+1}\right) \leq \Phi\left(\mathcal{C}_i\right)$. Viceversa, se $\boldsymbol{c}_{i+1}$ copre un quale A precedentemente non coperto in $\mathcal{C}_i$ (il che accade con probabilità p_{i+1}), allora $\Phi\left(\mathcal{C}_{i+1}\right) = \Phi\left(\mathcal{C}_i\right) - L + 1 = \Phi(\mathcal{C}_i) - (L-1)$.

Di conseguenza

$$\begin{aligned} (L-1) + \mathbb{E}\left[\Phi\left(\mathcal{C}_{i+1}\right) \mid \mathcal{C}_i\right] - \mathbb{E}\left[\Phi\left(\mathcal{C}_i\right) \mid \mathcal{C}_i\right] &\leq (L-1) + 0 \cdot (1 - p_{i+1}) - (L-1)\, p_{i+1} \\ &\leq (L-1) - (L-1)\, \frac{(k-i)L}{(k-i)L + i} \\ &= (L-1)\left(\frac{i}{(k-i)L + i}\right) \\ &< L \frac{i}{(k-i)L + i} \\ &< L \frac{k}{(k-i)L} = \frac{k}{k-i} \end{aligned}$$

Quindi

$$\begin{aligned}
\mathbb{E}\left[\Phi\left(\mathcal{C}_k\right)\right] &= k + \sum_{i=0}^{k-1} \left((L-1) + \mathbb{E}\left[\Phi\left(\mathcal{C}_{i+1}\right)\right] - \mathbb{E}\left[\Phi\left(\mathcal{C}_i\right)\right]\right) \\
&= k + \sum_{i=0}^{k-1} \mathbb{E}\left[(L-1) + \mathbb{E}\left[\Phi\left(\mathcal{C}_{i+1} \mid \mathcal{C}_i\right)\right] - \mathbb{E}\left[\Phi\left(\mathcal{C}_i\right) \mid \mathcal{C}_i\right]\right] \\
&\leq k \sum_{i=0}^{k-1} \frac{k}{k-i} \\
&= k + k \sum_{i=1}^{k} \frac{1}{i} \leq k(2 + \ln k)
\end{aligned}$$

dove abbiamo usato il bound sulla somma armonica $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k} \leq 1 + \ln k$.

La dimostrazione si conclude notando che, sotto le nostre assunzioni, $\mathrm{OPT} = \Phi\left(\mathcal{C}^*\right) = k$. □

Capitolo 5

Giochi e Mercati

5.1 Hedge e Exp3 per decision-making sequenziale

Si consideri un problema di decision-making online, nel quale un algoritmo deve rispondere a una sequenza di richieste che arrivano una alla volta. L'algoritmo deve compiere un'azione nel momento in cui arriva una richiesta, ma potrebbe scoprire solo in seguito che una delle scelte precedenti è stata sub-ottimale. Comunque, una volta intraprese, le azioni passate non possono essere cambiate. Illustriamo questo scenario tramite un esempio concreto.

Si consideri il processo di scegliere un buon momento per investire in un'azione. Per semplicità, assumiamo che ci sia una singolo stock possibile e il prezzo di questo può essere modellato come una sequenza di eventi binari: *su* o *giù* (in seguito generalizzeremo in modo da ammettere eventi non binari). Ogni mattina, tentiamo di indovinare se il prezzo quel giorno salirà o scenderà: se la nostra ipotesi è sbagliata perdiamo un dollaro, se è giusta non perdiamo nulla. I movimenti dell'azione sono modellati come arbitrari o addirittura antagonistici (*adversarial*), in quanto ci potrebbe essere un ambiente con l'obiettivo di causare la perdita maggiore possibile. Per bilanciare questa assunzione pessimistica, consideriamo che durante la formulazione della nostra ipotesi possiamo avere accesso a K *"esperti"*. Questi esperti possono essere correlati in maniera arbitraria e possono fornire suggerimenti più o meno affidabili. L'obiettivo dell'algoritmo è limitare le

perdite cumulative (i.e., il numero di predizioni sbagliate) fino a raggiungere all'incirca il migliore di questi esperti.

Questo può sembrare un obiettivo impossibile, in quanto non è noto fino al termine della sequenza quale sia il migliore esperto, mentre l'algoritmo deve prendere delle decisioni strada facendo.

Un primo algoritmo naïve potrebbe essere quello di decidere la predizione di *su* o *giù* in base alla opinione di maggioranza tra gli esperti. Si può facilmente vedere come questo algoritmo possa portare a pessimi risultati nel caso in cui la maggior parte degli esperti si sbagli ripetutamente.

Un metodo migliore per arrivare a una predizione è quello di mantenere un peso per ogni esperto. Inizialmente tutti gli esperti hanno pari peso, con il progredire del tempo l'algoritmo modifica il peso di ogni esperto in proporzione al numero di predizioni corrette effettuate da quest'ultimo. Di conseguenza, la predizione dell'algoritmo viene calcolata tramite una maggioranza pesata degli esperti. In altre parole, esperti che in passato si sono dimostrati più affidabili avranno peso maggiore. Una implementazione famosa di questo algoritmo è chiamata *Hedge*. Prima di entrare nei dettagli di questo algoritmo introduciamo in maniera formale il problema di effettuare decisioni online, chiamato *prediction from expert advice*.

Tale problema si basa sul seguente protocollo per decisioni sequenziali. Un set finito di esperti $\{1, \ldots, K\}$ è fissato e noto sia a colui che prende la decisione sia allo scenario (adversarial). A ogni round $t = 1, 2, \ldots$ lo scenario sceglie segretamente una perdita $\ell_t(i) \in [0, 1]$ per esperto i; il decision-maker sceglie un esperto I_t (possibilmente a caso), subendo di conseguenza una perdita $\ell_t(I_t)$, i valori di $\ell_t(i)$ per tutti gli esperti i vengono poi rivelati. La prestazione del decision-maker al tempo T viene misurata come la differenza tra il rischio sequenziale e la perdita media del miglior esperto per $\ell_1, \ldots, \ell_T \in [0, 1]^K$, ovvero

$$\underbrace{\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\ell_t(I_t)\right]}_{\text{rischio sequenziale}} - \underbrace{\min_{i=1,\ldots,K}\left(\frac{1}{T}\sum_{t=1}^{T}\ell_t(i)\right)}_{\text{perdita media del miglior esperto}}$$

dove il valore atteso è valutato rispetto all'estrazione di $I_1, \ldots, I_T$.

Vogliamo definire un algoritmo per scegliere $I_1, \ldots, I_T$ tale che per $T \to$

∞

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\ell_t(I_T)\right] - \min_{i=1,\dots,K}\left(\frac{1}{T}\sum_{t=1}^{T}\ell_t(i)\right) \to 0$$

Ovvero, il rischio sequenziale dell'algoritmo converge verso la perdita media dell'esperto con la performance migliore, irrilevantemente della sequenza di perdite $\ell_t \in [0,1]^K$.

Come primo tentativo potremmo considerare il semplice algoritmo che sceglie l'esperto con la migliore prestazione passata

$$I_T = \arg\min_{i=1,\dots,K}\sum_{s=1}^{t-1}\ell_s(i)$$

e I_1 è scelto arbitrariamente. Questo algoritmo però avrà sempre rimorso (differenza tra performance migliore e ottenuta) lineare per qualche sequenza di perdite. Ad esempio, considerando $K = 2$ e vettori di perdita scelti come segue

- $\ell_1 = (0, 1/2)$
- per $t > 1$, $\ell_2 = (0, 1)$, $\ell_3 = (0, 1)$, $\ell_4 = (1, 0)$, …

Per semplicità, assumiamo $I_1 = 0$ in modo che $\ell_1(I_1) = 0$. Allora $I_2 = 1$, dato che $\ell_1(2) = 1/2$. Questo implica $\ell_2(I_2) = 1$. Al passo $t = 3$ si avrà $\ell_1(1) + \ell_2(1) = 1$ e $\ell_1(2) + \ell_2(2) = 1/2$, quindi $I_3 = 2$ e $\ell_3(I_t) = 1$. Come si può facilmente vedere, dopo un numero qualsiasi di step $T > 1$, l'algoritmo accumulerà perdita $T - 1$ mentre l'esperto migliore avrà perdita al più $T/2$, il che implica rimorso lineare.

Algoritmo 10: Hedge

Input: fattore di apprendimento $\gamma \in (0, 1)$

Inizializzazione: $w_1(i) = 1$ per $i = 1, \dots, K$

1 **for** $t = 1, \dots, T$ **do**

2 Definire la distribuzione $p_t(i) = w_t(i)/W_t$ dove $W_t = \sum_{j=1}^{K} w_t(j)$

3 Scegli I_t in base a p_t

4 Rivela la perdita $\ell_t(I_t)$ e i valori di $\ell_t(i)$ per ogni esperto i

5 Aggiorna i pesi secondo $w_{t+1}(i) = w_t(i)e^{-\gamma\ell_t(i)}$

Per evitare il problema, randomizziamo la scelta dell'esperto: viene scelto l'esperto i allo step t con probabilità proporzionale a $\exp\left(-\gamma \sum_s \ell_s(i)\right)$. L'algoritmo risultante (vedi Algoritmo 10) si chiama *Hedge*.

