

A.A. 2023/24

# ESTRAZIONE E INTEGRAZIONE DI CONOSCENZA DAI DATI

PROF. SALA

FABS :)



## NOTA

**Questi appunti/sbobinatura/versione “discorsiva” delle slides sono per mia utilità personale,**  
quindi pur avendole revisionate potrebbero essere ancora presenti typos, commenti/aggiunte personali (che anzi, lascio  
di proposito) e nel caso peggiore qualche inesattezza!

Comunque spero siano utili! 

Attenzione: le lezioni del prof sono *piuttosto caotiche*, quindi fare una vera e propria sbobinatura è un po' impossibile.  
Faccio del mio meglio per scrivere qualcosa di utile.

Appunti e puttanate sono scritte *principalmente* in questo stile.

NOTA: QUESTO FILE è UN PO' MENO ACCURATO DEL SOLITO. CONSIGLIO DI USARLO COME STARTING POINT, E NON  
COME BIBBIA DEL CORSO.

Questo file fa parte della mia collezione di sbobinature,  
che è disponibile (e modificabile!) insieme ad altre in questa repo:  
<https://github.com/fabfabretti/sbominamento-seriale-uniVR>



## INDICE

NOTA .....	3
Indice.....	5
Introduzione.....	6
Regole di associazione e pattern mining .....	8
Applicazioni delle regole di associazione.....	10
Recuperare.....	<b>Error! Bookmark not defined.</b>
Recuperare.....	16
Clustering .....	17
Classificazione.....	20
Alberi e foreste .....	21
Applicazione delle random forest ad altri contesti.....	26
Text mining .....	28
Boosting.....	30
Sequence trees.....	33
Regressione lineare.....	39
Inductive conformative prediction.....	41
Appendici .....	46
Assignments.....	49
Sezione di laboratorio: codice visto in classe .....	55

## INTRODUZIONE

Abbiamo delle tecnologie che ci consentono di **raccogliere un ammontare di dati impensabile** fino a poco fa:

- Sia per il trasferimento fisico dei dati
- Sia per il fatto che abbiamo l'infrastruttura per sostenere.

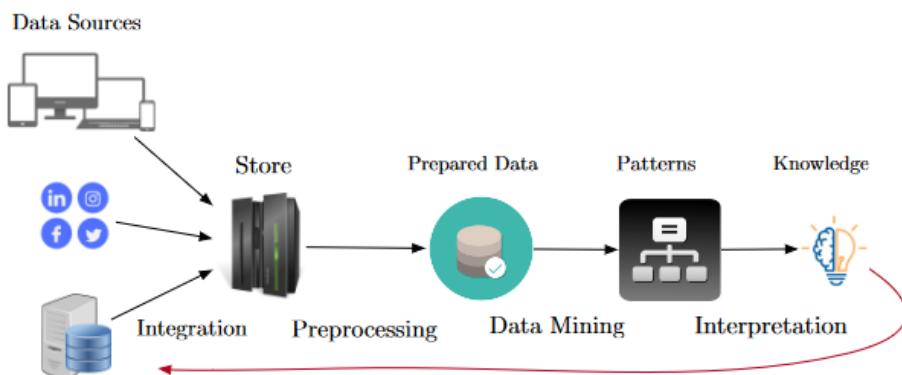
Il problema fondamentale che fa nascere la data science è che noi collezioniamo una marea di dati per ragioni operative, e ci si accorge che **da queste informazioni si possono estrarre i dati**. Solitamente è un lavoro postumo: prima li si raccolgono, poi ci si fa un'analisi; magari proprio perché un umano si rende conto della presenza di pattern, e ci si chiede se si possa automatizzare (sì).

Il problema è che poi quando si esce dall'area del dataset perfetto, si scopre un mondo diverso: le etichette non sono date, i dati hanno dei problemi e non sono informativi a sufficienza; alle volte il modello di dati da testare è un task non semplice.

Insomma, siamo **ricchi di dati ma poveri di conoscenza**. Solitamente si parte da una conoscenza superficiale, e nel corso analizzeremo le tecniche generali per arricchire i dati (integrarli, pulirli..) e estrarne informazioni.

### *Knowledge discovery*

Estrazione non banale di informazioni implicite, precedentemente sconosciute e potenzialmente utili, dai dati.



Le fasi sono:

- **Integrazione:** es. ho un database sulla somministrazione dei farmaci, ma i farmaci hanno nome commerciale. Aggiungo il principio attivo. Coinvolge anche il capire quali dati escludere.
- **Preprocessing:** potrei voler approssimare valori nulli, avere problemi di numerosità... voglio preparare i dati per il task.
  - **Data transformation:** i dati sono selezionati, trasformati e caricati in un data store, che gestisce con un formato specifico della task di data mining in corso. Questo data store viene detto “prepared data”.
  - **Data profiling:** prima di iniziare il data mining vero e proprio, i data scientist dovrebbero fare delle analisi preliminari con lo scopo di:
    - Trovare inconsistenze e gestirle
    - Determinare la fattibilità delle tecniche
    - Ecludere informazioni triviali che non vale la pena minare (=too much effort)
- **Data mining:** trovo dei pattern o classifico (=do etichette); trovare correlazioni... Il data mining vero e proprio applica vari modelli (classificazione, estrazione di regole..). Qui ci si concentra su:
  - **Identificare i modelli**
  - **Identificare le metriche**
  - **Progettare un modo sistematico di testare i modelli**
  - **Eseguire e valutare i risultati.**
- **Interpretazione:** è principalmente delegata all'esperto di dominio; ma poiché spesso egli non capisce la parte di data-mining, i risultati di solito devono essere:

- **Human readable:** devono essere leggibili via grafici, animazioni, tabelle... Anche l'utente che non ha conoscenza di ML deve capire.
- **Browsable:** bisogna considerare che il domain expert potrebbe volerli challengare, e quindi si dovrebbe permettere di viaggiare fra livelli di dettaglio.
- **Sintetici:** nonostante il fatto che potrebbero essere complessi e con tanti ivelli di dettaglio, ma le conclusioni dovrebbero essere più ad alto livello.

Quando si estrae qualcosa di nuovo dai dati, di solito quello che torna è che vengono integrati ai dati e si riesce ad estrarre ancora più informazioni.

Quindi, altre fasi sono:

- Identificare le sorgenti
- Capire come estrarle (potrei avere accesso limitato)
- Capire come integrarle.

## Strumenti

Teoria	Strumenti effettivi
<b>Integrazione di data</b> <ul style="list-style-type: none"> <li>• Da sorgenti eterogenee</li> <li>• Missing values</li> <li>• Discretizzazione Interoperabilità</li> </ul>	<ul style="list-style-type: none"> <li>- Pentaho Data Integration, ORM (Object Relational Mapping)</li> <li>- Clustering libraries</li> <li>- Conformal regression</li> <li>- Docker</li> <li>- Redis</li> <li>- Neo4j</li> <li>- XML Manipulation</li> </ul>
<b>Analisi e profiling di dati</b> <ul style="list-style-type: none"> <li>• OLAP Cubes</li> <li>• Dashboards</li> <li>• Entropia</li> <li>• Selezione di features</li> <li>• Rappresentazioni grafiche</li> <li>• Principal component analysis (PCA &lt;3)</li> </ul>	<ul style="list-style-type: none"> <li>- Siku Analytics</li> <li>- Statistical libraries</li> <li>- Learning libraries</li> <li>- Graphical libraries</li> <li>- Dashboard libraries</li> </ul>
<b>Unsupervised learning (knowledge extraction):</b> <ul style="list-style-type: none"> <li>• Association rules</li> <li>• Pattern mining</li> <li>• Neural models for event similarity</li> <li>• Process discovery techniques</li> <li>• Dynamic time warping</li> <li>• Clustering techniques</li> </ul>	<ul style="list-style-type: none"> <li>- Mlxtend</li> <li>- Tensorflow/Keras</li> <li>- Genshim</li> <li>- Apromore</li> <li>- Clustering libraries</li> <li>- DTW Library</li> </ul>
<b>Supervised learning (classificazione e regressione)</b> <ul style="list-style-type: none"> <li>• Valutazione di classificatori e regressori</li> <li>• Explainability</li> <li>• Ottimizzazione di iperparametri</li> <li>• Selezione di modello</li> <li>• Conformal prediction and regression</li> <li>• Modelli per dati temporali e sequenze</li> </ul>	<ul style="list-style-type: none"> <li>- scikit-learn</li> <li>- tensorflow/Keras</li> <li>- Nonconformist</li> <li>- Explainability libraries</li> </ul>

## REGOLE DI ASSOCIAZIONE E PATTERN MINING

!!! Ero assente !!!

### Definizioni

Dato un insieme di items  $It$ , una transazione  $tr$  è un sottoinsieme di  $It$ .

In maniera equivalente,

$$tr : It \rightarrow \{0,1\}$$

con

$$tr(it) = \begin{cases} 1 & it \in tr \\ 0 & \text{else} \end{cases}$$

Dato un insieme  $It$ , definiamo l'insieme di tutte le possibili transazioni come  $\mathbb{T}r = \{tr : tr \in It\}$ .  $|\mathbb{T}r| = 2^{|It|}$ .

*Insieme di transazioni*

Un insieme di transazioni  $Td : \mathbb{T}r \rightarrow \mathbb{N}$  è un multiset di transazioni, dove

$$|Td| = \sum_{tr \in \mathbb{T}r} Td(tr)$$

*Supporto*

Il supporto di  $tr$  in  $Td$  è la percentuale di transazioni  $tr'$  di  $Td$  che hanno  $tr$  come loro subset.

$$Sup(Td, tr) = \frac{\sum_{tr' \in \mathbb{T}r, tr \subseteq tr'} Td(tr')}{|Td|}$$

*Insieme degli item frequenti*

Data una certa soglia  $0 \leq \epsilon \leq 1$ ,  $tr \in \mathbb{T}r$  è un frequent itemset  $\Leftrightarrow Sup(Td, tr) \geq \epsilon$ .

Quindi, il frequent itemset di  $Td$  è l'insieme

$$\mathbb{F}i = \{tr \in \mathbb{T}r : Sup(Td, tr) \geq \epsilon\}$$

Proprietà

$\subseteq$ -downward closure di  $\mathbb{F}i$

Per ogni  $tr \in \mathbb{F}i$  abbiamo che  $tr \in \mathbb{F}i$  è vero per ogni  $tr' \subseteq tr$

*Conseguenza 1: maximal proper subsets*

Se  $tr \in \mathbb{F}i$  allora per ogni  $it \in tr$  ho che  $(tr \setminus \{it\}) \in \mathbb{F}i$

*Conseguenza 2: candidato k-size*

Dati i maximal proper subsets di  $tr_1 \dots tr_k$ . Se esiste  $1 \leq i \leq k$  tale che  $tr_i \notin \mathbb{F}i$ , allora  $tr \notin \mathbb{F}i$

### Generazione degli itemset: algoritmo Apriori

L'obiettivo principale dell'algoritmo Apriori è **identificare gli insiemi di articoli frequenti in un database transazionale**.

Un insieme di articoli frequenti è un **gruppo di articoli che si verificano frequentemente insieme nelle transazioni**.

Questi insiemi di articoli frequenti sono cruciali per trovare le regole di associazione, che possono rivelare relazioni tra diversi articoli.

Questo algoritmo appartiene a Agrawal e Srikant, e dimostra che le regole di associazione possono essere generate in una timescale realistica.

L'algoritmo Apriori è un metodo utilizzato nell'ambito del data mining per scoprire associazioni tra diversi elementi all'interno di un grande insieme di dati. Ecco come funziona in modo più dettagliato:

1. **Cosa stiamo cercando:** Supponiamo di avere un grande insieme di dati, come le ricevute degli acquisti dei clienti in un negozio di giocattoli. Vogliamo scoprire quali prodotti sono spesso acquistati insieme, in modo da poter suggerire prodotti correlati o ottimizzare la disposizione dei prodotti in negozio.
2. **Creazione di insiemi di oggetti:** Innanzitutto, creiamo insiemi di oggetti. Ad esempio, immaginiamo che ogni ricevuta rappresenti un insieme di giocattoli acquistati da un cliente.
3. **Generazione di itemset frequenti:** L'algoritmo Apriori inizia cercando gli "itemset frequenti". Questi sono gruppi di oggetti che sono spesso acquistati insieme da diverse persone. L'algoritmo esegue queste operazioni in modo iterativo:
  1. **Itemset di lunghezza 1:** Inizia cercando oggetti singoli che sono frequentemente acquistati. Ad esempio, quante volte è stata acquistata una "bambola" da sola? L'itemset è selezionato o meno solo se il suo supporto supera una certa soglia.
  2. **Itemset di lunghezza 2:** Poi guarda coppie di oggetti per vedere quali coppie sono frequenti. Ad esempio, quante volte "bambola" e "vestiti per la bambola" sono stati acquistati insieme?
  3. **Itemset di lunghezza 3:** Continua questo processo con insiemi di oggetti di lunghezza sempre maggiore.
  4. **A questo punto si continua finché una certa lunghezza (k) non trova nessun itemset.**

```

Crea  $L_1$  = insieme di itemsets supportati di cardinalità 1
k = 2
while ( $L_{k-1} \neq \emptyset$ ) {
  Crea  $C_k$  con  $L_{k-1}$ 
  Togli tutti gli itemsets di  $C_k$  che non sono supportati, creando  $L_k$ 
  k++
}
Fai l'unione di tutti i  $L_i$ .
  