L'analisi di tale algoritmo osserva il rapporto tra il peso totale degli esperti in round consecutivi

$$\begin{aligned}
\frac{W_{t+1}}{W_t} &= \sum_{i=1}^{K} \frac{w_{t+1}(i)}{W_T} \\
&= \sum_{i=1}^{K} \frac{w_t(i) e^{-\gamma \ell_t(i)}}{W_T} && \text{(def di } w_{t+1}\text{)} \\
&= \sum_{i=1}^{K} p_t(i) e^{-\gamma \ell_t(i)} && \text{(def di } p_t\text{)} \\
&\leq \sum_{i=1}^{K} p_t(i) \left(1 - \gamma \ell_t(i) + \gamma^2 \frac{\ell_t(i)^2}{2}\right) && (*) \\
&= \sum_{i=1}^{K} p_t(i) - \gamma \sum_{i=1}^{K} p_t(i) \ell_t(i) + \frac{\gamma^2}{2} \sum_{i=1}^{K} p_t(i) \ell_t(i)^2 \\
&= 1 - \gamma \sum_{i=1}^{K} p_t(i) \ell_t(i) + \frac{\gamma^2}{2} \sum_{i=1}^{K} p_t(i) \ell_t(i)^2
\end{aligned}$$

dove la disuguaglianza $(*)$ è data da $e^{-x} \leq 1 - x + \frac{x^2}{2}$ (il che vale per ogni $x \geq 0$). Inoltre, $\sum_{i=1}^{K} p_t(i) = 1$ in quanto p_t è una distribuzione di probabilità, quindi la somma di tutte le p_t è 1.

Prendendo il logaritmo si ottiene

$$\ln\left(\frac{W_{t+1}}{W_t}\right) \leq \ln\left(1 - \gamma \sum_{i=1}^{K} p_t(i) \ell_t(i) + \frac{\gamma^2}{2} \sum_{i=1}^{K} p_t(i) \ell_t(i)^2\right)$$

Notiamo ora che

$$0 < e^{-x} \leq 1 - x + \frac{x^2}{2} \implies -x + \frac{x^2}{2} > -1$$

di conseguenza

$$\sum_{i=1}^{K} p_t(i) \left(-\gamma \ell_t(i) + \frac{\gamma^2 \ell_t(i)^2}{2}\right) \geq \sum_{i=1}^{K} p_t(i)(-1) = -1$$

Ora usiamo $\ln(1+z) \leq z$ (il che vale per ogni $z > -1$) e sommando per tutti i $1, \ldots, T$ otteniamo

$$\begin{aligned}
\ln\left(\frac{W_{T+1}}{W_1}\right) = \sum_{t=1}^{T} \ln\left(\frac{W_{t+1}}{W_t}\right) &\leq \sum_{t=1}^{T} \ln\left(1 \underbrace{-\gamma \sum_{i=1}^{K} p_t(i)\ell_t(i) + \frac{\gamma^2}{2} \sum_{i=1}^{K} p_t(i)\ell_t(i)^2}_{z}\right) \\
&\leq \sum_{t=1}^{T} \left(-\gamma \sum_{i=1}^{K} p_t(i)\ell_t(i) + \frac{\gamma^2}{2} \sum_{i=1}^{K} p_t(i)\ell_t(i)^2\right) \\
&= -\gamma \sum_{t=1}^{T} \sum_{i=1}^{K} p_t(i)\ell_t(i) + \frac{\gamma^2}{2} \sum_{t=1}^{T} \sum_{i=1}^{K} p_t(i)\ell_t(i)^2
\end{aligned}$$

Questo è un upper bound per la perdita attesa dell'algoritmo.

D'altra parte, per ogni k fissato

$$\begin{aligned}
\ln\left(\frac{W_{T+1}}{W_1}\right) \geq \ln\left(\frac{w_{T+1}(k)}{W_1}\right) &= \ln\left(\frac{\exp\left(-\gamma \sum_{t=1}^{T} \ell_T(k)\right)}{\sum_{i=1}^{K} 1}\right) \\
&= -\gamma \sum_{t=1}^{T} \ell_T(k) - \ln(K)
\end{aligned}$$

In altre parole, sicuramente la somma di tutti i pesi (W_{T+1}) sarà maggiore o uguale al peso di un singolo esperto ($w_{T+i}(k)$). Questo è un lower bound per la perdita dell'algoritmo.

Mettendo assieme i due bound trovati per $\ln\left(\frac{W_{T+1}}{W_1}\right)$:

$$-\gamma \sum_{t=1}^{T} \ell_T(k) - \ln(K) \leq \ln\left(\frac{W_{T+1}}{W_1}\right) \leq -\gamma \sum_{t=1}^{T} \sum_{i=1}^{K} p_t(i)\ell_t(i) + \frac{\gamma^2}{2} \sum_{t=1}^{T} \sum_{i=1}^{K} p_t(i)\ell_t(i)^2$$

e dividendo per $-\gamma$ si ottiene

$$\sum_{t=1}^{T} \sum_{i=1}^{K} p_t(i)\ell_t(i) - \sum_{t=1}^{T} \ell_t(k) \leq \frac{\ln(K)}{\gamma} + \frac{\gamma}{2} \sum_{t=1}^{T} \sum_{i=1}^{K} p_t(i)\ell_t(i)^2 \qquad (\dagger)$$

Si noti inoltre che

$$\mathbb{E}\left[\ell_t(I_t)\right] = \sum_{i=1}^{K} p_t(i)\ell_t(i) \qquad (\ddagger)$$

per la definizione di valore atteso. Ovvero, il valore atteso della perdita dell'algoritmo al momento t. Inoltre, usando (†), il fatto che p_t è una distribuzione e la maggiorazione $\ell_t(i)^2 \leq 1$ (le perdite sono sempre tra 0 e 1), possiamo notare

$$\sum_{i=1}^{K} p_t(i)\ell_t(i)^2 \leq \sum_{i=1}^{K} p_t(i) \cdot 1 = 1 \tag{††}$$

Usando (‡) e (††) rispettivamente a sinistra e destra di (†) si ottiene che

$$\mathbb{E}\left[\sum_{t=1}^{T} \ell_t(I_t)\right] - \sum_{t=1}^{T} \ell_t(k) \leq \frac{\ln(K)}{\gamma} + \frac{\gamma}{2}T$$

Dato che la disuguaglianza precedente vale per tutti gli esperti k e tutti i fattori di apprendimento γ, bisogna trovare come definire γ:

- Se troppo piccolo $\frac{\ln(K)}{\gamma}$ esplode
- Se troppo grande $\frac{\gamma T}{2}$ aumenta troppo

Per trovare un valore "giusto" di γ

$$\frac{\ln(K)}{\gamma} = \frac{\gamma T}{2} \implies \gamma = \sqrt{\frac{2\ln(K)}{T}}$$

Usare γ come definito sopra risulta in

$$\underbrace{\mathbb{E}\left[\sum_{t=1}^{T} \ell_t(I_t)\right] - \sum_{t=1}^{T} \ell_t(k)}_{\text{Rimorso totale}} \leq \sqrt{2T\ln(K)}$$

dividendo entrambi i lati per T si ottiene

$$\underbrace{\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} \ell_t(I_t)\right] - \min_{i=1,\ldots,K}\left(\frac{1}{T}\sum_{t=1}^{T} \ell_t(i)\right)}_{\text{Rimorso medio}} \leq \sqrt{\frac{2\ln(K)}{T}}$$

dove il $\min_{i=1,\ldots,K}$ arriva dal fatto che il rimorso totale, per come definito sopra, vale per un qualsiasi esperto k, di conseguenza vale anche per il migliore di essi (ovvero quello con la perdita minima).

Con una dimostrazione non molto più complicata, si può provare lo stesso bound con una costante leggermente peggiore nel caso in cui i pesi dell'ultimo passaggio di Hedge (Linea 5) sono definiti come

$$w_{t+1}(i) = \exp\left(-\gamma_t \sum_{s=1}^{t} \ell_s(i)\right)$$

e $\gamma_t = \sqrt{2\ln(K)/t}$. Questo prova che il rischio sequenziale dell'algoritmo converge alla perdita media del miglior esperto per $T \to \infty$.

Sorprendentemente, un risultato simile può essere ottenuto anche se l'unica perdita rivelata è quella di $\ell_t(I_t)$. Si immagini il problema di posizionare pubblicità sul Web. Per ogni utente $t = 1, 2, \ldots$ un publisher sceglie la pubblicità I_t da un insieme di K pubblicità, per poi mostrarla all'utente corrispondente. Il publisher quindi perde 1 se la pubblicità non viene premuta dall'utente, 0 altrimenti. Dopo ogni interazione il publisher scopre se l'utente ha premuto o meno la pubblicità, ma non ha modo di sapere se avrebbe premuto o meno una delle altre.

Questo problema può essere modellato dal seguente protocollo per decisioni sequenziali, chiamato *multi-arm bandit*. Un set finito di azioni $\{1, \ldots, K\}$ è fissato e noto sia al decision-maker che allo scenario (adversarial). A ogni round $t = 1, 2, \ldots$ lo scenario sceglie segretamente una perdita $\ell_t \in [0, 1]$ per ogni azione i; il decision-maker sceglie un'azione I_t (possibilmente a caso), per poi andare a perdere $\ell_t(I_t)$, e viene rivelata solo la perdita per l'azione scelta. La prestazione del decision-maker all'istante T è misurata come la differenza tra il suo rischio sequenziale (perdita media attesa) e la media delle perdite date dalla sequenza di azioni migliori $\ell_1, \ldots, \ell_T$, ovvero la differenza

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} \ell_t(I_t)\right] - \min_{i=1,\ldots,K}\left(\frac{1}{T}\sum_{t=1}^{T} \ell_t(i)\right)$$

dove il valore atteso è considerato rispetto all'estrazione di $I_1, \ldots, I_T$.

È possibile modificare l'algoritmo Hedge in modo tale che il tasso di convergenza $\mathcal{O}\left(T^{-1/2}\right)$ sia preservato. L'idea è sostituire $\ell_t(i)$ (mai rivelato) nella fase di aggiornamento dei pesi (Linea 5) con una stima. Questo viene fatto da un algoritmo chiamato Exp3 (*Exponential-weight algorithm for Exploration and Exploitation*), il quale usa stimatori basati sull'importanza

$$\hat{\ell}_t(i) = \frac{\ell_t(i)}{p_t(i)}\mathbb{I}\{I_t = i\}$$

Dove

- $\ell_t(i)$ è la perdita reale dell'azione, nota solo se $I_t = i$
- $p_t(i)$ è la probabilità con cui l'algoritmo ha deciso di scegliere l'azione i al tempo t
- $\mathbb{I}\{I_t = i\}$ è la funzione indicatrice, vale 1 se l'algoritmo ha effettivamente scelto l'azione i, 0 altrimenti

In altre parole, se l'algoritmo *non* sceglie l'azione i ($I_t \neq i$) la stima della perdita $\hat{\ell}_t(i) = 0$, invece, se l'algoritmo sceglie l'azione i, la perdita osservata $\ell_t(i)$ viene "amplificata" dividendola per la probabilità di scelta $p_t(i)$, così compensando statisticamente il fatto che l'azione non viene sempre osservata.

Dato che $\hat{\ell}_t$ è una variabile casuale, anche $p_t(i)$ lo è. Tuttavia, $p_t(i)$ è definita per ogni data realizzazione di $I_1, \ldots, I_{t-1}$. Inoltre, per costruzione, $p_t(i) = \mathbb{P}(I_t = i \mid I_1, \ldots, I_{t-1})$, quindi

$$\begin{aligned}
\mathbb{E}\left[\hat{\ell}_t(i) \mid I_1, \ldots, I_{t-1}\right] &= \mathbb{E}\left[\frac{\ell_t(i)}{p_t(i)}\mathbb{I}\{I_t = i\} \mid I_1, \ldots, I_{t-1}\right] \\
&= \frac{\ell_t(i)}{p_t(i)}\mathbb{P}(I_t = i \mid I_1, \ldots, I_{t-1}) && \text{(def di } \mathbb{E}) \\
&= \frac{\ell_t(i)}{p_t(i)}p_t(i) = \ell_t(i) && (*)
\end{aligned}$$

e

$$\begin{aligned}
\mathbb{E}\left[\hat{\ell}_t(i)^2 \mid I_1, \ldots, I_{t-1}\right] &= \mathbb{E}\left[\frac{\ell_t(i)^2}{p_t(i)^2}\mathbb{I}\{I_t = i\} \mid I_1, \ldots, I_{t-1}\right] && (\mathbb{I}\{I_t = i\}^2 = \mathbb{I}\{I_t = i\}) \\
&= \frac{\ell_t(i)^2}{p_t(i)^2}\mathbb{P}(I_t = 1 \mid I_1, \ldots, I_{t-1}) && \text{(def di } \mathbb{E}) \\
&\leq \frac{1}{p_t(i)^2}p_t(i) && (\ell_t(i)^2 \leq 1) \\
&\leq \frac{1}{p_t(i)} && (**)
\end{aligned}$$

Procedendo come per l'analisi di Hedge, (†) può essere derivata con $\hat{\ell}_t$ al posto che ℓ_t

$$\sum_{t=1}^{T}\sum_{i=1}^{K} p_t(i)\hat{\ell}_t(i) - \sum_{t=1}^{T}\hat{\ell}_t(k) \leq \frac{\ln K}{\gamma} + \frac{\gamma}{2}\sum_{t=1}^{T}\sum_{i=1}^{K} p_t(i)\hat{\ell}_t(i)^2$$

Considerando il valore atteso da entrambi i lati si ottiene

$$\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i=1}^{K} p_t(i)\hat{\ell}_t(i)\right] - \mathbb{E}\left[\sum_{t=1}^{T}\hat{\ell}_t(k)\right] \leq \frac{\ln K}{\gamma} + \frac{\gamma}{2}\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i=1}^{K} p_t(i)\hat{\ell}_t(i)^2\right]$$

Per linearità del valore atteso, la tower rule $\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X \mid Y]]$ (vale per ogni coppia di variabili casuali X e Y) e il fatto che p_t è determinata dati $I_1, \ldots, I_{t-1}$, si ottiene

$$\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i=1}^{K} p_t(i)\mathbb{E}\left[\hat{\ell}_t(i) \mid I_1, \ldots, I_{t-1}\right]\right] - \mathbb{E}\left[\sum_{t=1}^{T}\mathbb{E}\left[\hat{\ell}_t(k) \mid I_1, \ldots, I_{t-1}\right]\right]$$
$$\leq \frac{\ln K}{\gamma} + \frac{\gamma}{2}\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i=1}^{K} p_t(i)\mathbb{E}\left[\hat{\ell}_t(i)^2 \mid I_1, \ldots, I_{t-1}\right]\right]$$

dove k è una qualsiasi azione.

Applicando $(*)$ e $(**)$ si ottiene

$$\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i=1}^{K} p_t(i)\ell_t(i)\right] - \sum_{t=1}^{T}\ell_t(k) \leq \frac{\ln K}{\gamma} + \frac{\gamma}{2}\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i=1}^{K} p_t(i)\frac{1}{p_t(i)}\right] \qquad (\star)$$

Similmente a Hedge, a sinistra il rischio totale atteso dell'algoritmo, a destra l'upper bound per le perdite.

Ora, notando che

$$\mathbb{E}[\ell_t(I_t) \mid I_1, \ldots, I_{t-1}] = \sum_{i=1}^{K} p_t(i)\ell_t(i)$$

usando la tower rule $\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X \mid Y]]$ possiamo scrivere

$$\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i=1}^{K} p_t(i)\ell_t(i)\right] = \mathbb{E}\left[\sum_{t=1}^{T}\mathbb{E}[\ell_t(I_t) \mid I_1, \ldots, I_{t-1}]\right] = \mathbb{E}\left[\sum_{t=1}^{T}\ell_t(I_t)\right]$$

Inoltre

$$\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i=1}^{K} p_t(i)\frac{1}{p_t(i)}\right] \leq KT$$

ovvero, al massimo K perdite per T round.

Ancora una volta, similmente a Hedge, si mette assieme, dividendo per T e scegliendo $\gamma = \sqrt{2\ln(K)/(KT)}$, ottenendo

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\ell_t(I_t)\right] - \min_{i=1,\dots,K}\left(\frac{1}{T}\sum_{t=1}^{T}\ell_t(i)\right) \leq \sqrt{\frac{2K\ln K}{T}}$$

Questo può sorprendere, fino a un fattore di $\sqrt{K}$, questo è lo stesso tasso di convergenza ottenibile nel problema di *prediction with expert advice* dove tutte le perdite sono rivelate al termine di ogni round. Il termine aggiuntivo $\sqrt{K}$ può essere visto come conseguenza del fatto che, in questo caso, in ogni round vediamo solo $1/K$-esimo del numero totale di perdite.

Similmente a Hedge, si può eseguire Exp3 con

$$w_{t+1}(i) = \exp\left(-\gamma_t \sum_{s=1}^{t}\hat{\ell}_s(i)\right)$$

e $\gamma_t = \sqrt{2\ln(K)/t}$. Questo risulta nello stesso bound, con una costante leggermente peggiore.

5.2 Aste al primo e secondo prezzo

Quando vengono modellate aste si assume che ogni offerente i abbia un valore intrinseco $v_i \in [0,1]$ per l'oggetto messo all'asta. L'offerente è disposto a comprare l'oggetto per un prezzo fino a tale valore, ma non di più.

Aste al rialzo. Note anche come "aste inglesi", si tratta di aste interattive real time in cui il venditore alza gradualmente il prezzo finché non rimane un solo acquirente. Queste corrispondono ad aste in busta chiusa al secondo prezzo, nelle quali gli offerenti fanno offerte segrete in simultanea e il più alto offerente vince pagando il prezzo della seconda offerta più alta.

Aste al ribasso. Note anche come "aste olandesi", il venditore parte da un prezzo alto per poi abbassare gradualmente il prezzo finché qualcuno non accetta. Queste corrispondono ad aste in busta chiusa al primo prezzo, nelle quali gli offerenti fanno offerte segrete in simultanea e il più alto offerente vince, pagando il prezzo della propria offerta.

Una *shading strategy* $s : [0,1] \to [0,1]$ è una mappa da valori a offerte. Assumendo che tutti gli offerenti siano razionali, vale che $s(v) \leq v$ per ogni

$v \in [0,1]$ (l'offerta non sarà mai superiore al valore). Quindi $s(0) = 0$. Assumiamo inoltre che s sia monotona: $v' > v$ implica $s(v') > s(v)$. In altre parole, se il valore aumenta, aumenta anche l'offerta.

Aste al secondo prezzo. Se ci sono n offerenti con valori $v_1, \ldots, v_n$ i quali usano shading strategies $s_1, \ldots, s_n$ la funzione di payoff per l'offerente 1 in un'asta al secondo prezzo è

$$f_1(v_1, \ldots, v_n, s_1, \ldots, s_n) = \mathbb{I}\left\{ s_1(v_1) > \max_{i \neq 1} s_i(v_i) \right\} \cdot \left(v_1 - \max_{i \neq 1} s_i(v_i) \right)$$

similmente si possono definire quelle per gli altri offerenti. In altre parole, la funzione indicatrice indica quando l'offerente considerato è quello che ha fatto l'offerta più alta, rendendo il payoff 0 altrimenti, mentre il valore stesso del payoff è dato dalla differenza tra il valore dell'oggetto per l'offerente considerato e il valore dell'offerta del secondo miglior offerente.