```

**APRIORI**  $Td, \epsilon \Rightarrow F_i$

$$F_i \leftarrow \emptyset$$

$$It \leftarrow \bigcup_{tr \in Td} tr$$

$$k \leftarrow 1$$

$$\text{candidati} \leftarrow \{\{it\}: it \in It\}$$

**REPEAT**

$$F_{ik} \leftarrow \{tr \in \text{candidati}, \text{Sup}(Tr, u) > \epsilon\}$$

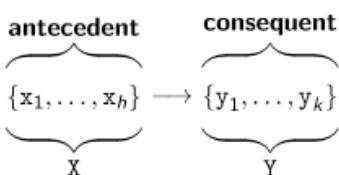
$$F_i \leftarrow F_i \cup F_{ik}$$

$$\text{candidati} \leftarrow \left\{ \begin{array}{l} tr: \\ \left| \left\{ tr': tr \subseteq tr' \wedge \left| \left\{ tr' : tr' \subseteq tr \right\} \right| = k+1 \right\} \right| \end{array} \right\}$$

$$k \leftarrow k+1$$

until candidati  $\neq \emptyset$

## Regola di associazione



Def. Supporto di una regola di associazione

Il supporto di una regola di associazione è il **supporto dell'unione dei suoi antecedenti e conseguenti**.

$$\text{Sup}(X \rightarrow Y) = \text{Sup}(X \cup Y)$$

Def. Confidence di una regola

La confidence di una regola di associazione è il **rappporto fra il supporto della regola e il supporto dell'antecedente**.

$$\text{Conf}(X \rightarrow Y) = \frac{\text{Sup}(X \cup Y)}{\text{Sup}(X)}$$

Data una treshold di confidenza  $0 \leq \delta \leq 1$ :

$$Td \models X \rightarrow Y \Leftrightarrow \text{Sup}(X \rightarrow Y) \geq \epsilon \text{ and } \text{Conf}(X \rightarrow Y) \geq \delta$$

## Association Rule

**syntax:**  $X \rightarrow Y$

$$X, Y \subseteq It$$

**constraints:**  $X, Y \neq \emptyset$   
 $X \cap Y = \emptyset$

## APPLICAZIONI DELLE REGOLE DI ASSOCIAZIONE

10-10-2023

Visti i concetti sulle regole di associazione, aggiungiamo un po' di spicce incrociandole con altri ambiti.

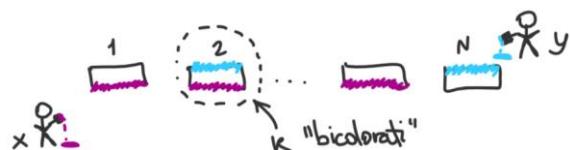
### Statistica

Possiamo applicare il mondo della statistica alle regole dell'associazione, nello specifico introducendovi il concetto di **p-value**.

Consideriamo un universo  $U$  con cardinalità  $|U| = N$  e  $0 < k < N$ . I sottoinsiemi distinti di dimensione  $k$  in  $U$  saranno il binomiale.

$$\binom{N}{k} = \frac{N!}{k!(N-k)!}$$

Immaginiamo uno scenario in cui due persone  $X$  e  $Y$  devono colorare una serie di muri; ognuna delle due può colorare almeno ciascun lato di muro. Dato un  $k$ , qual è la probabilità che 2 persone abbiano colorato  $k$  pareti su entrambi i lati sapendo quante lati ha dipinto ciascuno dei due?



Sarà:

Probabilità di aver colorato ugualmente la prima parete e diverse le altre

+ Probabilità di aver colorato ugualmente la seconda parete e diverse le altre

+ ...

→ **combinatoria geometrica**

Possiamo individuare, per ogni muro, se è colorato da  $X$  e da  $Y$ ; e possiamo contare il numero di mura colorate da uno, dall'altro, da entrambi o da nessuno nella seguente tabella:

$X$	$\neg X$	$Y$	$\neg Y$	
$C_{xy}$		$C_{x-y}$		Voglio calcolare la $P$ che $C_{xy} = k$
	$C_{-xy}$		$C_{-x-y}$	

Possiamo calcolare questa probabilità come

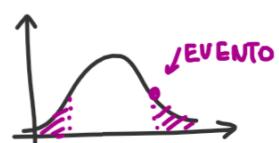
$$P(C_{xy} = k) = \frac{\binom{C_y}{k} \binom{N - C_y}{C_x - k}}{\binom{N}{C_y}}$$

Li interpretiamo come "scelta di x"  
Dati tutti i casi in cui  $x$  colora, quanti sono quelli in cui  $y$  si "incontra" per avere  $k$  muri in comune?  
modi in cui posso colorare le pareti  $y$   
modi in cui posso colorare le pareti fuori dalla colorazione di  $y$

Def. p-value

Probabilità che accada un certo evento + somma degli eventi più rari o equamente rari.

Più il p-value è basso, più l'ipotesi è significativa e non dipende "solo" dalla distribuzione.



Come applico questo concetto a una regola?

Supponiamo che sia importante il supporto di  $X \cup Y$ . In questo caso il p-value sarebbe la somma dei casi più rari, quindi le probabilità che  $C_{XY} > k$ .

$$pvalue(X \rightarrow Y) = P(C_{XY} \geq N_{XY}) = 1 - \sum_{k=0}^{N_{XY}-1} P(C_{XY} = k)$$

! Attenzione: è commutativa!  $pvalue(X \rightarrow Y) = pvalue(Y \rightarrow X)$ .

**Quindi non è utile per filtrare, ma lo è per verificare se la regola è importante.**

## Teoria dell'informazione

### Entropia

*Def. Entropia*

Misura della quantità di informazione che può essere trasmessa bitweise.

Prendiamo come sempio il gioco "Indovina chi". In termini più tecnici, con ogni domanda chiediamo una feature, e eliminiamo parte dell'insieme. Il punto dell'entropia è: qual è il numero minimo di domande da fare per arrivare alla soluzione?

"Avessi saputo l'entropia avrei potuto vincere merendine e scommesse a mani basse fra i miei amici!"

In generale, mi conviene codificare l'informazione in  $\log_2 N$  bit (=domande) a patto che gli oggetti siano distribuiti sulla normale. Per esempio: se ho 8 oggetti, faccio sempre 3 domande: mando 0/1 se è nella prima/seconda metà dell'array degli oggetti possibili.



Le cose però **cambiano se le feature hanno una distribuzione diversa**. Prendiamo un esempio, dove ciascun simbolo ha una certa frequenza – come, pensandoci, la lingua italiana:

A	0.50	$\sum = 1$
B	0.25	
C	0.15	
D	0.10	

Potrei comunque fare 2 ( $\log_2 4$ ) domande, oppure potrei **sfruttare la distribuzione**: nei casi più frequenti uso un solo bit, e ne uso tre per quelli più rari.

A	0.50 ... 1
B	0.25 ... 01
C	0.15 ... 001
D	0.10 ... 000

In questo modo, la lunghezza media del messaggio è

$$\sum (\text{probabilità} \cdot \# \text{bit}) = 0.5 \cdot 1 + 0.25 \cdot 2 + (0.15 + 0.10) \cdot 3 = 1.75$$

Quindi arrivo a una lunghezza media di 1.75 anziché 2 bit! 🎉🎉🎉

Posso quindi definire l'entropia come il # di bit che servono a trasmettere il messaggio in media.

$$H(x) = \sum_{x \in X} P(x, x) \cdot \log_2 \left( \frac{1}{P(x, x)} \right)$$

$\overbrace{\quad \quad \quad}^{\#\text{bit usati}}$  più un simbolo è raro, meno influenza

$= -\sum_{x \in X} P(x, x) \log_2 (P(x, x))$  lo simbolo σ (es. "A")

## Cross Entropy

Ora supponiamo che la mia distribuzione con cui costruisco la codifica sia una stima, e che sia diversa rispetto a un'altra distribuzione che invece è quella reale. Qual è l'entropia rispetto a quest'altra distribuzione?

Riprendiamo l'esempio di prima, aggiungendo una stima:

	REALE STIMA ... codifica	
A	0.25	0.50 ... 1
B	0.50	0.25 ... 01
C	0.15	0.15 ... 001
D	0.10	0.10 ... 000

A questo punto, la lunghezza reale media del messaggio risulta più lunga:

$$0.25 \cdot 1 + 0.50 \cdot 2 + (0.15 + 0.10) \cdot 3 = 2$$

La cross entropia guarda proprio il caso in cui mando un messaggio con una distribuzione diversa rispetto a quella usata nella codifica.

$$H(x, y) = \sum_{x \in X} p(y, x) \log_2 \left( \frac{1}{p(x, y)} \right)$$

distribuzione  
 su cui ho  
 calcolato la  
CODIFICA

 ↓  
 distribuzione  
REALE

! Non è simmetrica !

In Python possiamo applicare questa formula con una pucciosissima list comprehension + reduce.

```
reduce(lambda x, y : x+y, [x*log2(x) for x in [0.50, 0.25, 0.15, 0.10]])
```

"Questa non è scienza, è alta pasticceria"

## Kullback-Leibler divergence

La stima qualitativa della differenza è detta divergenza (=di quanto cambia l'entropia?). Questa differenza si chiama Kullback-Leibler divergence:

$$KL(X || Y) = \sum_{x \in X} p(x, \text{reale}) \log_2 \left( \frac{p(x, \text{reale})}{p(x, \text{codifica})} \right)$$

Un esempio di applicazione di questo concetto è il Machine Learning. In ML, come funzione di loss da minimizzare, lo scarto quadratico fa molto cagare, e quindi si usa proprio KL:

$$\begin{aligned}
 \text{LOSS} &= \sum_{x \in X_1} p(x, x) \log_2 \left( \frac{p(x, x)}{p(y, x)} \right) + \sum_{x \in X_0} (1-p(x, x)) \log_2 \left( \frac{1-p(x, x)}{p(y, x)} \right) \\
 &= \sum_{x_1} \log_2 \frac{1}{p(y, x)} + \sum_{x_0} \log_2 \frac{1}{1-p(y, x)}
 \end{aligned}$$

Sample che devono rispondere 1      sample che devono rispondere 0  
 ↓  
 vira a 1      ↓  
 cioè so che in  $x_1$       perché so che in  $x_0$   
 ho sempre risposta 1      ho sempre risposta 0

Quindi, per ogni  $X$  del sample sommo questa funzione loss e cerco di minimizzarla.

Usiamo questa e non lo scarto quadratico proprio perché questa è esponenziale: per una minima divergenza migliora di molto, mentre una MSQ migliora (=diminuisce) "solo" quadraticamente.

## J Measure

Collegandoci a ciò che abbiamo visto sulle regole, possiamo dire che data una regola  $X \rightarrow Y$ :

- $P(Y)$  è il supporto
- $P(Y|X)$  è la confidenza

Definiamo quindi una nuova misura: la J Measure.

### Def. J Measure

È il contenuto informativo che  $X$  mi dà rispetto a  $Y$ , e la calcolo come la divergenza di informazioni nel determinare  $X$  sapendo  $Y$  rispetto a calcolarlo senza saperlo.

$$\begin{aligned} J(X \rightarrow Y) &= P(x) \cdot K\mathcal{I}(P(y|x) || P(y)) \\ &= P(x) \cdot \left( P(y|x) \log_2 \left( \frac{P(y|x)}{P(y)} \right) + 1 - P(y|x) \log_2 \left( \frac{1 - P(y|x)}{1 - P(y)} \right) \right) \end{aligned}$$

! Più la J Measure è alta, meglio è. Inoltre, è direzionale.

In colloquiale, se un canale mi manda  $Y$ , quanto il fatto di avere un canale che mi manda  $X$  aiuta la trasmissione? **Questo è importante, perché se sapere  $X$  cambia di molto le probabilità di  $Y$  allora posso cambiare la codifica e risparmiare bit.**



Per esprimere in termini di supporto, posso prendere l'espressione sopra e sostituire:

$$P(x) = \text{Sup}(x) \quad P(x|y) = \frac{\text{Sup}(x \cup y)}{\text{Sup}(x)}$$

"Avete fatto teoria dei giochi? No? Ottimo, meno sapete meglio è! Altrimenti non posso plasmare le vostre giovani menti"

La J Measure gode di alcune proprietà utili:

### Massimo stabile

La J measure ha un **massimo stabile che dipende esclusivamente da  $X$** . Nel dataset, prendendo il supporto per  $X$ , tutte le possibili  $J$  che hanno  $X$  per antecedente saranno al più

$$J(X \rightarrow Y) \leq \text{Sup}(x) \log_2 \left( \frac{1}{\text{Sup}(x)} \right)$$

Avere un massimo teorico è utile: infatti, così possiamo dividere il valore ottenuto per il massimo possibile, riconducendoci a un valore in  $[0,1]$ .

### Specialization bound

Preso un  $X \rightarrow Y$ , vogliamo poter fare pruning e scegliere le regole migliori. Mi chiedo quindi: ha senso specializzare (=restringere l'ipotesi aggiungendo altre lettere)?

In generale, testare una regola può essere molto costoso. Per esempio, dato un database di  $N$  attributi e  $M$  righe, potrei trovarmi a dover testare su un problema NPC, con complessità quinid esponenziale – come succede sui grafi. È importante quindi definire in cosa è la complessità computazionale: se un problema è NPC sugli attributi, che sono solitamente una quantità moderata, può essere ancora fattibile attuando delle euristiche. Al contrario, se la complessità è sulle righe... arrivederci e basta. In questi casi, potrebbe essere molto costoso non andare a tentoni nel trovare le regole.

Prendendo quindi un caso in cui la  $J(X \rightarrow Y)$  è soddisfacente, mi chiedo: ha senso specializzare  $X \rightarrow Y$  in  $X \cup Z \rightarrow Y$ , dove  $Z$  è un insieme di qualsiasi attributo? Quanto posso pensare di migliorare facendolo?

$$\begin{array}{c} \text{MAX} \\ \boxed{\text{potenziale } X \cup Z \rightarrow Y} \\ \boxed{X \rightarrow Y} \end{array}$$

Ho un ostacolo: **so già** che quanto posso essere fortunato e raffinare la soluzione è al massimo quanto espresso dalla formula

$$J(X \cup Z \rightarrow Y) \leq I(X) \cdot \max \left\{ \frac{P(Y|X) \log_2 \left( \frac{1}{P(Y)} \right)}{1 - P(Y|X) \log_2 \left( \frac{1}{1 - P(Y)} \right)} \right\}$$

Quindi, quando questo limite superiore è molto piccolo so già che non ha senso cercare di specializzare. Anche perché **specializzare fa perdere supporto!**

## Teoria dei giochi

Questa robina qua è valsa un nobel in economia e uno in medicina :)

*Indovinello!*

Ho 10 pirati che hanno trovato un forziere da 100 monete. I pirati hanno una gerarchia stretta: *capitano > ... > mozzo*. Per spartire il tesoro, il pirata di grado più alto propone una sua spartizione. Dopo di che si vota: se  $\geq$  metà dei pirati vota a favore, la spartizione avviene; altrimenti, gli altri pirati ammazzano quello di grado più alto e il procedimento si ripete con il secondo in grado.

Proviamo a ragionare induttivamente, da informatici:

- 1)  $P_1$  prende tutto, vota + e se ne va  
100
- 2)  $P_1, P_2$  prende tutto, vota +  $(1/2\checkmark)$  e se ne va  
100
- 3)  $P_1, P_2, P_3$  A  $P_3$  non conviene che muova  $P_1$ , poiché  
99 1  $P_1$  terrebbe tutto. Quindi, basta 1 moneta  
poiché l'offerta sia vantaggiosa e vota e  
fornisce  $(2/3\checkmark)$

Il contesto è che abbiamo dei calciatori in offerta sul calciomercato; ognuno guadagna un certo tot. Dobbiamo costruire una squadra. Ciascun giocatore garantisce un certo tot di goal/punti, ma certe combinazioni non vanno bene: per esempio, alcuni giocatori da soli porterebbero un sacco di punti, ma fanno le primedonne e quando giocano in squadra non portano tutto questo vantaggio. Oppure, alcuni giocatori sono buoni ma per alcuni loro difetti, ad esempio un'altezza non sufficiente, da soli non formano una squadra valida.

"Lo so bene, perché ho provato a giocare [a basket] e sono alto 1.20"

Quindi sappiamo che decidere chi prendere non dipende dal valore intrinseco di ogni giocatore, ma piuttosto dalla **combinazione totale del team**. Questo tipo di funzione è detta **CPO (coalition payoff function)**, ed è una funzione  $CPO : 2^X \rightarrow \mathbb{R}$  che associa ad ogni sottoinsieme di  $X$  (possibili giocatori) un certo punteggio.

Per sempio, se  $X = \{a, b, c\}$ , potrei avere

$$\begin{aligned} CPO(\{a, b, c\}) &= 10 \rightarrow b + c \text{ non} \\ CPO(\{a, b\}) &= 15 \text{ vanno bene} \\ CPO(\{a, c\}) &= 12 \text{ insieme} \end{aligned}$$

"Io ho perso un sacco di soldi al fantacalcio \*risata lunga e disperata\* Non ho mai più giocato di azzardo in vita mia. Sto scoprendo adesso a 40 anni come avrei dovuto vivere l'infanzia"

## Fairness

Avendo preso i soggetti, vogliamo decidere il valore sul mercato dei singoli. Per esempio, un giocatore può essere fortissimo, ma litiga con tutti abbassando il valore di ogni coalizione che lo include. Oppure potrebbe esserci un certo esperto e essenziale, senza il quale la squadra non ha valore, e quindi tale per cui  $CPO(X) > 0$  sse  $e \in X$ ; ma che se metto nella coalizione più di un solo e il valore sul mercato si abbassa perché non è più così essenziale.

Per decidere questo valore sul mercato, devo **rispettare tre principi che assicurano la fairness**. L'obiettivo è **dare valori di mercato fair sapendo la CPO**. Indichiamo il valore di mercato con  $s$ .

### Simmetria

Per ogni agente, se due agenti hanno la stessa competenza in ogni coalizione, allora devono avere lo stesso valore di mercato.

$$\forall X \in \mathcal{X} : (\underbrace{\{x, y\}}_{\text{coalizione}} \cap X = \emptyset \wedge \overline{CPO}(\{x, y\}) - CPO(x) = CPO(\{x, y\}) - CPO(y))$$

$\Downarrow$

$$S_{CPO, x} = S_{CPO, y}$$

Dummy “È il nome giusto”

È il caso di una persona che, se è aggiunto a una coalizione, aggiunge esattamente quello che farebbe se giocasse da solo; nella nostra interpretazione, è equivalente alla gente che **vale un botto ma gioca di merda quando è con gli altri**.

$$X \subseteq \mathcal{X} : (x \in X \Rightarrow CPO(X) = CPO(X \setminus \{x\}) + CPO(\{x\}))$$

Additivity “Fabiola, dove vivi? Non è stalking eh.. anzi no dai, non dirmelo”

Dati due assegnamenti sullo stesso numero di persone e la CPO delle spese e dei guadagni...

Prese due funzioni così, abbiamo calcolato i valori di mercato nelle due tranches, se sommiamo le due CPO allora il valore di mercato è esattamente la somma dei valori di mercato singoli.

COSTO	GUADAGNO	TOTALE
$CPO(x)$	$CPO'(x)$	$\overline{CPO}(x) = CPO(x) + CPO'(x)$
$\downarrow$	$\downarrow$	$\downarrow$
$S_{CPO, x}$	$S_{CPO', x}$	$S_{\overline{CPO}, x} = S_{CPO, x} + S_{CPO', x}$

In altre parole, se il valore di mercato di pagamenti + spese, che chiamiamo  $\overline{CPO}(X)$ , è esattamente la somma degli altri CPO:  $\overline{CPO}(X) = CPO(X) + CPO'(X)$ , posso dire che il valore di mercato totale  $\overline{s}_{\overline{CPO}, X} = \overline{s}_{CPO, X} + \overline{s}_{CPO', X}$ .

$$\overline{CPO}(x) = CPO(x) + CPO'(x) \Rightarrow S_{\overline{CPO}, x} = S_{CPO, x} + S_{CPO', x}$$

### Teorema di non mi ricordo

Esiste un solo assegnamento di valore di mercato che rispetta queste tre condizioni E una quarta condizione bonus, ovvero che...

$$\sum_{X \in \mathcal{X}} S_{CPO, x} = CPO(X)$$

... il valore di mercato totale è esattamente il valore di mercato dei singoli, allora so che esiste un solo assegnamento tale per cui

**...LEZIONI MANGANTI...**

## CLUSTERING

24-10-2023

### Algoritmo mean shift

È una buona alternativa a k-mean in quanto k-mean è NP, perché cerca di minimizzare. Mean-shift, al contrario, non minimizza nessuna misura: fa un procedura iterativa, che porta un aggiornamento sul centroide ottimale.

L'idea dell'algoritmo sta nel nome: **shift verso la media, ma non la media di tutti i punti**: definisco una funzione di "kerneling", di "vicinato", di  $x$ .

$$N(d) = \begin{cases} 1 & \text{se } d \leq \delta \\ 0 & \text{else} \end{cases}$$

distanza soglia

Quindi, il meanshift di  $x$  sarà

$$m(x) = \frac{\sum_{\bar{x} \in X} N(\bar{x}-x) \cdot \bar{x}}{\sum_{\bar{x} \in X} N(\bar{x}-x)}$$

Su più dimensioni posso fare la norma

L'algoritmo procede iterativamente.

#### MEAN SHIFT

$$i \leftarrow 0$$

$$X_0 \leftarrow X$$

WHILE  $\neg \Psi(X_i, \varepsilon)$ :

$$X_{i+1} = \left\{ m(x, X_i) : \forall x \in X_i \right\}$$

vicinato  
Sostituisco ad ogni punto  
la media

Bisogna, quindi, definire la distanza da usare (norma, euclidea, etc) e la soglia di tolleranza  $\varepsilon$ .

Ogni punto di un intorno sta a meno  
di  $\varepsilon$  da tutti gli altri dell'intorno.

$$\Psi(X_i, \varepsilon) = \text{true se } \forall x \forall x' \text{ t.c. } N(x-x') = 1 \rightarrow \|x-x'\| < \varepsilon$$

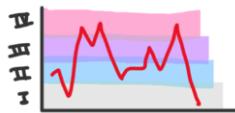
La palla di clustering è  $2\varepsilon$ .

$\varepsilon$  deve essere molto più piccola di  $d$ : l'idea è che sia **piccola quanto il "piccolo calcolabile" della macchina**, ovvero che non ha senso andare avanti di precisione perché la macchina non riesce a calcolare a quella precisione.

Praticamente ci fermiamo quando l'intorno di ciascun punto è il punto stesso (con tolleranza  $\varepsilon$ ).

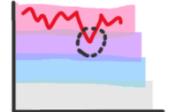
## Applicazione

Supponiamo di essere un'applicazione di corsa, che traccia i BPM durante l'attività fisica, e li memorizza su quattro fasce possibili di intensità. Il risultato è una serie temporale arbitraria (=non stazionaria).



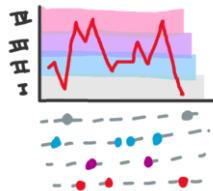
Vorrei fare in modo di “privilegiare” le misurazioni che stanno in una fascia di misurazione superiore. Volendo quindi partizionare la time series, ovvero dividerla in segmenti adiacenti che mi dicono in che fascia sono stato.

Ad esempio, con un grafichetto del tipo a destra.

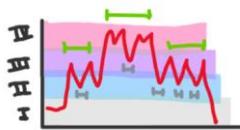


Voglio che, sebbene io sia tecnicamente tornato in terza fascia per una misurazione, il segmento sopra abbia la priorità e quindi risulti che io sono sempre stato in quarta fascia.

Potrei quindi applicare mean shift: lo lancio solo sulla time series, proiettata PER FASCIA. Vedo solo i tempi e le misurazioni sulla stessa fascia.



Applicando mean shift, ottengo un cluster che mi indica il massimo e il minimo (tempo) di ciascun cluster. A questo punto posso ricavarmi gli intervalli, e in caso di collisione prendo sempre l'intervalle di fascia più alta.



Come faccio a ricavare il parametro  $\delta$ ?

Qui non sto tentando di minimizzare la distanza dal centroide, quindi non posso applicare lo stesso metodo che ho usato in k-mean. Ci sono due grandi famiglie di metodi:

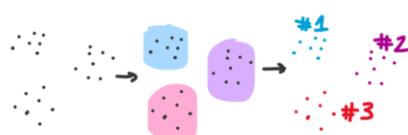
### Esterni

Si servono di una misura esterna, ovvero fondamentalmente usano il fatto che abbiano una classe associata a ogni punto. Misurano quanto i clusters generati sono omogenei rispetto all'etichetta.

### Interni

#### *Intra-cluster measure*

Etichettiamo ciascun nodo col numero del cluster che vi è stato associato.



Quindi, ho una funzione di clustering che per ogni sample mi mappa in un numero.

$$\ell: \mathbb{X} \rightarrow \mathbb{N}$$

Definiamo due funzioni:

$$P_{\text{in}} : \{(x, x') : x, x' \in \mathbb{X}, d(x) = d(x')\} \rightarrow \begin{array}{l} \text{tutte le coppie di punti} \\ \text{clusterizzate insieme} \end{array}$$

$$P_{\text{out}} : \{(x, x') : x, x' \in \mathbb{X}, d(x) \neq d(x')\} \rightarrow \begin{array}{l} \text{tutte le coppie di punti} \\ \text{non clusterizzate insieme} \end{array}$$

Date queste informazioni, possiamo definire la inter-cluster e intra-cluster measure come:

INTRA-CLUSTER

$$\sum_{(x, x') \in P_{\text{in}}} \frac{d(x, x')}{|P_{\text{in}}|} \downarrow \text{low is good}$$

INTER-CLUSTER

$$\sum_{(x, x') \in P_{\text{out}}} \frac{d(x, x')}{|P_{\text{out}}|} \uparrow \text{high is good}$$

Un punto che ha cattivi indici in entrambi i casi è un **outlier**, e posso usare questi indici per fare **anomaly detection** e tirarlo fuori.

Da queste due misure possiamo avere un indice di bontà del mio clustering:

$$\text{BONTÀ} = \frac{\text{INTRA}}{\text{INTER}} \downarrow \text{low is good}$$

L'idea è che posso fare molti calcoli, e calcolare il cluster su tante distanze possibili e poi vedere quale si comporta meglio nelle inter e intra cluster measures. Questo è detto hyperparameter optimization.