Diciamo che una strategia s_1 domina per l'offerente 1 se

$$f_1(v_1, \ldots, v_n, s_1, \ldots, s_n) \geq f_1\left(v_1, \ldots, v_n, s', \ldots, s_n\right), \quad \forall v_1, \ldots, v_n, s_1, \ldots, s_n, s'$$

ovvero, se porta a payoff maggiore.

Teorema 5.2.1. *In un'asta al secondo prezzo, la strategia $s : v \mapsto v$ domina per qualsiasi offerente.*

Dimostrazione. Si consideri i con valore v_i e offerta b_i. Consideriamo prima $b_i > v_i$. Se i è vincente con $b_i = v_i$, allora aumentare l'offerta non aumenta il payoff. Se i perde con $b_i = v_i$, allora il payoff rimane zero a meno che la nuova offerta non superi la più alta $\max_{j \neq i} b_j > v_i$. In questo caso il payoff diventa negativo, di conseguenza l'offerente i non dovrebbe considerare $b_i > v_i$.

Ora consideriamo $b_i < v_i$. Se i è perdente con $b_i = v_i$, decrementare l'offerta non cambia il payoff. Se i è vincente con $b_i = v_i$, allora il payoff rimane $v_i - \max_{j \neq i} b_j > 0$ a meno che la nuova offerta non vada al di sotto della seconda più alta $\max_{j \neq i} b_j$, in tale caso il payoff diventa zero. Quindi i non dovrebbe essere $b_i < v_i$. □

In altre parole:

- Non bisogna considerare il caso $b_i > v_i$ in quanto irrazionale

- Nel caso di $b_i < v_i$, se i perde con $b_i = v_i$ al decrescere dell'offerta il payoff non cambia, mentre se i vince con $b_i = v_i$ al decrescere dell'offerta il payoff rimane lo stesso o diventa 0 nel caso in cui l'offerta non sia più la più alta

Aste al primo prezzo. Se ci sono due offerenti con valori v_1, v_2 i quali usano shading strategies s_1, s_2, la funzione di payoff per l'offerente 1 in un'asta al primo prezzo è

$$g_1(v_1, v_2, s_1, s_2) = \mathbb{I}\{s_1(v_1) > s_2(v_2)\}(v_1 - s_1(v_1))$$

similmente si può definire per il secondo offerente.

Assumendo che i valori v_1, v_2 siano derivati da due variabili casuali V_1, V_2, un equilibrio per i due offerenti è una coppia di strategie tale che

$$\begin{aligned} \mathbb{E}[g_1(v_1, V_1, s_1, s_2) - g_1(v_1, V_2, s', s_2)] \geq 0 \\ \mathbb{E}[g_2(V_1, v_2, s_1, s_2) - g_2(V_1, v_2, s_1, s')] \geq 0 \end{aligned} \qquad \forall s', v_1, v_2$$

Ovvero, il guadagno atteso (rispetto alle variabili V_1 e V_2, in quanto il reale valore per l'avversario non è noto) per ognuno dei giocatori usando la strategia corrente (s_1 o s_2) è maggiore del guadagno atteso dato da qualsiasi altra strategia (s'). Nessun giocatore ha incentivo a cambiare la propria offerta.

Teorema 5.2.2. *Se i valori V_1, V_2 per i due offerenti sono estratti indipendentemente da una distribuzione uniforme sull'intervallo $[0,1]$ e i due offerenti usano la stessa shading strategy s, allora (s, s) con $s : v \mapsto v/2$ è un equilibrio per gli offerenti in un'asta al primo prezzo.*

Dimostrazione. Dato il valore v_1, il payoff atteso per il l'offerente 1 è

$$\begin{aligned} \mathbb{E}[g_1(v_1, V_2, s, s)] &= \mathbb{P}(s(v_1) > s(v_2))(v_1 - s(v_1)) && \text{(def di } \mathbb{E} \text{ e } g_1) \\ &= \mathbb{P}(v_1 > V_2)(v_1 - s(v_1)) && \text{(monotonia)} \\ &= v_1(v_1 - s(v_2)) && \text{(distribuzione uniforme)} \end{aligned}$$

dove abbiamo usato le assunzioni di monotonia e distribuzione uniforme per V_1, V_2. La condizione di equilibrio per l'offerente 1 quindi dice che

$$v_1(v_1 - s(v_1)) \geq v_1\left(v_1 - s'(v_1)\right)$$

Dato che l'offerente 2 non offrirà mai più di $s(1)$, possiamo assumere che s' soddisfi la condizione $s'(v) \in [0, s(1)]$ per ogni $v \in [0,1]$. Infatti, offrire

più di $s(1)$ può solo ridurre il payoff dell'offerente 1, senza aumentare le probabilità di vittoria. Questo implica che possiamo trovare $v \in [0,1]$ tale che $s'(v_1) = s(v)$. Quindi la condizione di equilibrio diventa

$$v_1 \left(v_1 - s(v_1)\right) \geq v_1 \left(v_1 - s(v)\right)$$

Sostituendo $s(v) = v/2$ si ottiene che

$$\frac{v_1^2}{2} \geq v_1^2 - \frac{vv_1}{2}$$

Moltiplicando per 2 da entrambi i lati si ottiene $v_1^2 + v^2 - 2vv_1 \geq 0$, che è sempre vera. □

5.3 Ottimizzazione del prezzo di riserva in aste al secondo prezzo

Nell'ambito di pubblicità online, i publisher vendono il loro spazio agli advertiser attraverso aste al secondo prezzo tramite ad exchanges. Per ogni visita creata sul sito del publisher, l'ad exchange crea un'asta on-the-fly. Dati empirici mostrano come una scelta informata per il prezzo di riserva, squalificando ogni offerta al di sotto di tale prezzo, può avere impatto significativo sul profitto del venditore. Assumiamo che il venditore sia osservando l'offerta più alta e il profitto.

Il guadagno del venditore in un'asta al secondo prezzo è calcolato come segue: se il prezzo di riserva r non è maggiore della seconda maggiore offerta $b(2)$, allora l'oggetto viene venduto al miglior offerente e il guadagno del venditore è pari a $b(2)$. Se r è tra $b(2)$ e $b(1)$, l'oggetto viene venduto al miglior offerente e il guadagno coincide con il prezzo di riserva. Infine, se r è maggiore di $b(1)$, allora l'oggetto non viene venduto e il guadagno del venditore è zero. Formalmente, il guadagno del venditore è

$$g\left(r, b(1), b(2)\right) = \max\left\{r, b(2)\right\} \mathbb{I}\left\{r \leq b(1)\right\}$$

Si noti come il guadagno dipenda solamente da prezzo di riserva r e le due migliori offerte $b(1) \geq b(2)$. Assumiamo che tutte le quantità siano nell'intervallo $[0,1]$.

All'inizio di ogni asta $t = 1, 2, \ldots$, il venditore calcola un nuovo prezzo di riserva $r_t \in [0,1]$. In seguito vengono raccolte le offerte $b_t(1), b_t(2), \ldots$

e il venditore guarda al guadagno $g_t(r_t) = g\left(r_t, b_t(1), b_t(2)\right)$, assieme alla migliore offerta $b_t(1)$. Sapere $g_t(r_t)$ e $b_t(1)$ permette di calcolare $g_t(r)$ per ogni $r \geq r_t$. Per motivi tecnici, usiamo la perdita $\ell_t(r_t) = 1 - g_t(r_t)$ al posto del guadagno.

La funzione perdita $\ell_t : [0,1] \to [0,1]$ soddisfa la condizione semi-Lipschitz

$$\ell_t(y+\delta) \geq \ell_t(y) - \delta, \quad \forall 0 \leq y \leq y+\delta \leq 1 \tag{$\dagger$}$$

ovvero, per ogni valore di $y \in [0, 1-\delta]$ e δ positivo, la perdita $\ell_t(y+\delta)$ non può essere minore della perdita $\ell_t(y) - \delta$. Intuitivamente, aggiungere δ al prezzo di riserva porta a una perdita di almeno δ in meno.

Il rimorso è definito come

$$R_T = \mathbb{E}\left[\sum_{t=1}^{T} \ell_t(r_t)\right] - \inf_{0 \leq y \leq 1} \sum_{t=1}^{T} \ell_t(y)$$

dove il valore atteso è rispetto alla casualità nel valore di r_t. Il primo termine rappresenta il valore atteso per le perdite subite in tutte le T aste, il secondo invece rappresenta la perdita totale nel caso del migliore (dato da inf) prezzo di riserva costante; di conseguenza, il rimorso rappresenta la differenza tra come ha performato l'algoritmo e come avrebbe performato la migliore strategia costante possibile.

Introduciamo l'algoritmo Exp3-RTB (Real-Time Bidding, Algoritmo 11), una variante di Exp3 che sfrutta il migliore feedback $\{\ell_t(y) : y \geq r_t\}$. L'algoritmo usa una discretizzazione dello spazio d'asta $[0,1]$ in $K = \lceil 1/\gamma \rceil$ aste $y_k := (k-1)\gamma$ per $k = 1, \ldots, K$.