Applicazione alle regole di associazione

Clusterizzo perché se voglio fare le regole di associazione potrebbe non interessarmi differenziare "10.4" da "10.5"; ai fini delle regole è la stessa cosa, quindi posso **"unire"** tutte le misure simili in un cluster.

!!! Lo svantaggio è che questo significa dover prima trasformare i dati per avere valori significativi: così facendo mi pongo delle limitazioni, Per esempio, se ho una regola

$$X[10..20], Y[8...12] \rightarrow Z[8..12]$$

Non potrò ricavare altre regole che usano X o Y con intervalli diversi (a meno di fare casini...)

Prendendo quindi una tabella con i miei valori, dovrò **sostituire i valori reali con i valori del cluster**, e poi usare Pandas, che offre il metodo GetDummies per convertire la tabella con i valori di cluster in una tabella che abbia tutti i cluster possibili e 0/1 se la misurazione vi appartiene o meno.

	Temperature	clustering	T <sub>e</sub> C <sub>0</sub>	T <sub>e</sub> C <sub>1</sub>	T <sub>e</sub> C <sub>2</sub>	T <sub>e</sub> C <sub>3</sub>
0	30	0	0	0	0	1
1	20	1	0	0	0	0
2	15	2	0	1	0	0
3	25	3	0	0	1	0
4	-200	4	Nan	0	0	0

Se ho più di un attributo, semplicemente concateno il risultato di tutti gli attributi.

A questo punto posso **usare una qualunque libreria per vedere gli itemset frequenti per un certo supporto**, e su quelli costruisco le regole.

# CLASSIFICAZIONE

31/10/2023

Il problema della classificazione può essere formalizzato come il trovare una certa funzione tale per cui

$$f_{X,Y} : \mathbb{R}^n \rightarrow \{0,1\}$$

Ai fini del nostro corso, questa funzione è costruita su un certo dataset  $X, Y$  con  $X$  items e  $Y$  features.

## Classificatori esterni

Sono l'equivalente dell'aver allenato una buona funzione di similarità, definita come

$$\sim : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$$

Che dati due sample mi dice quanto questi due sono simili. Di solito si usa KNN; i dati di un KNN già allenato sono  $\bar{X}, \bar{Y}, \sim, K$ .

Data un'istanza sconosciuta, possiamo sfruttare questa funzione di similarità per decidere la classificazione dell'istanza nel seguente modo:

$$\text{predict}(\hat{X}) = \text{take closest samples in } \bar{X}, \bar{Y} \text{ to } \hat{X}; \text{return } \sum \hat{Y} \geq \frac{k}{2}$$

Ovvero, abbiamo sommato sulla colonna gli 0,1; se ho più 1 otterrò un valore superiore alla metà di  $k$ , quindi ritornerò 1, altrimenti 0.

L'idea è che prendo i  $k$  samples più vicini; ognuno di questi "vota" sulla classe di cui fa parte e vince la maggioranza.

Questo semplificatore è molto semplice, ma data una buona funzione di similarità in realtà funziona già benino :)

Vantaggi	Svantaggi
<ul style="list-style-type: none"><li>Molto efficiente :)</li><li>Facile da implementare :)</li><li>Parametrizzabile su <math>k</math></li><li>Facilmente "explainable": mi basta tornare i <math>k</math> samples simili. Piace molto ai medici.</li><li>Sono sufficienti pochi samples</li></ul>	<ul style="list-style-type: none"><li>Devo avere una buona funzione di similarità; praticamente ho spostato il problema...</li></ul>

## ALBERI E FORESTE

Si tratta di un algoritmo ricorsivo.

$$DT\_fit(X, Y, loss)$$

Prende il dataset con  $m$  features e sceglie su quale feature e su quale valore della feature fare lo split, separando in due il dataset. (per semplicità, consideriamo tutte features binarie)

Preso il dataset, scelgo un valore, splitto in due su quello e tolgo anche la colonna di quel valore – dato che in ogni sottodataset avrò che quel valore è costante.

Dopo di che, riapplico la stessa procedura a tutti i nodi attenuti, finché tutti i nodi nel sottocampione sono della stessa classe, aka  $Y = 1 \vee Y = 0$ .

La cosa importante qui è la loss. La funzione di loss, in pratica, è

$$\text{loss} : \underbrace{\mathbb{R}^{m \times n}}_X \times \underbrace{\{0,1\}^m}_Y \times \underbrace{\{1 \dots m\}}_{\text{feature } i} \rightarrow \mathbb{R}$$

Ogni ramo non etichettato può avere lunghezza diversa; per esempio, potrei avere che in un path ottengo tutto “uniforme” sulla classe che mi interessa classificare prima che su un altro.

L’idea è che applico la loss su ognuna delle colonne; quella che risulta più alta è quella che uso per fare lo split.

L’albero è alto al più quanto il numero di feature.

In generale preferiamo alberi più piccoli:

- Fanno meno test: descrivono in maniera più sintetica il dataset
- Generalizzano (= quanto il classificatore riesce a classificare un sample mai visto) meglio: per esempio, se il nuovo sample ha degli attributi mancanti, potrei riuscire a classificarlo lo stesso.

Trovare l’albero minimo, tuttavia, è NPC. Vanno benissimo le altre tecniche, come l’Information Gain. Se le features sono buone, la complessità è nel numero di features; altrimenti è  $(\text{colonne} + \text{valore})^n$

### Information gain

Un esempio di loss function è costruito sull’entropia: calcolo il drop di entropia che ottengo splittando sull’attributo scelto – ovvero, sarà 0 sulle classi che poi mi risulteranno uniformi (sulla classe che mi interessa) dopo lo split.

$$E(X, y) = \sum_{\substack{\text{sample } i \\ \text{etichette}}} p \log \frac{1}{p} + (1-p) \log \frac{1}{(1-p)}$$

→ se non è binaria

Guardo quanto si abbassa l’entropia facendo lo split su quell’attributo

$$IG_i(x, y) = E(x, y) - \frac{\sum x_i \cdot E(x_{i=1}, y_{i=1}) + (1 - \sum x_i) E(x_{i=0}, y_{i=0})}{|x|}$$

← e' in proporzione!

Quindi selezioni la feature come

$$i = \underset{1 \leq i \leq m}{\operatorname{argmax}} (IG)$$

Quindi minimizziamo l'entropia, massimizziamo l'information gain.

### Considerazioni

- Posso cambiare la loss e mettere quella che preferisco
- Definisco  $X, Y$  consistente se  $\forall i, j : X[i] = X[j] \wedge Y[i] \neq Y[j]$ ; ovvero, non posso avere due righe uguali con etichette diverse: sample uguali per etichette uguali. Con questa assunzione si arriva sempre ad avere foglie omogenee.

Oltre all'information gain possiamo usare anche altri indici, come Gini (impurità del dataset) o Chi quadro.

È molto explainable: mi basta dare il path della classificazione nell'albero.

### Overfitting

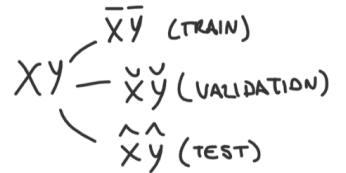
È un problema grosso, in quanto rende tutto da buttare; consiste nel fatto che la classificazione funziona molto bene sui dati di training ma molto male sui dati reali di test.

Ce ne possiamo accorgere dividendo  $X, Y$  in  $\bar{X}, \bar{Y}$  (train) e  $\hat{X}, \hat{Y}$  (test). Poi usiamo  $f_{\bar{X}\bar{Y}}$  su  $\hat{X}$ , e verifichiamo che il risultato corrisponda a  $\hat{Y}$ .

$$\text{ACCURATEZZA} = \frac{\sum |f_{\bar{X}\bar{Y}}(\hat{x}) - \hat{y}|}{|\hat{X}|}$$

Poiché i decision tree fatti con l'entropia sono molto veloci da allenare, posso dividere il dataset in tre:

Abbiamo già visto train e test; validation serve in quanto a ogni fase dell'allenamento ci dà un hint su come si sta comportando il dataset; è diversa dal test, in quanto la validazione influenza la creazione dell'albero.



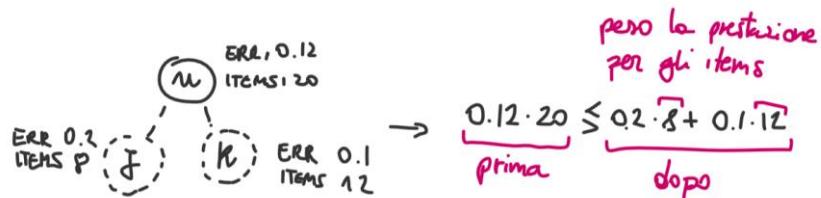
L'idea è che se creiamo un albero troppo grande rischiamo di fare overfitting. Quindi, per ogni foglia, decido se andare avanti a dividere o meno in base a come si comporta l'albero sui dati di validation; se si comporta peggio, banalmente, mi fermo alla foglia corrente.

! Questo non mi assicura che io non avrei migliorato le prestazioni se fossi andato avanti, ad esempio, due foglie!  
Approssima: rischio addirittura di fare underfitting, perché nulla mi assicura che io non stia perdendo test che invece servirebbero.

Guardo, quindi, due cose:

- Prendiamo il nodo corrente  $u$ . Quanti elementi del dataset di validation arrivano al mio nodo? Se il mio nodo fosse foglia e "votasse", quanti elementi del dataset di validation sarebbero classificati correttamente?
- Ora splitto il nodo  $u$  in  $j$  e  $k$ . Se mi fermassi in  $j$  e  $k$ , quanti items passerebbero da questi nodi, e quanto classificherei correttamente?

Faccio il confronto fra i due risultati e guardo se ne vale la pena. Ad esempio:

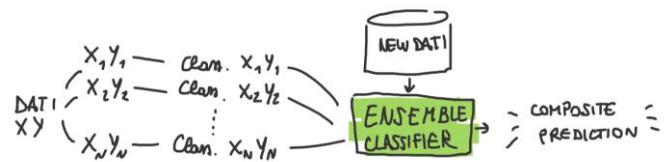


### Random forest

Facciamo tanti alberi. :)

Trasformiamo il dataset in tanti dataset, alleniamo quindi tanti alberi, e poi faccio votare i tanti alberi: arrivato l'elemento da classificare, lo passo per tutti i tree e decido chi vince in base a un certo criterio.

L'idea è che anziché avere 1 classificatore forte ho una serie di classificatori deboli, ma trovo una funzione di "votazione" per gli alberi che valorizzi ciascun albero in quello in cui è capace.



Vediamo i meccanismi di voto.

### 1. Majority

$$\sum_{i=1}^N \frac{T_{x_i, y_i}(x)}{w} \geq \frac{w}{2} \text{ THEN TRUE, ELSE FALSE}$$

### 2. Weighted majority

(il peso delle parole dipende da chi le dice cit. Sala cit. Fabri Fibra)

Supponiamo che ci siamo tenuti una validazione, e misuriamo l'accuratezza sulla validazione. Uso questa validazione per creare un peso, che è tanto alto quanto più l'albero è accurato.

$$w_i = 1 - \frac{\sum |T_{x_i, y_i}(x) - y|}{|x|}$$

A questo punto sommo i pesi di chi classifica 1 e di chi classifica 0, e vedo chi vince :)

$$\sum_{i, T_{x_i, y_i}(x)=1} w_i \geq \sum_{i, T_{x_i, y_i}(x)=0} w_i \text{ THEN TRUE ELSE FALSE}$$

### 3. Track record

Nasce dall'osservazione che un classificatore potrebbe essere bravissimo a predire la classe più numerosa A, ma far cagare su altre classi B e C meno numerose.

		PREDICTE →	
		REALI ↓	
REALI	0	0	FP
	1	FN	TP

Si usa uno strumento chiamato matrice di confusione: sempre sul dataset di validazione, facciamo una stima di espertaggine su ogni classe. Per ogni albero costruiamo una matrice dove sulle colonne ho le classi predette, e sulle righe le classi reali: sulla diagonale avrò le previsioni corrette, e sul resto le previsioni errate.

	0	1
0	.8	.3
1	.2	.7

Normalizzando le colonne ottengo un dato che si può leggere come "ogni volta che il classificatore dice 0, è giusto all'80% e sbagliato al 20%".

A questo punto, i miei pesi saranno dei vettori  $1 \times 2$  con i pesi della predizione se è azzeccata o se è canata:

$$w_{c_j, \bar{c}_j}^i = CM_{norm}[c_j, \bar{c}_j]$$

E quindi decido se vince il true o il false vedendo quale delle due mi massimizza la somma dei pesi.

$$\operatorname{argmax}_{c_j} \left( \sum_{i=1}^n w_{c_j}^i \right)$$

## Post pruning

Un'alternativa a fare il pruning "mano mano", usando il dataset di validazione, è farlo ad abero concluso. In questo caso, per ciascun nodo da splittare calcolo

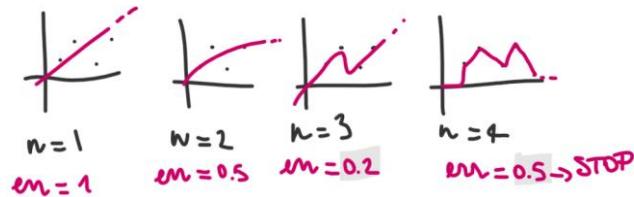
$$\text{IF } \text{err}(v) \cdot \#(v) \leq \text{err}(v_T) \cdot \#(v_T) + \text{err}(v_F) \cdot \#v_F$$

THEN PRUNE ( $\rightarrow = v_T \text{ e } v_F \text{ diventano foglie}$ )

Quindi, prima faccio gli split e poi cancello i rami che non vanno bene. Sulla radice "tiro" la cosa più probabile, aka che è più presente nei sample; qualunque cosa che ci azzecca meglio di questo (molto, probabilmente) viene accettata.