Teorema 5.3.1. *L'algoritmo Exp3-RTB con $0 < \gamma \leq 1$ soddisfa*

$$R_T \leq \gamma T \left(2 + \frac{1}{4} \ln \frac{e}{\gamma}\right) + \frac{2 \ln \lceil 1/\gamma \rceil}{\gamma}$$

In particolare, $\gamma = T^{-1/2}$ risulta in $R_T = \mathcal{O}\left((\ln T)\sqrt{T}\right)$.

Dimostrazione. La dimostrazione segue la stessa idea dell'analisi del rimorso di Exp3. La differenza fondamentale è un controllo più stretto del termine di varianza, permesso dal miglior feedback.

Si scelga un qualsiasi prezzo di riserva $y_k = (k-1)\gamma$. Controlliamo prima il rimorso associato alle azioni estratte da p_t (il rimorso associato con q_t lo si

Algoritmo 11: Exp3-RTB

Input: Parametro di esplorazione $0 < \gamma \leq 1$

Definisci il parametro $\eta = \gamma/2$ e la distribuzione uniforme p_1 su $\{1, \ldots, K\}$ dove $K = \lceil 1/\gamma \rceil$

for $t = 1, 2, \ldots$ **do**

  Calcola la distribuzione

$$q_t(k) = (1 - \gamma)p_t(k) + \gamma \mathbb{I}\{k = 1\}$$

  per $k = 1, \ldots, K$

  Estrai $I_t \sim q_t$ e imposta $r_t = (I_t - 1)\gamma$

  for each $k = 1, \ldots, K$ **do**

    Calcola la perdita stimata

$$\hat{\ell}_t(k) = \frac{\ell_t(y_k)}{\sum_{j=1}^{k} q_t(j)} \mathbb{I}\{I_t \leq k\}$$

  for each $k = 1, \ldots, K$ **do**

    Calcola il nuovo assegnamento di probabilità

$$p_{t+1}(k) = \frac{\exp\left(-\eta \sum_{s=1}^{t} \hat{\ell}_s(k)\right)}{\sum_{j=1}^{K} \exp\left(-\eta \sum_{s=1}^{t} \hat{\ell}_s(j)\right)}$$

può studiare come diretta conseguenza). Più precisamente, dato che le perdite stimate $\hat{\ell}_t(j)$ sono non negative, possiamo applicare l'analisi standard di Exp3 per ottenere

$$\sum_{t=1}^{T}\sum_{i=1}^{K} p_t(i)\hat{\ell}_t(i) - \sum_{t=1}^{T}\hat{\ell}_t(k) \leq \frac{\eta}{2}\sum_{t=1}^{T}\sum_{j=1}^{K} p_t(j)\hat{\ell}_t(j)^2 + \frac{\ln(K)}{\eta} \qquad (\ddagger)$$

ovvero, lower e upper bound per il rimorso dell'algoritmo.

Scrivendo $\mathbb{E}_{t-1}[\cdot]$ per il valore atteso condizionato su $I_1, \ldots, I_{t-1}$, si noti che

$$\mathbb{E}_{t-1}\left[\hat{\ell}_t(j)\right] = \ell_t(y_j)$$
$$\mathbb{E}_{t-1}\left[p_t(j)\hat{\ell}_t(j)^2\right] = \frac{p_t(j)\ell_t(y_j)^2}{\sum_{i=1}^{j} q_t(i)} \leq \frac{q_t(j)}{(1-\gamma)\sum_{i=1}^{j} q_t(i)}$$

dove abbiamo usato la definizione di q_t e il fatto che $\ell_t(y_j) \leq 1$ per assunzione.

Quindi, prendendo il valore atteso da entrambi i lati di $(\ddagger)$ implica, similmente a come fatto nell'analisi di Exp3,

$$\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i=1}^{K} p_t(i)\ell_t(y_i)\right] - \sum_{t=1}^{T}\ell_t(y_k) \leq \frac{\eta}{2\,(1-\gamma)}\sum_{t=1}^{T}\mathbb{E}\left[\sum_{j=1}^{K}\frac{q_t(j)}{\sum_{i=1}^{j} q_t(i)}\right] + \frac{\ln K}{\eta}$$

Definendo $s_t(j) = \sum_{i=1}^{j} q_t(i)$ possiamo limitare superiormente la somma con un integrale

$$\begin{aligned}
\sum_{j=1}^{K}\frac{q_t(j)}{\sum_{i=1}^{j} q_t(i)} &= 1 + \sum_{j=2}^{K}\frac{s_t(j) - s_t(j-1)}{s_t(j)} \\
&= 1 + \sum_{j=2}^{K}\int_{s_t(j-1)}^{s_t(j)}\frac{1}{s_t(j)}dx && \left(\int_A^B C\cdot dx = (B-A)C\right) \\
&\leq 1 + \sum_{j=2}^{K}\int_{s_t(j-1)}^{s_t(j)}\frac{1}{x}dx && \left(x \leq s_t(j) \Leftrightarrow \frac{1}{x} \geq \frac{1}{s_t(j)}\right) \\
&= 1 + \int_{q_t(1)}^{1}\frac{1}{x}dx && \text{(unione degli integrali)} \\
&\leq 1 - \ln\left(q_t(1)\right) && (\ln(1) = 0) \\
&\leq 1 + \ln\left(\frac{1}{\gamma}\right) && \left(q_t(1) \geq \gamma \Leftrightarrow \frac{1}{q_t(1)} \leq \frac{1}{\gamma}\right)
\end{aligned}$$

dove abbiamo usato $q_t(1) \geq \gamma$ (per definizione di q_t). Quindi, sostituendo nel bound precedente, otteniamo

$$\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i=1}^{K} p_t(i)\ell_t(y_i)\right] - \sum_{t=1}^{T}\ell_t(y_k) \leq \frac{\eta T \ln(e/\gamma)}{2(1-\gamma)} + \frac{\ln K}{\eta} \qquad (*)$$

Questo è il bound del rimorso rispetto alla distribuzione p_t, ma l'algoritmo usa q_t, quindi controlliamo il rimorso delle riserve $r_t = (I_t - 1)\gamma$, dove I_t è estratto da $q_t = (1-\gamma)p_t + \gamma\delta_1$. La distribuzione q_t è definita come

$$q_t(k) = (1-\gamma)\, p_t(k) + \gamma$$

ovvero, con probabilità $(1-\gamma)$ viene usata la distribuzione p_t, mentre con probabilità γ usa il prezzo più basso (favorendo "l'esplorazione", scegliere il prezzo più basso permette di calcolare quale sarebbe stato il guadagno/perdita per qualsiasi valore di $r > r_t$).

Abbiamo

$$\begin{aligned}
\mathbb{E}\left[\sum_{t=1}^{T}\ell_t(r_t)\right] - \sum_{t=1}^{T}\ell_t(y_k) &= \mathbb{E}\left[\sum_{t=1}^{T}\left((1-\gamma)\sum_{i=1}^{K} p_t(i)\ell_t(y_i) + \gamma\ell_t(y_1)\right)\right] - \sum_{t=1}^{T}\ell_t(y_k) \\
&\overset{(1)}{\leq} (1-\gamma)\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i=1}^{K} p_t(i)\ell_t(y_i)\right] + \gamma T - \sum_{t=1}^{T}\ell_t(y_k) \\
&\overset{(*)}{\leq} \frac{\eta T \ln(e/\gamma)}{2} + \frac{\ln K}{\eta} + \gamma T \qquad (**)
\end{aligned}$$

In seguito alla prima uguaglianza

- $(1-\gamma)\sum_{i=1}^{K} p_t(i)\ell_t(y_i)$ indica la perdita media dovuta alla strategia p_t
- $\gamma\ell_t(y_1)$ indica la perdita dovuta all'esplorazione forzata del prezzo più basso
- $\sum_{t=1}^{T}\ell_t(y_k)$ indica la perdita totale del miglior prezzo fisso

Il passaggio (1) è dato dal fatto che

$$\gamma\sum_{t=1}^{T}\ell_t(y_1) \leq \gamma T$$

in quanto la perdita per ognuno dei T turni di esplorazione è al massimo 1. L'ultima disuguaglianza vale per $(*)$.

Per concludere la dimostrazione, limitiamo superiormente il rimorso contro un qualsiasi valore fissato di $y \in [0,1]$. Dato che esiste $k \in \{1,\dots,K\}$ tale che $y \in [y_k, y_k+\gamma]$ (ovvero, qualsiasi valore di y deve essere "vicino" a uno dei valori discretizzati) e dato che ogni ℓ_t soddisfa la condizione semi-Lipschitz (†), abbiamo che $\ell_t(y) \geq \ell_t(y_k) - \gamma$, quindi $\ell_t(y_k) \leq \ell_t(y) + \gamma$. Questo risulta in

$$\min_{k=1,\dots,K} \mathbb{E}\left[\sum_{t=1}^{T} \ell_t(y_k)\right] \leq \min_{0\leq y\leq 1} \sum_{t=1}^{T} \ell_t(y) + \gamma T$$

Questo significa che "accontentarsi" della discretizzazione costa al massimo γT in rimorso addizionale.

Mettendo l'ultima disuguaglianza in $(**)$ e ricordando che $K = \lceil 1/\gamma \rceil$ e $\eta = \gamma/2$, finalmente otteniamo

$$R_T \leq \frac{\gamma T}{4} \ln \frac{e}{\gamma} + \frac{2\ln\lceil 1/\gamma \rceil}{\gamma} + 2\gamma T$$

Scegliere $y \approx T^{-1/2}$ termina la dimostrazione. □

Si noti che, se al posto di Exp3-RTB avessimo usato Exp3 con $\eta > 0$ nella griglia di $K = \lceil 1/\gamma \rceil$ prezzi, avremmo ottenuto un bound nella forma

$$R_T \leq \frac{\ln K}{\eta} + \frac{\eta}{2} KT + \gamma T = \frac{\ln\lceil 1/\gamma \rceil}{\eta} + \frac{\eta T}{2\gamma} + \gamma T$$

che, per $\gamma = T^{-1/3}$ e $\eta = T^{-2/3}$ fornisce $R_t = \mathcal{O}\left(T^{2/3}\right)$ ignorando fattori logaritmici, un bound molto peggiore di quello ottenuto tramite Exp3-RTB usando il miglior feedback dato dalla struttura di questo problema.