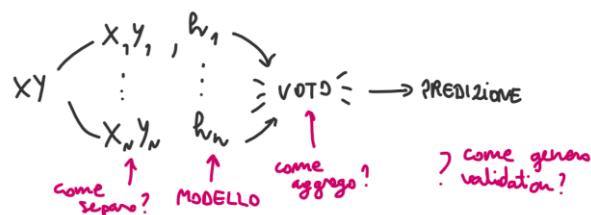
## Hyperparameter optimization

L'idea è che creo una famiglia di modelli a complessità crescente. Ad esempio, sempre basandomi sul dataset di training, creando un neural network a 1 neurone, poi passo a due neuroni, etc. Vado avanti finché il passo successivo mi dà un risultato peggiore di quello precedente sul dataset di validazione.



Eventualmente, anziché fermarmi al primo peggioramento, nulla mi vieta di andare avanti e vedere come va; questo ha senso perché non ho un minimo garantito, quindi potrei avere un peggioramento seguito da un miglioramento significativo.

Nel contesto dell'ensemble classification, come abbiamo già visto, divido il dataset. Potrei anche avere tuple duplicate, con l'idea che più volte compare una tupla, più questa avrà peso sul risultato finale. Ad esempio, potrei inserire ciascun sample proveniente da un povero stupido medico una volta, e invece inserire i samples che vengono da ricerca comprovata 10 volte per dare loro un peso maggiore.



Quest'idea dell'ensemble funziona molto bene sui decision tree, perché tipicamente si usa su weak classifiers.

## Bagging – bootstrap aggregating

Funziona sorprendentemente bene.

```
X, Y = []
FOR 0 ≤ i ≤ |X|
    i ← sample(0...|X|)
    (X, Y).append((X[i], Y[i]))
X_v, Y_v = (X, Y) \ (X[i], Y[i]) quelli che restano fuori li
                                            metto in VALIDATION
```

Ogni dataset risultante è grande quanto l'originale, ma estraendo (con reinserimento)  $n$  volte succederà che avrò dei samples "lasciati fuori" e dei samples inseriti più volte. Quindi, raddoppiamo il peso di alcuni samples e togliamo peso agli altri.

Tutti i samples che non sono mai stati inseriti, quindi, possono essere usati come dataset di validazione.

Ma con questo metodo, quanti samples lascio fuori in media?

• una estrazione ha probabilità:

$$\frac{1}{N} \text{ di prendere un certo sample}$$

$$1 - \frac{1}{N} \text{ di non prenderlo}$$

• faccio  $N$  estrazioni, quindi  $\left(1 - \frac{1}{N}\right)^N$  di non renderlo mai

• limite:

$$\lim_{N \rightarrow \infty} \left(1 - \frac{1}{N}\right)^N$$

$$\downarrow z = -w \quad = \lim_{z \rightarrow -\infty} \left(1 + \frac{1}{z}\right)^{-z}$$

$$= \lim_{z \rightarrow -\infty} \left(\left(1 + \frac{1}{z}\right)^z\right)^{-1} = e^{-1} = \frac{1}{e}$$

limite fondamentale

$e^{-1}$

$\sim 30\% \text{ di "perderci" il sample}$

↓  
30% delle tuple *sia in validation*

Questo limite viene raggiunto molto velocemente: già a 100 samples succede che siamo molto vicini.

### Bagging sulle features

Se ho tante features e poche colonne – come potrebbe succedere ad esempio nel caso del DNA – posso fare bagging sulle colonne: duplico i sample tante volte, e poi applico il bagging sulle colonne in modo tale che ogni copia del samples avrà colonne duplicate/escluse.

### Random forest

È un ensemble di alberi dove ciascun albero è allenato su un dataset fatto con bagging; facciamo crescere l'albero finché la precisione su validation scende.

Il numero di alberi nell'ensemble è un parametro.

## APPLICAZIONE DELLE RANDOM FOREST AD ALTRI CONTESTI

### Clustering

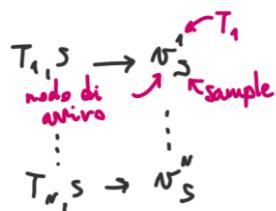
Quando abbiamo bisogno di una misura di similarità ma non sappiamo come generarla, in quanto non ne abbiamo a disposizione di nuove, le random forest sono l'alleato più inaspettato dell'inquisizione spagnola.

Versione easy: esempio con NLP

Prendiamo un certo task di NLP: ho un certo insieme di tweet categorizzati in positivi e negativi. Abbiamo anche allenato un'ottima random forest sulle frasi di questi tweet. Partendo da questa random forest, vogliamo generare una misura di distanza fra due sentences:

$$dist_{RF}(s, z)$$

Qual è quindi la rappresentazione interna delle random forests, che definiamo footprint:



(dal nodo di arrivo posso risalire univocamente al cammino percorso, dato che ho un albero senza cicli e cose)

Con questa rappresentazione, il massimo di similarità è se per ogni albero della foresta i due samples fanno lo stesso cammino.

Quindi, possiamo definire la similarità come

$$dist(s, z) = \frac{|\{s, v_s^j = v_z^j\}|}{\# \text{di alberi} \rightarrow w}$$

Posso applicare questa cosa a 1 solo albero, ma in questo modo avrei solo 0 e 1 come distanza (uguali o diversi). Quindi, in realtà, ha molto più senso usare il diametro dell'albero (distanza più grande all'interno dell'albero) e la distanza dei nodi, detta anche Bregman distance, che consiste nel misurare i salti necessari per salire dal primo nodo fino al last common ancestor e poi riscendere al secondo nodo.

In questo modo, come ci aspettiamo, otteniamo 1 se sono agli antipodi e 0 se sono lo stesso nodo. Partendo da questo posso applicare sulle foreste:

$$\frac{\sum_{i=1}^w BD(T_i, s, z)}{w} \quad \begin{aligned} & \left( \begin{array}{l} = \text{MEDIA DELLA BD} \\ = \text{media del \# di domande} \\ \text{che separano due sample} \end{array} \right) \end{aligned}$$

### Fake task

Ora supponiamo di voler trovare la misura di similarità su un dataset X, che non ha Y e non ha nessuna random forest. Nel caso precedente, in realtà, abbiamo visto che le labels non servivano a niente: ho solo usato l'albero e la sua forma, ma le label sono inutili! Quindi quello che posso fare è costruire un finto task di classificazione per ottenere l'albero, e poi usare l'albero per misurare la distanza.

In pratica: prendo il dataset, creo un dataset farlocco sulla classificazione, alleno random forest sul dataset fittizio e poi uso random forest per fare la distanza.

$$X \rightarrow \bar{X}, \bar{Y} \rightarrow RF_{\bar{X}\bar{Y}} \rightarrow \text{dist}_{RF_{\bar{X}\bar{Y}}}$$

Questa cosa si usa spesso per le time series, in modo da potervi poi applicare Nearest Neighbor 1.

Come costruisco  $\bar{X}$  e  $\bar{Y}$ ?

*Caso binario*

$X \in \{0,1\}^{n \times m}$ : il mio dataset è un insieme di bit.

La probabilità di ciascun elemento (supporto) sarà

$$p_j = \frac{\sum_{i=1}^n X_{i,j}}{n}$$

Dato questo, genero un secondo dataset  $\tilde{X} \in \{0,1\}^{n \times m}$  nel seguente modo:

$$\begin{aligned} \tilde{X} &= [] \\ \text{FOR } 1 \leq i \leq n \\ \text{FOR } 1 \leq j \leq m \\ \tilde{x}_{i,j} &= \text{sample}([1,0], p_j) \end{aligned}$$

quindi 'ero'  
 $\tilde{p}_j = \frac{\sum \tilde{x}_{i,j}}{n}$

Quindi, la differenza fra  $X$  e  $\tilde{X}$  sarà che

$$\begin{array}{ll} X & \tilde{X} \\ \text{distribuzione } \in & \text{stessa distribuzione di } X \\ \text{eventuali legami di} & \text{ma è random, no legami} \\ \text{casualità} & \end{array}$$

Il nostro task fake, quindi, sarà imparare a distinguere fra  $X$  e  $\tilde{X}$ . Mi basta quindi creare  $\bar{X}\bar{Y}$  come

$$\bar{X}\bar{Y} = \begin{array}{|c|c|} \hline \bar{X} & \bar{Y} \\ \hline \bar{X} & 1 \\ \hline \bar{X} & 0 \\ \hline \end{array}$$

vero  
simulato

*Features categoriche*

Nel caso di feature categoriche, anziché avere  $[0,1]$  ho una distribuzione dei valori, oppure creo una colonna per ogni valore possibile e le metto a 1:

	color	blue	tor	odor	color
	blue	red	white	blue	
1	1	0	0	1	
2	0	1	0	2	
3	0	0	1	3	

*Features numeriche*

Per feature numeriche, mi basta calcolare media e deviazione standard di ciascuna colonna e tirare su una distribuzione normale  $N(\mu, \sigma)$  generata con questi dati. O, in alternativa, discretizzo.

## TEXT MINING

È il classico esempio di fake task che si usa per quando ho poche righe e mooooalte colonne.

ChatGPT

Non è altro che una fake task!

Se prendiamo le traduzioni, che erano lo scopo originale di ChatGPT, abbiamo un grafico del tipo



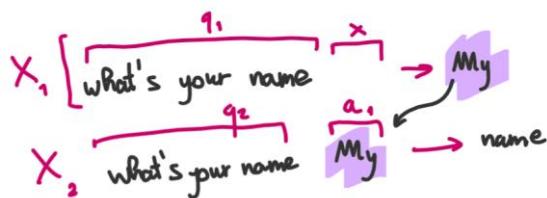
Usando invece le architetture di tipo "sequence to sequence", diventa



ChatGPT è stato allenando dandogli due testi come ingresso:

- Testo effettivamente da tradurre
- Testo già tradotto fino a quel momento

E come output si chiede la parola successiva.



Non è una buona idea rappresentare questa cosa come una mega matrice sparsa dove metto a 1 le parole che compaiono.

Il modo in cui funziona è che osservo che (es.) i pronomi personali come my, your, etc si trovano sempre nello stesso contesto. Quindi non definisco la similarità solo sui sinonimi: due parole sono simili se compaiono in contesti simili.

ChatGPT salva un vettore di grandezza  $n$  per ogni parola; la distanza del coseno fra questi vettori è la distanza. Il motivo per cui si usa la differenza del coseno è che essa non tiene conto della lunghezza relativa dei due vettori, e questo è molto importante in quanto la lunghezza dei vettori indica solo la frequenza di una parola.

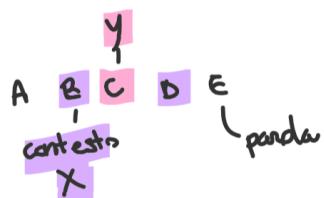
Per fare un paragone, misurare la distanza del coseno è come se io stessi guardando le stelle e definessi come distanza quanto mi devo "girare" per vedere due stelle. (es. massima distanza:  $180^\circ$ )



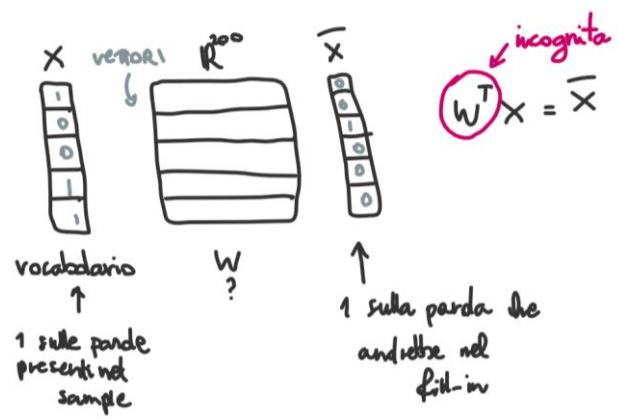
Allenamento

### 1. Softmax word bagging

Prendo piccoli contesti intorno alle parole e quelli definiscono il mio sample. I sample del contesto NON sono considerati in ordine.



Quello che mi interessa è imparare i pesi della matrice con i pesi delle parole: però alleno la cosa su un fake task, che consiste nel prendere delle frasi e far fare il "fill the gaps". L'obiettivo è minimizzare l'entropia ottenuta in questo task.

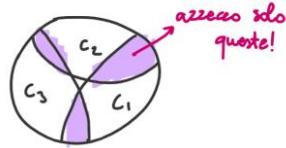


*Skip gram*

È il contrario: come input ho la parola, e cerco di indovinare il contesto.

## BOOSTING

Supponiamo di avere dei weak classifier in una classificazione binaria su  $[-1,1]$ . Ciascun classifier classifica bene una parte di piano:



Le uniche parti che vengono classificate correttamente sono quelle in overlap, perché hanno la maggioranza dei voti corretti. Potrei continuare ad aggiungere classifiers, però meh...