5.4 Il teorema minimax di Von Neumann

Consideriamo un gioco come la morra cinese (carta, forbice, sasso) oppure pari-e-dispari, dove il giocatore e il suo avversario rivelano simultaneamente le proprie mosse. Questi giochi sono detti a somma zero, in quanto ciò che vince il giocatore lo perde l'avversario e viceversa; possiamo rappresentarli con una matrice reale G di dimensioni $m \times n$, dove le m righe rappresentano le mosse del giocatore e le n colonne rappresentano le mosse dell'avversario. Se il giocatore sceglie la mossa i e l'avversario sceglie la mossa j, allora il giocatore guadagna $G_{i,j}$ e l'avversario perde $-G_{i,j}$. Se $G_{i,j} > 0$, allora

l'avversario paga $G_{i,j}$ al giocatore; se invece $G_{i,j} < 0$ è il giocatore a pagare $G_{i,j}$ all'avversario.

Per esempio, la matrice G di morra cinese è indicata qui sotto a sinistra

	carta	forbice	sasso
carta	0	-1	$+1$
forbice	$+1$	0	-1
sasso	-1	$+1$	0

	alto	basso	centro
sinistra	$+3$	-1	$+2$
destra	-1	$+2$	-2

Ragioniamo cosa può fare il giocatore nel gioco presentato nella tabella di destra. La mossa "sinistra" fornisce un guadagno massimo di $+3$ e una perdita massima di -1. la mossa "destra" fornisce un guadagno massimo di $+2$ e una perdita massima di -2. Quindi il giocatore preferirà giocare sinistra. L'avversario, usando un argomento del tutto simile, preferisce giocare "centro". Se entrambi i giocatori seguono il proprio ragionamento, abbiamo che il giocatore vince due punti all'avversario. D'altra parte, l'avversario può immaginare che il giocatore faccia il ragionamento che lo porti a giocare "sinistra" e di conseguenza potrebbe scegliere "basso" invece che "centro", in modo da prendere un punto al giocatore. A sua volta, però il giocatore può prevedere la contromossa dell'avversario e prevenirla, e così via. Per spezzare questo circolo e neutralizzare il ragionamento dell'avversario il giocatore può usare la randomizzazione.

Supponiamo che il giocatore riveli all'avversario la distribuzione $\boldsymbol{p}$ da cui estrarrà la propria mossa $I \in \{1, \ldots, m\}$ (vettore di probabilità sulle mosse). L'avversario sceglie quindi la propria mossa $J \in \{1, \ldots, n\}$, dopodiché la mossa del giocatore viene estratta da $\boldsymbol{p}$ e il gioco termina. In questo modo il giocatore delega la scelta effettiva della sua mossa alla randomizzazione usata per estrarre I (non implica necessariamente che ogni mossa venga scelta con la stessa probabilità). Il circolo vizioso è così spezzato in quanto l'avversario conosce $\boldsymbol{p}$ e può quindi calcolare la propria mossa migliore *a meno della randomizzazione usata per estrarre I* sulla quale il giocatore non ha però più alcun controllo. Rivelando la propria strategia, sembra però ovvio che il giocatore abbia avvantaggiato l'avversario. Il teorema minimax mostra come, sorprendentemente, questo non sia vero.

L'avversario può utilizzare la conoscenza di $\boldsymbol{p}$ per giocare la mossa J che

minimizza il valore atteso del guadagno del giocatore

$$J = \arg \min_{j=1,\dots,n} \sum_{i=1}^{m} G_{i,j} p_i \tag{†}$$

dove $p_i = \mathbb{P}(I = i)$.

A questo punto, il giocatore sceglierà la distribuzione $\boldsymbol{p}^*$ che massimizza il proprio guadagno atteso

$$\boldsymbol{p}^* = \arg \max_{\boldsymbol{p}} \left(\min_{j=1,\dots,n} \sum_{i=1}^{m} G_{i,j} p_i \right)$$

Si noti che l'avversario che conosce $\boldsymbol{p}$ non ha alcun vantaggio a estrarre la propria mossa J da una distribuzione. Ovvero

$$\min_{j=1,\dots,n} \sum_{i=1}^{m} G_{i,j} p_i = \min_{\boldsymbol{q}} \sum_{j=1}^{n} \left(\sum_{i=1}^{m} G_{i,j} p_i \right) q_j = \min_{\boldsymbol{q}} \boldsymbol{p}^\top G \boldsymbol{q} \tag{‡}$$

dove l'ultimo termine utilizza la notazione matriciale per denotare la doppia somma che lo precede. Il payoff per l'avversario in questo caso può essere visto come una media pesata del payoff sulle singole colonne e la media pesata di un insieme di valori non può mai essere inferiore al valore minimo dell'insieme; in altre parole, se c'è una strategia pura (mossa deterministica) dell'avversario che causa maggiore perdita al giocatore, è inutile "diluirla" con altre mosse; ogni strategia randomizzata ha un "caso peggiore" deterministico.

Sfruttando questa identità, possiamo allora scrivere il guadagno atteso del giocatore come

$$\max_{\boldsymbol{p}} \left(\min_{\boldsymbol{q}} \boldsymbol{p}^\top G \boldsymbol{q} \right)$$

Ovvero, la distribuzione di probabilità $\boldsymbol{p}$ che massimizza il guadagno nel caso peggiore.

Simmetricamente, possiamo pensare che sia l'avversario a rivelare $\boldsymbol{q}$ al giocatore, che quindi può calcolare la propria mossa migliore I per massimizzare il valore atteso della propria vincita

$$I = \arg \max_{i=1,\dots,n} \sum_{j=1} G_{i,j} q_j$$

La strategia migliore dell'avversario porta allora a un guadagno atteso per il giocatore pari a

$$\min_{\boldsymbol{q}} \left(\max_{i=1,\ldots,m} \sum_{j=1}^{n} G_{i,j} q_j \right) = \min_{\boldsymbol{q}} \left(\max_{\boldsymbol{p}} \boldsymbol{p}^\top G \boldsymbol{q} \right) \qquad (*)$$

Il teorema minimax dice che il guadagno atteso del giocatore non cambia a seconda di chi sia il primo a rivelare la propria strategia randomizzata.

Teorema 5.4.1 (Minimax)**.** *In qualsiasi gioco G a somma zero vale*

$$\max_{\boldsymbol{p}} \left(\min_{\boldsymbol{q}} \boldsymbol{p}^\top G \boldsymbol{q} \right) = \min_{\boldsymbol{q}} \left(\max_{\boldsymbol{p}} \boldsymbol{p}^\top G \boldsymbol{q} \right)$$

Un altro modo di interpretare il teorema minimax è il seguente. Sia V_G il valore comune delle formule al membro sinistro e destro nell'enunciato del teorema. Allora il giocatore ha una strategia

$$\boldsymbol{p}^* = \arg\max_{\boldsymbol{p}} \left(\min_{\boldsymbol{q}} \boldsymbol{p}^\top G \boldsymbol{q} \right)$$

che gli garantisce un guadagno atteso di almeno V_G qualunque sia la strategia $\boldsymbol{q}$ dell'avversario. Viceversa, l'avversario ha una strategia

$$\boldsymbol{q}^* = \arg\min_{\boldsymbol{q}} \left(\max_{\boldsymbol{p}} \boldsymbol{p}^\top G \boldsymbol{q} \right)$$

che gli garantisce un guadagno atteso del giocatore pari ad al più V_G qualunque sia la strategia $\boldsymbol{p}$ del giocatore.

Nel seguito, assumiamo senza perdita di generalità che gli elementi di G siano riscalati nell'intervallo $[-1, 1]$. Per comodità di notazione, nel seguito usiamo $p(i)$ e $q(i)$ per denotare, rispettivamente, le componenti di $\boldsymbol{p}$ e $\boldsymbol{q}$.

Lemma 5.4.1. *Per qualsiasi intero positivo T e per qualsiasi sequenza $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_T$ di strategia dell'avversario, l'algoritmo Hedge con parametro $\eta > 0$ garantisce*

$$\sum_{t=1}^{T} \boldsymbol{p}_t^\top G \boldsymbol{q}_t \geq \max_{\boldsymbol{p}} \sum_{t=1}^{T} \boldsymbol{p}^\top G \boldsymbol{q}_t - \frac{2 \ln m}{\eta} - \eta T$$

Questo fornisce un bound (parte destra) al guadagno totale del giocatore (parte sinistra) il quale usa Hedge contro la sequenza di mosse dell'avversario, rispetto alla migliore strategia fissa.

Dimostrazione. Per applicare Hedge, riscaliamo gli elementi di G nell'intervallo $[0, 1]$. Definiamo quindi il vettore delle perdite delle azioni al tempo t come $\boldsymbol{\ell}_t = (\mathbf{1} - G\boldsymbol{q}_t)/2 \in [0, 1]$, dove $\mathbf{1} = (1, \ldots, 1)$. Vogliamo trasformare il problema da massimizzazione del guadagno a minimizzazione delle perdite. L'analisi di Hedge garantisce che le distribuzioni $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_t$ calcolate dall'algoritmo soddisfano (come visto nella Sezione 5.1)

$$\sum_{t=1}^{T} \boldsymbol{\ell}_t^\top \boldsymbol{p}_t \leq \min_{i=1,\ldots,m} \sum_{t=1}^{T} \ell_t(i) + \frac{\ln m}{\eta} + \frac{\eta}{2} T \qquad (**)$$

Dato che nel simplesso delle probabilità $\left\{\boldsymbol{p} \geq 0 : \boldsymbol{p}^\top \mathbf{1} = 1\right\}$ (insieme di tutte le distribuzioni di probabilità) la funzione lineare

$$F(\boldsymbol{p}) = \sum_{t=1}^{T} \boldsymbol{\ell}_t^\top \boldsymbol{p}$$

è minimizzata in un vertice del simplesso (mosse pure i), abbiamo che

$$\min_{i=1,\ldots,m} \sum_{t=1}^{T} \ell_t(i) = \min_{\boldsymbol{p}} \sum_{t=1}^{T} \boldsymbol{\ell}_t^\top \boldsymbol{p}$$

Ricordando che $\boldsymbol{\ell}_t = (\mathbf{1} - G\boldsymbol{q}_t)/2$ abbiamo che, per un $\boldsymbol{p}$ arbitrario

$$\boldsymbol{p}^\top \boldsymbol{\ell}_t = \boldsymbol{p}^\top \frac{\mathbf{1} - G\boldsymbol{q}_t}{2} = \frac{\mathbf{1} - \boldsymbol{p}^\top G \boldsymbol{q}_t}{2}$$

, Infine, moltiplicando per 2 entrambi i membri di $(**)$ per cancellare il fattore $1/2$, otteniamo la tesi. □

Siamo pronti ora per la dimostrazione del teorema minimax.

Dimostrazione di minimax (5.4.1). Sia

$$\boldsymbol{q}^* = \arg\min_{\boldsymbol{q}} \max_{\boldsymbol{p}} \boldsymbol{p}^\top G \boldsymbol{q}$$

ovvero, la migliore strategia per l'avversario.

Allora

$$\max_{\boldsymbol{p}} \min_{\boldsymbol{q}} \boldsymbol{p}^T G \boldsymbol{q} \leq \max_{\boldsymbol{p}} \boldsymbol{p}^\top G \boldsymbol{q}^* = \min_{\boldsymbol{q}} \max_{\boldsymbol{p}} \boldsymbol{p}^\top G \boldsymbol{q}$$

ovvero, se il giocatore sceglie la sua strategia ottima $\boldsymbol{p}$ contro $\boldsymbol{q}^*$ il risultato non può superare il valore del gioco per definizione. In altre parole, se il giocatore deve dichiarare per primo è svantaggiato o alla pari rispetto al caso in cui è l'avversario a dover dichiarare per primo.

Per dimostrare l'altra direzione, supponiamo che $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_T$ siano generati da Hedge rispetto alle strategie $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_T$ dell'avversario definite come

$$\boldsymbol{q}_t = \arg\min_{\boldsymbol{q}} \boldsymbol{p}_t^\top G \boldsymbol{q}$$

ovvero, a ogni turno l'avversario gioca la risposta migliore alla strategia corrente del giocatore, quello che minimizza il guadagno del giocatore.

Definiamo anche

$$\bar{\boldsymbol{p}} = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{p}_t \quad \text{e} \quad \bar{\boldsymbol{q}} = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{q}_t$$

ovvero, rispettivamente, strategia media di giocatore e avversario.

Possiamo quindi scrivere

$$\begin{aligned}
\max_{\boldsymbol{p}} \min_{\boldsymbol{q}} \boldsymbol{p}^\top G \boldsymbol{q} &\geq \min_{\boldsymbol{q}} \bar{\boldsymbol{p}}^\top G \boldsymbol{q} \\
&= \min_{\boldsymbol{q}} \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{p}_t^\top G \boldsymbol{q} && \text{(def di } \bar{\boldsymbol{p}}\text{)} \\
&\geq \frac{1}{T} \sum_{t=1}^{T} \min_{\boldsymbol{q}} \boldsymbol{p}_t^\top G \boldsymbol{q} \\
&= \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{p}_t^\top G \boldsymbol{q}_t && \text{(def di } q_t\text{)} \\
&\geq \max_{\boldsymbol{p}} \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{p}^\top G \boldsymbol{q}_t - \frac{2 \ln m}{\eta T} - \eta && \text{(Lemma 5.4.1)} \\
&= \max_{\boldsymbol{p}} \boldsymbol{p}^\top G \bar{\boldsymbol{q}} - \frac{2 \ln m}{\eta T} - \eta && \text{(def di } \bar{\boldsymbol{q}}\text{)} \\
&\geq \min_{\boldsymbol{q}} \max_{\boldsymbol{p}} \boldsymbol{p}^\top G \boldsymbol{q} - \frac{2 \ln m}{\eta T} - \eta && \text{(prop. di } \bar{\boldsymbol{q}}\text{)}
\end{aligned}$$

dove abbiamo applicato il Lemma 5.4.1. Scegliendo

$$\eta = \sqrt{\frac{2 \ln m}{T}}$$

otteniamo

$$\max_{\boldsymbol{p}} \min_{\boldsymbol{q}} \boldsymbol{p}^\top G \boldsymbol{q} \geq \min_{\boldsymbol{q}} \max_{\boldsymbol{p}} \boldsymbol{p}^\top G \boldsymbol{q} - \sqrt{\frac{8 \ln m}{T}}$$

Dato che la disuguaglianza vale per qualsiasi T, per $T \to \infty$ abbiamo che

$$\max_{\boldsymbol{p}} \min_{\boldsymbol{q}} \boldsymbol{p}^\top G \boldsymbol{q} \geq \min_{\boldsymbol{q}} \max_{\boldsymbol{p}} \boldsymbol{p}^\top G \boldsymbol{q}$$

che conclude la dimostrazione. □

5.5 Boosting

Imparare una funzione binaria

Istanza. Sia $\mathcal{X} = \{x_1, \ldots, x_n\}$ lo spazio delle istanze di istanze, contenente n punti (valore finito), e sia $\mathcal{Y}\{-1, +1\}$ lo spazio delle label, contenente una label negativa (-1) e una positiva $(+1)$. Si assuma esista una funzione binaria $f : \mathcal{X} \to \mathcal{Y}$ con il compito di associare ogni punto alla relativa label, inoltre esiste un insieme finito $\mathcal{H} = \{h_1, \ldots, h_m\}$ di m funzioni binarie $h_i : \mathcal{X} \to \mathcal{Y}$; la classe $\mathcal{H}$ è convenzionalmente nota come *hypothesis class* e f come *ground truth*. Infine, sia $\boldsymbol{q}$ qualsiasi distribuzione su $\mathcal{X}$, fissata ma sconosciuta; equivalentemente, possiamo considerare $\boldsymbol{q}$ come una distribuzione sugli indici $[n] = \{1, \ldots, n\}$ dei punti in $\mathcal{X}$.

Funzione costo. Misuriamo la performance delle predizioni date da una qualsiasi funzione $h : \mathcal{X} \to \mathcal{Y}$ rispetto alla ground truth f e alla distribuzione $\boldsymbol{q}$, usando una *funzione costo* $c : \mathcal{Y} \times \mathcal{Y} \to [0, 1]$, la quale assegna un costo $c(\hat{y}, y) \in [0, 1]$ a ogni coppia di label predetto $\hat{y} \in \mathcal{Y}$ e label effettivo $y \in \mathcal{Y}$. In particolare, assumiamo che $c(\hat{y}, y) = 0$ se e solo se $\hat{y} = y$. Di conseguenza, possiamo definire la *perdita* di h sulla distribuzione $\boldsymbol{q}$ come

$$\ell_{\boldsymbol{q}}(h) = \mathbb{E}_{j \sim \boldsymbol{q}} \left[c\left(h(x_j), f(x_j)\right) \right] = \sum_{j \in [n]} q_j \cdot c\left(h\left(x_j\right), f\left(x_j\right)\right)$$

Obiettivo. Dato accesso ad $\mathcal{H}$ e f, si vuole ottenere un'aggregazione delle funzioni ipotesi di $\mathcal{H}$, $h^\star = h^\star\left(h_1, \ldots, h_m\right)$, tale che $h^\star = f$.

Notazione	Significato
$\mathcal{X} = \{x_1, \ldots, x_n\}$	spazio delle istanza
$\mathcal{Y} = \{-1, +1\}$	spazio delle label
$f : \mathcal{X} \to \mathcal{Y}$	label effettivi (ground truth)
$\mathcal{H} = \{h_1, \ldots, h_m\}$	hypothesis class
$\mathbf{q} = (q_1, \ldots, q_n)^\top$	distribuzione su $[n]$
$c : \mathcal{Y} \times \mathcal{Y} \to [0, 1]$	funzione costo

Tabella 5.1: Riassunto della notazione.

Boosting as a game

L'idea è che ogni funzione ipotesi h_i fornisce delle informazioni riguardo la ground truth f, in base alla dimensione della perdita rispetto alla distribuzione $\boldsymbol{q}$.