In questo contesto, la nostra classificazione sarà

$$\text{classificazione} = \text{sign}[\alpha^1 h^1(X) + \cdots + \alpha^N h^1(X)]$$

Dove alpha sono i pesi che do a ciascun weak classifier, e h sono i classifier veri e propri.

Ora vediamo una magica tecnica che ci permette di avere un decadimento esponenziale dell'errore.

!! Sceglio un singolo weak classifier. A ogni passo lo cambieremo con un weak classifier che è calcolato dando peso diverso agli item del dataset, dando più peso a quelli

Al passo 1, cerchiamo il test che minimizza l'errore; "taglio" in tutti i nodi e prendo il test che mi dà il minimo di valori errati.

$$\sum w^i(x) = 1$$

Definiamo l'errore come

$$\varepsilon^i = \frac{\sum_{\text{wrong}}}{N} = \frac{\sum_{\text{wrong}} 1}{N}$$

Ora voglio spingere al massimo questo classificatore. Se al passo successivo facessi la stessa cosa, ovviamente avendo lo stesso dataset otterrei gli stessi pesi del passo #1... quindi do peso maggiore alle cose che ho canato al passo 1! Riarrango i pesi in modo che quelli che ho azzeccato pesano meno, e gli altri pesano di più. Mantengo costante che la somma dei pesi deve essere 1.

$$\sum w^i(x) = 1$$

Si sono accorti, in maniera rigorosa e dimostrabile, che

1.

$$\alpha^t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon(t)}{\varepsilon(t)} \right)$$

2. I pesi sono sempre

$$w_i^{t+1} = \frac{w_i^t}{z} \cdot e^{-\alpha^t h^t(x) y(x)}$$

↑ se classificato correttamente  
-1 else  
NORMALIZZAZIONE

Quindi, i pesi al passo successivo saranno sempre

$$w_i^{t+1} = \frac{w_i^t}{z} \cdot \begin{cases} \sqrt{\frac{\epsilon(t)}{1-\epsilon(t)}} & \text{se vero} \\ \sqrt{\frac{1-\epsilon(t)}{\epsilon(t)}} & \text{se falso} \end{cases}$$

Ora: se sostituisco questa cosa dentro la formula dei pesi a  $t + 1$  e svolgo, ottengo:

$$\begin{aligned} z &= \sqrt{\frac{\epsilon(t)}{1-\epsilon(t)}} \sum w_i^t + \sqrt{\frac{1-\epsilon(t)}{\epsilon(t)}} \cdot \sum w_i^t \quad \text{WRONG} \\ r &= \sqrt{\frac{\epsilon(t)}{1-\epsilon(t)}} (1-\epsilon(t)) + \sqrt{\frac{1-\epsilon(t)}{\epsilon(t)}} \cdot \epsilon(t) \\ &= \sqrt{\frac{\epsilon(t)(1-\epsilon(t))}{1-\epsilon(t)}} + \sqrt{\frac{(1-\epsilon(t))\epsilon(t)}{\epsilon(t)}} \\ &= \sqrt{\frac{\epsilon(t)(1-\epsilon(t)^2)}{1-\epsilon(t)}} \end{aligned}$$

16 novembre 2023

(recupera)

$$\begin{aligned} N &\leftarrow |x| \\ t &\leftarrow 1 \\ w_x^1 &\leftarrow 1/h \\ \text{REPEAT} \\ h^t &\leftarrow \operatorname{argmax}_{x_i \in X} \frac{R_i(x) - y_i w_i}{z} \\ \epsilon^t &\leftarrow \sum_{x_i \in X} w_i^t \quad \text{Se due sample sono} \\ \alpha^t &\leftarrow \frac{1}{2} \ln \frac{1-\epsilon^t}{\epsilon^t} \quad \text{classificati correttamente} \\ z &= \sum w_i \left[ w_i^{t+1} \leftarrow \frac{w_i^t}{z} \cdot e^{-\alpha^t \cdot h^t(x_i)} \right] \forall x_i \end{aligned}$$

UNTIL  $|f(x_i)| = 1$ ,

$$z = \frac{1}{2} \sqrt{\frac{\varepsilon^t}{1-\varepsilon^t}} \cdot \sum_{\substack{x_i \\ \text{TRUE}}} w_i^t + \frac{1}{2} \sqrt{\frac{1-\varepsilon^t}{\varepsilon^t}} \cdot \sum_{\substack{x_i \\ \text{FALSE}}} w_i^t$$

*e' la def di  $\varepsilon^t$*

$$= \frac{1}{2} \sqrt{\frac{\varepsilon^t(1-\varepsilon^t)}{1-\varepsilon^t}} + \frac{1}{2} \sqrt{\frac{1-\varepsilon^t\varepsilon^{t+2}}{\varepsilon^t}} = z \sqrt{\varepsilon(1-\varepsilon^2)} ???$$

*POTREI CALCOLARE I PESI DIRETTAMENTE SAPENDO SOLO  $\varepsilon$  e I PESI SBAGLIATI*

$$w_i^{t+1} = \frac{1}{2} \frac{w_i^t}{\varepsilon^t} \text{ FALSE}$$

*Spingendoci oltre non mi serve nemmeno  $\varepsilon$*

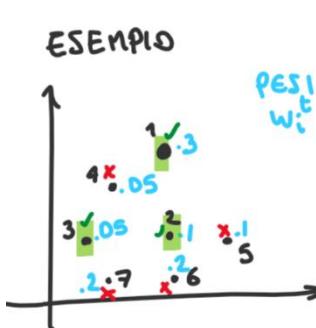
$$\sum w_i^{t+1} = \frac{1}{2(1-\varepsilon)} \sum_T w_i^t + \frac{1}{2\varepsilon^t} \sum_F w_i^t$$

*$\sum_T w_i^{t+1}$  e  $\sum_F w_i^{t+1}$  fanno entrambi  $1/2$*

$$w_i^{t+1} = \frac{w_i^t}{w_T} \cdot \frac{1}{2}$$

*ciascuno  $w_i^t$  e  $w_T$*

*$\sum$  di quelli classificati CORRETTAMENTE*



$$w_i^{t+1} = \frac{w_i^t}{w_1 + w_2 + w_3} \cdot \frac{1}{2}$$

es.  $w_1 = \frac{0.3}{0.3 + 0.5 + 0.2} \cdot \frac{1}{2}$

$$= \frac{0.3}{0.95} \cdot \frac{1}{2}$$

In questo modo, i pesi di quelli giusti decrescono e i pesi di quelli sbagliati diminuiscono.

Questa è la tecnica meno sensibile all'overfitting.

## SEQUENCE TREES

Prendiamo un dataset del tipo:

$$\begin{aligned} \text{seq} &= [(vt_1, \text{label}_1, \text{value}_1) \dots (vt_n, \text{label}_n, \text{value}_n)] \\ \text{dove } vt_1 &\leq \dots \leq vt_n \\ X &= \{\text{seq}\} \\ Y &= \{0,1\} \end{aligned}$$

Immagino di muovermi sul dataset come se fossi un automa a stati finiti che si muove sul suo nastro :)

$\exists (l, d) = \text{yes} \Leftrightarrow \text{from the current point, I observe an event } l \text{ within } d \text{ units of time.}$

1. Immaginiamo che io sia arrivata a un certo punto nella lista; vai al primo punto che soddisfa la condizione e ignora il resto, else resta fermo dove è.
2.  $v \leq q?$   
Faccio un test sulla terza componente.

### Sequence tree

Con questi test posso generare un albero con le seguenti caratteristiche:

$$\begin{array}{c} \text{arco} \\ \text{NO} \quad \gamma \quad \text{arco} \\ \text{YES} \end{array} \quad Q: V \rightarrow \mathbb{R} \cup (\mathcal{L} \times \mathbb{R}) \\ T = (V, E_a, E_v, Q, C) \quad C: V \rightarrow \{0,1\}$$

Requisiti strutturali:

1. Ho al più un arco false e un arco true:

$$\forall v \in V, |\{(u, v) \in E_a\}| \leq 1$$

$$\wedge |\{(u, v) \in E_v\}| \leq 1$$

2. Posso fare un test sul valore solo dopo aver chiesto qualcosa sulla label:

$$\text{type}(v) = \text{value} \Leftrightarrow Q(v) \in \mathbb{R}$$

$$\begin{aligned} \forall v, \text{type}(v) = \text{value}, \text{ then} \\ \left\{ \begin{array}{l} \exists v': \text{type}(v') = \exists^{\text{evento}} \wedge (v', v) \in E_T \\ (= \text{faccio un test di valore dopo aver trascorso un evento}) \\ \exists v': \text{type}(v') = \text{value} \wedge (v', v) \in (E_f \cup E_v) \end{array} \right. \end{aligned}$$

### Predizioni

Le predizioni saranno:

```

PREDICT ( $T$ , seq, start=0, value=None)
    v = ROOT ( $T$ )
    IF leaf (v) THEN RETURN (v)
    IF type (v) =  $\exists$ 
        ( $L, d$ )  $\leftarrow Q(v)$ 
         $J \leftarrow \min J$ 
        SEQ [ $J$ ]. vt - start  $\leq d$ 
        SEQ [ $J$ ]. label =  $L$ 
        IF  $J = \text{None}$ 
            PREDICT ( $T_L$ , seq, start, value)
        ELSE
            PREDICT ( $T_T$ , seq [ $J+1, :]$ , SEQ [ $J$ ]. vt, SEQ [ $J$ ]. value)
        IF TYPE (v) = value  $\leftarrow$  per il vincolo costruttivo non è mai null
         $z \leftarrow Q(v)$ 
        IF VALUE  $\leq z$ 
            PREDICT ( $T_T$ , seq, start, value)
        ELSE
            PREDICT ( $T_F$ , seq, start, value)
    
```

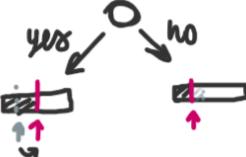
21/11/2023

Per vincolo strutturale, un ramo value può essere solo:

- Figlio di un ramo pair sul suo ramo yes
- Figlio di un altro ramo value.

L'obiettivo è classificare una sequenza di eventi.

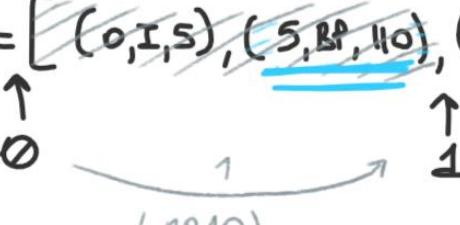
Abbiamo due tipi di domanda, rappresentata da un diverso nodo:

<b>TE (<math>l, d</math>)</b>  <b>Temporary event</b>	<b>Temporary event TE (<math>l, d</math>)</b> : riesci a muoverti nella sequenza che non hai ancora consumato di un tempo minore a $d$ (durata) trovando un evento che ha la $l$ (label) ricercata? Quando è sì, ci muoviamo al punto più vicino che soddisfa quella condizione.	
<b>VE (<math>v</math>)</b>  <b>Value</b>	<b>Value event VE (<math>v</math>)</b> : nodo "classico" degli alberi di decisione, ovvero ho solo una threshold ( $v$ ), che è un reale, e divido fra $\leq$ (yes) e $>$ (no).	
 <b>CLASSE (risposta)</b>	<b>Classe</b>	

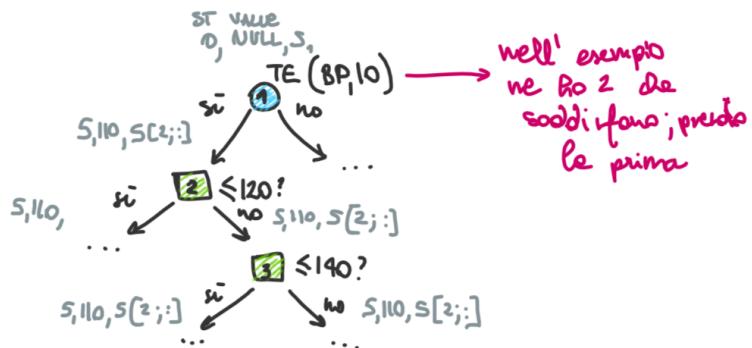
Esempi

Prendiamo come esempio la sequenza:

$$S = [(0, I, 5), (5, BP, 110), (8, I, 12), (9, BP, 130) \dots]$$



Un esempio di atomic test over sequences trees può essere:



Assumiamo solo sequenze con le seguenti caratteristiche:

- A lunghezza variabile
- Per semplicità, tutte con *starting time* = 0
- Solo valori *float*.

Implementiamo quindi un oggetto *sequence tree* con due metodi fondamentali *fit* e *predict*, dove

$$\text{FIT}(X, Y)$$

insieme di sequenze

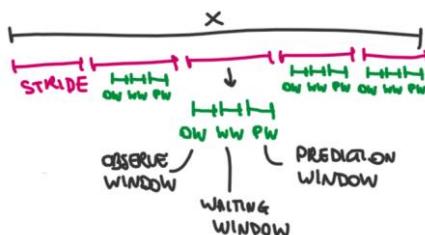
diametrazione delle sequenze

$$X = \left( \begin{array}{c} [(), \dots ()] \\ [(), (), (), ()] \\ \vdots \\ [(), ()] \end{array} \right)$$

$$Y = \begin{array}{c} X \\ \circ \\ | \\ \vdots \\ 0 \end{array}$$

Questi oggetti si usano molto in anomaly detection, per esempio sulle caldaie.