Binary prediction game. Se non avessimo le informazioni fornite da $\mathcal{H}$, il meglio che possiamo sperare di fare sarebbe scegliere la migliore distribuzione sui possibili label $\mathcal{Y}$ in modo tale da minimizzare il costo atteso nel caso peggiore. Questo si può modellare come un gioco two-player zero-sum con una matrice di costo 2×2 (al posto del payoff) C, che chiameremo *binary prediction game*. Il giocatore cerca di scegliere l'ipotesi h_i che minimizza l'errore rispetto alla distribuzione $\boldsymbol{q}$ scelta dall'avversario, il quale tenta di massimizzare l'errore dell'algoritmo.

Dato il gioco C, il teorema Minimax di von Neumann (5.4.1) dice che la strategia minimax porta a una perdita pari al valore del gioco V_C quando l'avversario porta al peggior valore di verità possibile (worst-case scenario). Questo metodo per determinare la possibile ground truth f non è deterministico, nè, in genere, ottimale.

Mostreremo che è possibile risolvere questi problemi date alcune assunzioni ragionevoli su come $\mathcal{H}$ si relaziona a f. Il framework di boosting considera la *weak-learning assumption*.

Assunzione 5.5.1 (Weak learning)**.** *Per ogni distribuzione $\boldsymbol{q}$ su $[n]$, esiste $i \in [m]$ tale che l'ipotesi h_i garantisce $\ell_{\boldsymbol{q}}(h_i) \leq V_C - \gamma$ per qualche costante $\gamma > 0$.*

In altre parole, questa assunzione dice che, data una qualsiasi distribuzione $\boldsymbol{q}$, possiamo sempre trovate una funzione in $\mathcal{H}$ che garantisce un qualche

vantaggio γ rispetto alla perdita data dal valore del gioco V_C, ottenuta dalla migliore predizione effettuata ignorando $\mathcal{X}$ e $\mathcal{H}$. Non esiste una distribuzione sui dati "così difficile" da rendere inutili tutte le ipotesi; c'è sempre una ipotesi che "indovina" una parte sufficiente delle label tale da battere la predizione casuale.

Boosting game. Definiamo ora un gioco più strutturato rispetto a quello dato dalla matrice C. Sia $M \in [0,1]^{m \times n}$ tale che ogni riga i corrisponde alla ipotesi h_i e ogni colonna j corrisponde al punto x_j. Ogni casella di M è definita come

$$M_{i,j} = c\left(h_i\left(x_j\right), f\left(x_j\right)\right), \quad \forall i \in [m], \forall j \in [n]$$

In altre parole, $M_{i,j}$ è il costo della predizione $h_i(x_j)$ data dall'ipotesi h_i sul punto x_j. Si può osservare come $\ell_{\boldsymbol{q}}(h_i) = (M\boldsymbol{q})_i$ (perdita attesa dell'ipotesi h_i rispetto $\boldsymbol{q}$, con $()_i$ si intende prendere solo la i-esima riga), quindi la weak learning assumption può essere riscritta come

$$\max_{\boldsymbol{q}} \min_i \ell_{\boldsymbol{q}}(h_i) = \max_{\boldsymbol{q}} \min_i (M\boldsymbol{q})_i \leq V_C - \gamma$$

Per il teorema minimax di von Neumann, il lato sinistro della disuguaglianza diventa

$$\max_{\boldsymbol{q}} \min_i (M\boldsymbol{q})_i = \max_{\boldsymbol{q}} \min_{\boldsymbol{p}} \boldsymbol{p}^\top M\boldsymbol{q} = \min_{\boldsymbol{p}} \max_{\boldsymbol{q}} \boldsymbol{p}^\top M\boldsymbol{q} = \min_{\boldsymbol{p}} \max_j \left(M^\top \boldsymbol{p}\right)_j$$

di conseguenza, assieme alla disuguaglianza data dalla weak-learning assumption, abbiamo equivalentemente che

$$\min_{\boldsymbol{p}} \max_j \left(M^\top \boldsymbol{p}\right)_j \leq V_C - \gamma$$

In altre parole, esiste una distribuzione $\boldsymbol{p}$ sugli indici $[m]$ dell'ipotesi tale che l'*ipotesi randomizzata* h_I ottenuta campionando $I \sim \boldsymbol{p}$ ha costo atteso al più $V_C - \gamma$ su qualsiasi punto x_j. Definiamo

$$\boldsymbol{p}^\star = \arg\min_{\boldsymbol{p}} \max_j \left(M^\top \boldsymbol{p}\right)_j \tag{$\dagger$}$$

ovvero la distribuzione di cui sopra. Si noti che può essere calcolata efficientemente tramite un programma lineare.

Voto di maggioranza cost-sensitive. Si può pensare alla strategia mista $p^\star = (p_1^\star, \ldots, p_m^\star)^\top$ come a dei pesi per le ipotesi in $\mathcal{H}$, la quale fornisce peso maggiore alle ipotesi che risultano in costo minore sui punti di $\mathcal{X}$ (come definito dalla matrice M).

Quindi, data $\boldsymbol{p}^\star$, possiamo trovare un modo deterministico per assegnare i label. Dato un qualsiasi punto $x \in \mathcal{X}$, l'idea è quella di testare l'ipotesi randomizzata h_I su ognuno delle due possibili label in $\mathcal{Y}$ e calcolare il costo atteso. In altre parole, per ogni $y \in \mathcal{Y}$ calcoliamo

$$\mathbb{E}_{I \sim \boldsymbol{p}^\star} \left[c\left(h_I\left(x\right), y\right) \right] = \sum_i p_i^\star c\left(h_i\left(x\right), y\right) = c(-y, y) \sum_{i : h_i(x) \neq y} p_i^\star$$

Intuitivamente, vogliamo scegliere la label y che minimizza tale costo atteso. Se una h_i è "brava" avrà $\boldsymbol{p}^\star$ alto e di conseguenza peso maggiore nel voto finale. In altre parole, l'algoritmo decide, per ogni punto $x \in \mathcal{X}$, qual'è l'etichetta $y \in \mathcal{Y}$ corretta secondo il voto della maggioranza delle ipotesi $h \in \mathcal{H}$.

L'indice finale usato per l'assegnamento diventa quindi

$$h^\star(x) = \arg\min_{y \in \mathcal{Y}} c\left(-y, y\right) \sum_{i : h_i(x) \neq y} p_i^\star, \qquad \forall x \in \mathcal{X}$$

ovvero, $h^\star$ sceglie il label y che la maggioranza di $\mathcal{H}$ pesata su $\boldsymbol{p}^\star$ indovina correttamente, dopo aver considerato il contributo dei due costi non negativi $c(-1, +1)$ e $c(+1, -1)$.

Per provare che $h^\star$ è effettivamente il "predittore perfetto" rispetto alla ground-truth f, dobbiamo provare il fatto seguente sul valore del binary prediction game C.

Fatto 5.5.1. *Il binary prediction game C ha $V_C \leq \max\{\alpha c^+, (1-\alpha)c^-\}$ per ogni $\alpha \in [0, 1]$.*

Si tratta di un limite superiore al valore del gioco; indipendentemente dalla strategia scelta, il costo atteso nel caso peggiore non può superare il costo di predire sempre una label fissa.

Siamo ora pronti a provare il teorema.

Teorema 5.5.1. *La funzione $h^\star$ è uguale a f.*

Dimostrazione. Per assurdo, assumiamo $h^\star \neq f$. Questo vuol dire che esiste indice $k \in [n]$ tale che $h^\star(x_k) \neq f(x_k)$ ($h^\star$ sbaglia su x_k). Sia $y_k = f(x_k)$ e definiamo

$$w^- = \sum_i p_i^\star c\left(h_i(x_k), y_k\right) = \sum_{i:h_i(x_k)\neq y_k} p_i^\star$$

(ovvero la somma dei pesi $p_i^\star$ delle ipotesi che sbagliano su x_k) e

$$w^+ = \sum_{i:h_i(x_k)=y_k} p_i^\star = 1 - w^-$$

(ovvero la somma dei pesi delle ipotesi che indovinano).

Quindi, abbiamo che $h^\star(x_k) \neq f(x_k)$ corrisponde a $h^\star(x_k) = -y_k$. Usando la definizione di $h^\star$ questo vuol dire che

$$\left(1 - w^-\right) c\left(y_k, -y_k\right) = w^+ c\left(y_k, -y_k\right) \leq w^- c\left(-y_k, y_k\right)$$

ovvero, il costo atteso per la label giusta era maggiore di quello per la label sbagliata.

Di conseguenza abbiamo che

$$\begin{aligned} w^- c\left(-y_k, y_k\right) &= \max\left\{w^- c\left(-y_k, y_k\right), \left(1 - w^-\right) c\left(y_k, -y_k\right)\right\} \\ &\geq V_C \end{aligned} \tag{5.5.1}$$

dove la disuguaglianza è data dal Fatto 5.5.1. Questo è un lower bound.

D'altra parte

$$\begin{aligned} w^- c(y_k, -y_k) &= \sum_i p_i^\star c\left(h_i\left(x_k\right), y_k\right) && \text{(def di } w^-\text{)} \\ &= \left(M^\top \boldsymbol{p}^\star\right)_k \\ &\leq \max_j \left(M^\top \boldsymbol{p}^\star\right)_j \\ &= \min_{\boldsymbol{p}} \max_j \left(M^\top \boldsymbol{p}\right)_j && \text{(def di } \boldsymbol{p}^\star\text{)} \\ &\leq V_C - \gamma && (5.5.1) \end{aligned}$$

dove l'ultima disuguaglianza è data dalla weak-learning assumption (Assunzione 5.5.1). Questo è un upper bound.

Combinando le due disuguaglianze, si ottiene

$$V_C \leq w^- c^+ \leq V_C - \gamma$$

il che è una contraddizione dato che $\gamma > 0$. □