Ogni sequenza  $x \in X$  potrebbe essere una casa, oppure potrei sezionare un'unica casa per time zones come



Se faccio scelgo una stride tale per cui non ho sovrapposizioni ho solo rilevamenti di anomalie sulle ultime  $PW$  ore. Quindi, prendo la time series e faccio partire una scaletta di finestre shiftate e sovrapposte per rilevare su tutto.

Come decido la dimensione delle finestre? BHO

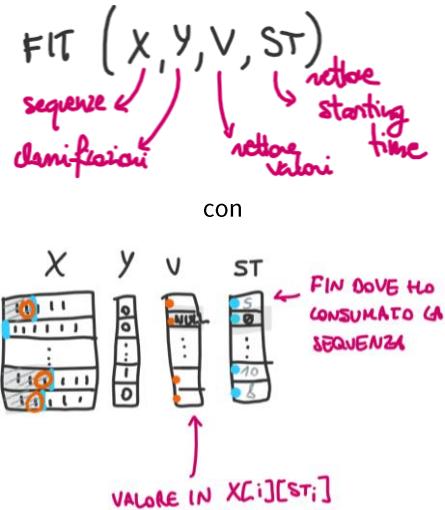
$WW \rightarrow$

- too long: l'anomalia succede prima della PW
- too short: non c'è abbastanza per prevedere perché non cattura tutto il pattern
- OW → too short: non ha abbastanza dati

$PW \rightarrow$

- too long: becco un'anomalia che non causa variazioni nelle WW

## Algoritmo di FIT



1. Genero tutte le sequenze: devo scegliere una coppia label-durata

$$\text{DL-PAIRS} \leftarrow \{(d, l) : \exists i \in X, \exists 0 \leq j \leq |X|, x_i.\text{label} = l \text{ t.c. } x_j.\text{vt} - ST_i = d\}$$

↳ stessa label e distanza d

Qual è la coppia migliore che mi garantisce l'abbassamento dell'IG? Potrei fare il prodotto cartesiano di *label x duration*, però è una cosa un po' sprecona; è una soluzione bovina cit.

Piuttosto, uso massimizzo una funzione

2. Verifico la coppia migliore su cui spezzare:

$$\text{SPLIT-PAIRS} \leftarrow \underset{\text{argmax}}{\{(d, l) \in \text{DL-PAIRS} \text{ t.c. } \text{IG}(Y, y_T^{(d, l)}, y_F^{(d, l)})\}}$$

con

- $y_T^{(d, l)} = \{i \text{ t.c. } \exists 0 \leq j \leq |X|, x_j.\text{vt}_i - ST_i \leq d \wedge x_j.\text{label} = l\}$
- $y_F^{(d, l)} = \{i \text{ t.c. } i \notin y_T^{(d, l)}\}$
- $\text{IG} = \underbrace{E(Y)}_{\text{entropia originale}} - \left( \frac{|y_T|}{|Y|} \cdot E(y/y_T) + \frac{|y_F|}{|Y|} \cdot E(y/y_F) \right)$  entropia dopo lo split

INDEX	CLASS	$y_T$	$y_F$
0	0	✓	
1	1	✓	
2	0	✗	
3	1	✗	
4	1	✓	
5	0	✗	
6	1	✓	
7	1	✗	

soddisfa  
 (d, e)  
 posso  
 proiettare  
 x, v nulla  
 clane

Tanto mi interessa  
 solo le cardinalità

$$\rightarrow IG = E(Y) - E(Y_T + Y_F)$$

$$E(Y) = \frac{5}{8} \log \frac{5}{5} + \frac{3}{8} \log \frac{3}{3}$$

$$E(Y_T + Y_F) = \frac{1}{2} \left( \frac{3}{4} \log \frac{1}{3} + \frac{1}{4} \log 4 \right) + \frac{1}{2} \left( \frac{2}{4} \log \frac{4}{2} + \frac{2}{4} \log \frac{2}{2} \right)$$

3. If value != None, allora posso valutare uno split value: devo decidere se guardare un altro evento o splittare sul valore

$$SPLIT\_PAIR\_IG \leftarrow IG^{(d,e)}(X, Y_T, Y_F^{(d,e)})$$

IF VALUE  $\neq$  None decidere se splittare un altro evento o se splittare sul valore (0/1)

$$SPLIT\_VALUE \leftarrow \underset{v \in V}{\operatorname{argmax}} (IG(Y, Y_T^{\leq v}, Y_F^{\leq v}))$$

- con
- $Y_T^{\leq v} = \{i : v_i \leq v\}$
  - $Y_F^{\leq v} = \{i : i \notin Y_T^{\leq v}\}$

$$SPLIT\_VALUE\_IG = IG(Y, Y_T^{\leq \text{split-value}}, Y_F^{\leq \text{split-value}})$$

Confrontando i due risultati, quindi, decido come splittare:

```

IF split_value_lG > split_pair_lG
    n ← create_value_node (split_value)
    n.child.true = FIT ( $X[y_T^{\leq \text{split-value}}], Y[y_T^{\leq \text{split-value}}], V, ST$ )
    n.child.false = FIT ( $X[y_F^{\leq \text{split-value}}], Y[y_F^{\leq \text{split-value}}], V, ST$ )
    RETURN n
ELSE
    n ← create_pair_node (split_d, split_l)
    n.child.true ← FIT ( $X[y_T^{(d,l)}], Y[y_T^{(d,l)}], V_T, ST_T$ )
    n.child.false ← FIT ( $X[y_F^{(d,l)}], Y[y_F^{(d,l)}], V_F, ST_F$ )
    RETURN n

```

$\exists j \text{ t.c. } X[j].vt - ST \leq d \wedge X[j].label == l$  formula  
 $j = \min(\Phi)$   
 RETURN  $\underbrace{X}_{X_T}[j+1, end], \underbrace{Y}_{Y_T}[x], \underbrace{X[j].vt}_{ST_T}, \underbrace{X[j].value}_{V_T}$

#### 4. Casi di terminazione

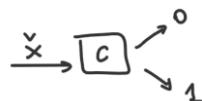
```

IF  $Y = \epsilon 1 \vee Y = \emptyset$  (MONOCLASSE)
    RETURN create_leaf (value(Y))
    sequenza finita non ho riscontro
    IF  $X == [] \wedge \text{value} == \text{None} \vee \text{constant}$  ho riscontro ma non posso dividere
        RETURN create_leaf (majority(Y))

```

## REGRESSIONE LINEARE

Fino ad ora abbiamo visto casi del tipo



Ora invece vogliamo guardare un caso del tipo



Valutiamo il caso supervisionato, ovvero che  $X, Y \in \mathbb{R}^{|x|}$ : cioè ad esempio ho dei potenziali di investimenti, e voglio predire la percentuale di ritorno per decidere se investire o meno. È molto più **difficile**.

### Errore: MSE

Come misura di errore usiamo **mean square error** (MSE). Questo perché potremmo prendere l'errore non quadrato, ma poi si è notato che l'errore converge molto più lentamente.

$$\underset{f \in F}{\operatorname{argmin}} (\text{MSE } (f(x), y))$$

$\hookrightarrow \forall f \in F$ , ovvero ogni possibile rete neurale

### Regressione lineare

È il problema di regressione più semplice. Prendo:

- Sample  $X$  di dimensione  $n \times n$
- Pesi  $w \in \mathbb{R}^n$

$$w^T \times X + b = \hat{y}$$

pesi  $\rightarrow$   
 sample  $\rightarrow$   
 bias  $\rightarrow$   
 valore della regressione

Prendo i pesi che ottimizzano MSE.

Con solo questi dati sono **molto vincolati** perché ho **tanti pesi quante sono le features**.

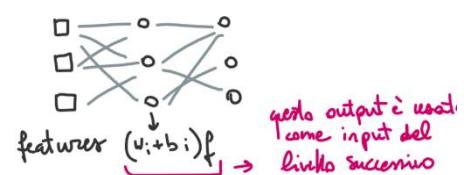
Posso solamente scegliere i pesi/bias migliori per minimizzare MSE: sto prendendo il miglior piano/retta/etc, ma sempre piano/retta/etc sono (perché ho un numero fisso di incognite).

Si può risolvere analiticamente, sfruttando un paio di trick:



Prendo una certa funzione da applicare al risultato:  $f(w^T X + b) = \hat{Y}$   
 Anziché dare direttamente il valore di  $w^T X + b$ , diamo una **funzione applicata** a questo; per esempio, una funzione molto comune è RELU ( $0 \text{ se } x \leq 0, v \text{ se } x > 0$ )

!!! La  $f$  applicata serve perché **altrimenti i livelli successivi della rete diventano negligible**, perché essendo che ogni strato è al derivata del precedente (e la derivata di una funzione è la funzione derivata moltiplicata per la funzione) mi trovo che la prima funzione viene moltiplicata per  $n$  volte.



## Feature engineering

Anziché avere complessità nella rete, posso spalmare la complessità nelle feature. Cioè: se ho 3 feature posso fare ben poco, perché mi danno solamente 3 pesi. Però nessuno mi vieta di **costruire delle feature artificiali** combinando le feature reali, come ad esempio...

$$\begin{aligned} & x_1, x_2, x_3 \\ & +x_1 \cdot x_2 \quad +x_1^2 \quad +x_1 \cdot x_2^2 \\ & \dots \end{aligned}$$

In questo modo aumento il numero di features, e ho più pesi a disposizione. Dopo di che, alleno e guardo che pesi la fit ha dato alle singole features: i pesi più alti corrisponderanno alle feature più utili.

Questo modello, essendo molto **semplice**, è anche abbastanza **explainable**: se per delle feature  $x_1, x_2$  ho I pesi alti, allora so che l'interazione fra quelle due feature è rilevante.

Alcuni parametri sono:

- **Scale**: inserisco il range dei dati
- **Momentum**: permette di dare una sorta di "kickstart" alla funzione se il gradiente si adagia
- **Learning rate**: sarebbe di quanto scendo ad ogni passo (moltiplicato per il gradiente)

$$\mathcal{E} = \begin{array}{|c|} \hline \text{VECTORE} \\ \hline \text{GRADIENTE} \\ \hline (\text{derivate}) \\ \hline \nabla x \\ \hline \end{array}$$

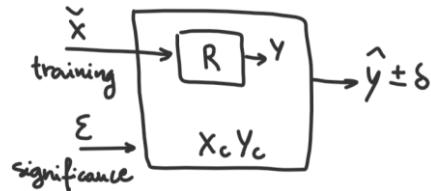
- È una buona abitudine scalare tutti i numeri sul range  $\frac{x - \mu}{\sigma}$

## INDUCTIVE CONFORMATIVE PREDICTION

30/nov/2023

[RECUPERA LEZIONE PREC](#)

Dato un regressore, posso complicare un po' la struttura in questo modo:



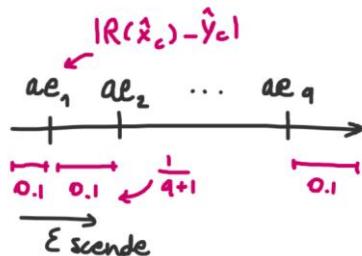
La scatola esterna prende il risultato di  $R$ , lo **confronta col dataset di calibrazione e ritorna un intervallo** tale per cui  $y_{reale} \in y_{predetta} \pm \delta$  con una significance  $\epsilon$ , ovvero

$$\lim_{n \rightarrow +\infty} \frac{|X \in X_n, Y \in (\hat{Y} \pm \delta)|}{n} = \epsilon$$

...ovvero, se prendo 100 sample mai visti prima e li classifico impostando  $\epsilon = 0,1$ , so che il **90% delle predizioni cadrà nell'intervallo** di regressione e il 10% cadrà fuori.

Sto "forzando" un intervallo di accuratezza sulla regressione.

E' molto facile usare questo test di calibrazione: **più voglio essere preciso, più devo avere un  $\delta$  largo**.



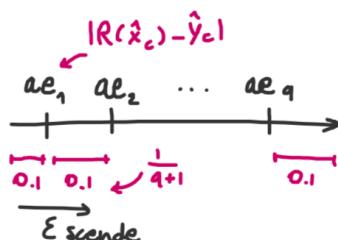
*Exchangeability*

Sotto la nozione di **exchangeability dei sample** (che è più debole di, ad esempio, statistically dependent), il sampling deve essere uguale per tutti gli elementi; per esempio, la macchina usata per le rilevazioni deve essere la stessa calibrazione.

Non conformity function custom

Il dataset di calibrazione deve avere anche la classificazione. A partire da questo dataset posso creare la **non conformity function**, ovvero una funzione che ha **valore tanto alto quanto più alta è la non conformità del sample**.

Ad esempio, se ho 9 sample, posso prendere l'**absolute error** di ciascun sample e ordino i sample per absolute error, dividendo lo spazio in 10 ( $9+1$ ) intervallini. Ciascun intervallino è equiprobabile con  $\frac{1}{q+1}$ , con  $q$  = numero dei sample in  $X_C$ .



**Quando arriva un nuovo sample, classifico  $R(x) = \hat{y}$  dove  $\hat{y}$  sarà il centro del mio intervallo, e il  $\epsilon$  mi determina il  $\delta$ .**

Poiché ho delle classi (=sono in un ambiente discreto), **prenderò tutte le classi che si trovano all'interno del mio intervallo** sulla linea degli absolute error:

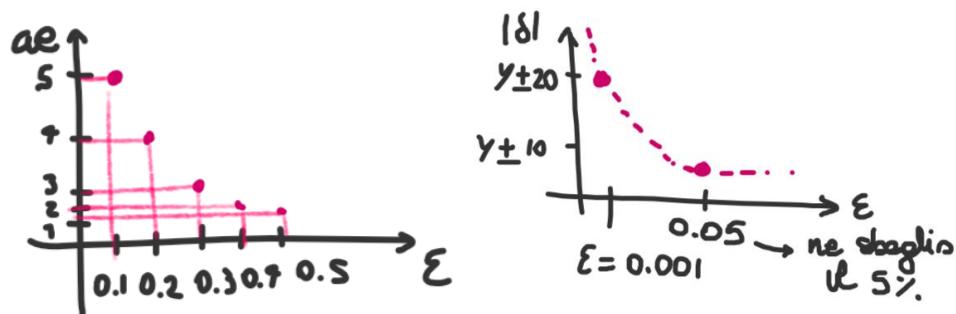
$$\text{Significanza} \\ \alpha = \lfloor (1-\varepsilon)(q+1) \rfloor$$

Ad esempio, se ho il 20% di errore ( $\varepsilon = 0.2$ ) , arrivo fino a  $(1 - \varepsilon)ae_8$  : questo perché significanza 20% == confidenza 80%.

NB: nella realtà, avendo un buon regressore, mi aspetto di ottenere degli intervalli stretti all'inizio con degli intervalli più lunghi alla fine per gli outlier.



Quindi l'efficienza sarà:



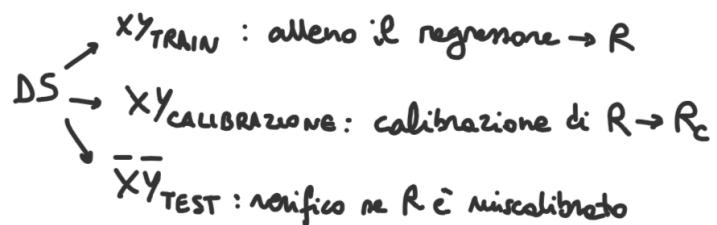
Se il dataset non è miscalibrato, allora dovrei sempre avere una forma di tipo

Efficienza

L'efficienza è l'area sottesa dalla curva; meno la curva si mangia il quadrato, meglio è.

## Testare la miscalibrazione

Quello che invece dobbiamo testare è la miscalibrazione. Prendo il mio dataset originale e lo separo:



Se il dataset non è miscalibrato troverò che

$$\text{for } \varepsilon \in [0, 1], \frac{\bar{y}_T \notin R_c(\bar{x}, \varepsilon)}{|\bar{y}_T|} \approx 1 - \varepsilon$$

Ovvero, circa  $(1 - \varepsilon)\%$  del dataset cade nel bucket sbagliato.

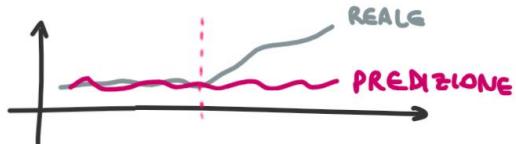
(codice)

Attenzione: come scatola interna (aka funzione di non conformità) potrei usare qualunque cosa: **un risultato diverso da quello sopra non è indicativo di nulla sulla funzione di conformità, ma al contrario dimostra una miscalibrazione del dataset**, e quindi che il dataset non è exchangeable.

## Concept drift detection

Questo può essere anche sintomo di un concetto tipico del machine learning, ovvero il **concept drift**.

Supponiamo che vogliamo prevedere l'andamento delle vendite lungo un asse temporale, per decidere dipendenti/soldi da investire per il prossimo semestre.



Il classificatore potrebbe non predire più bene la realtà da un certo punto in poi, perché **la realtà è cambiata** (es. nuove leggi, cambio di mercato, etc) e **qualcosa che prima era ininfluente ora lo è diventato**.

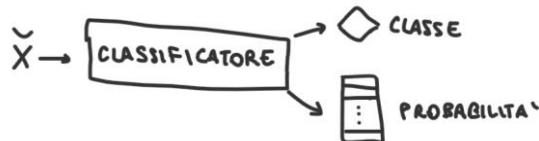
Se mi accorgo di questo fatto, cosa che posso fare con la miscalibration detection, posso riallenerne e quindi aggiornare il modello. 😊

E' utile perché se cade l'assunzione della calibrazione allora possiamo usare quella opposta:

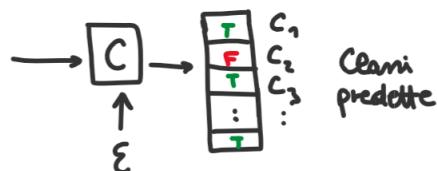
$$\begin{aligned} \text{exchangeability} &\rightarrow 1 - \varepsilon \text{ misclassification} \\ (\text{non valida, quindi}) \\ \neg (1 - \varepsilon \text{ misclassification}) &\rightarrow \text{NOT EXCHANGEABILITY} \end{aligned}$$

## Proiezione conforme e conformal prediction

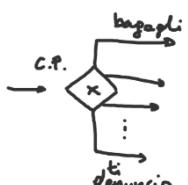
Siamo in una situazione del tipo:



Ottengo quindi una distribuzione di probabilità sulle classi. Un conformal predictor è un preditore del tipo



Che predice più di una classe! Per esempio, quindi, se ho significance = 0 ( $\varepsilon$  massima) allora mi risponde sempre con tutte le classi (massimo  $\delta$ ).



Ad esempio, prendiamo un bot che classifica il tipo di lamentela di un call center, da "non so a che ora è l'aereo" a "ora vi faccio causa".

Posso avere un bottino che mi riduce il campo di azione, e chiamo l'operatore umano solo quando sono indeciso fra le classi o ho delle classi speciali specifiche

Ovviamente l'ideale è avere che mi risponde con True sulla classe corretta, e False su tutte le altre.

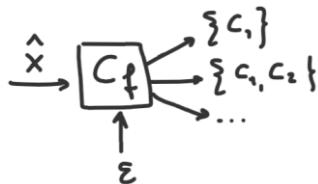
### Efficienza

In questo caso, l'efficienza è la media della somma dei True corretti.

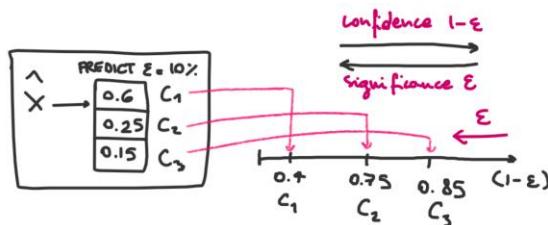
Per la miscalibrazione verifichiamo che al più  $\varepsilon$  volte il sample **non sta** in  $\hat{Y}_{TEST}$  ha classificazione  $\perp$ . E' utile perché così potremmo comunque fare un classificatore che restringe i casi difficili, pur non riconoscendo tutti i casi singolarmente.

05-nov-2023

## Non conformity per la calibrazione



Come per la volta scorsa, ma **anziché usare MSE usiamo una qualunque funzione che ci dà valore alto su cose canate**. Possiamo addirittura usare una classificazione anziché MSE!



In questo esempio ho tre candidati, che allineo sulla retta.

Quindi, **quale percentuale di samples non conformi devo lasciarmi "a destra" per garantire  $\varepsilon$** ? Ovvero, ripetendo: prendo il numero di sample; se il 90% dei sample di calibrazione sta a destra dell'  $1 - \varepsilon$  allora prendo solo la prima classe; per il 10 % dei valori è  $> 0.75$  prendo sia  $C_1$  e  $C_2$ ; altrimenti prendo tutti e tre. Per sapere l' $\varepsilon$  minimo con cui includo una classe devo sommare tutte le probabilità fino a quella della classe. Più è piccolo l' $\varepsilon$ , più sono costretto a includere cose nella regione.

Tutti gli intervalli fanno  $p = \frac{1}{n+1}$ ; (not sure what the fuck is going on here)

L'ordine con cui prendo le classi dipende dall'ordine con cui escono le classi del predict (aka, credo, più un sample è strano all'interno della sua classe, più viene a destra).

Posso anche trasformare questo classificatore multiclasse in un insieme di classificatori binari che mi danno 1 se appartiene alla classe e 0 anywhere else.

$$h_i \rightarrow (X, y).apply(\lambda x : x == C_i)$$

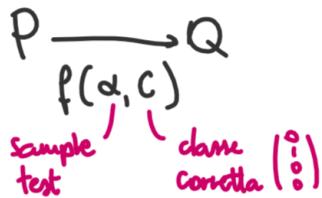
Il vantaggio di fare così è che i classificatori binari di solito funzionano bene anche in caso di dataset sbilanciati. Come funzione di non conformità prendiamo  $1 - h_i(x)$  per tutte le componenti. La somma delle componenti del vettore non viene 1, ma va bene lo stesso. 😊

E' dimostrato che facendo le cose in questo modo, assunto che i dati non sono miscalibrati, allora se do ad esempio 0.01 di  $\varepsilon$  su 100 sample mai visti ne sbaglio una quantità tendente a 1, e questo è valido per ogni funzione di non conformità.

### Dimostrazione

La dimostrazione è un'analisi di p-value.

Immaginiamo di avere una distribuzione di valori  $P$  e di applicarvi una funzione – che per noi sarà il classificatore; i dati usciranno dalla funzione con una nuova distribuzione.



In generale non conosciamo  $P$  e  $Q$ , ma sappiamo che entrambe sono distribuzioni.

Se prendiamo un insieme di sample

$$\alpha_1, \dots, \alpha_q \quad (\text{VALORI DI CALIBRAZIONE})$$

E prendo i valori

$$\frac{|\{d_i : \hat{\alpha}_c \leq \alpha_i\}| + 1}{q+1} \sim \begin{array}{l} \text{DISTRIBUZIONE} \\ \text{UNIFORME} \\ [0 \dots 1] \end{array}$$

Prendendo una serie di sample di test, e per ciascuno sappiamo la classe reale...

$$\hat{x}_1, \dots, \hat{x}_n$$

$c_1, \dots, c_n$

Se per ciascuno calcoliamo lo score della classe, esso punta a  $\alpha_c$ .

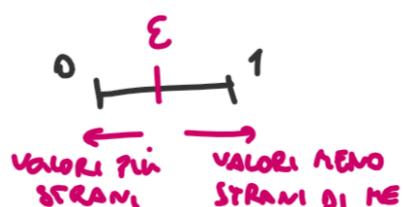
Il p-value ci dice che all'aumentare di questi valori, postulando (=ho un oracolo che mi dice) la classe giusta, la formula sopra tende alla distribuzione uniforme..

Se prendo un test set abbastanza grande di cui so anche la classe corretta, e per ogni elemento calcolo quanti elementi del dataset di calibrazione hanno una percentuale minore di sample, i valori sono distribuiti uniformemente tra 0 e 1.

Il dataset di calibrazione che ha valore di non conformità maggiore del valore di non conformità del sample sconosciuto nella classe vera del test set.

Essendo uniformemente distribuiti, ne sbaglio al più  $\varepsilon$ .

Praticamente, sto buttando via tutte le classi per cui c'è "gente più strana di me"



Anche se  $\hat{\alpha}_c$  è giusto, se mi risulta molto strano per la sua classe mi trovo a buttarlo via! Perché è uniformemente distribuito, e quindi ne sbagliero un  $\varepsilon\%$ .

## APPENDICI

12-dic-2023

Facciamo un po' di osservazioni a lato di cose da fare prima e dopo aver fatto la classificazione.

### Visualizzare una conformal

Possiamo usare una conformal in tanti contesti; per esempio, voglio predire se un paziente può sviluppare la sepsi.

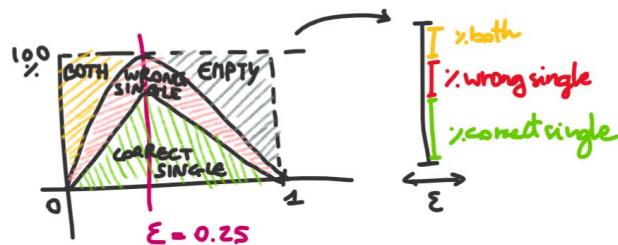
Faccio la predizione in ogni punto di tempo – per esempio a 1,8 e 16 ore dall'inizio della finestra – e mi aspetto che la percentuale di pazienti per i quali si risponde “non lo so” (e DEVO rispondere non lo so per poter garantire  $(1 - \varepsilon)\%$  categorizzazioni corrette!) diminuisca.

Dato che sono io a fissare una  $\varepsilon$ , e in questo caso scelgo 80 e rilevo:

- 1 h: solo il 57% dei pazienti viene classificato in una classe singola
- 8 h: 73%
- 16 h: 79%

Nota: per  $\varepsilon$  molto alte (ovvero voglio essere certo di sbagliare, anziché certo di rispondere giusto) potrebbe succedere che il mio sistema mi risponda *empty*, ovvero che per essere certo di canare non mi risponda proprio.

Un modo molto carino di plottare queste informazioni è lo **stackplot**:



In pratica, la somma di # wrong e # both sarà esattamente  $\varepsilon$  perché abbiamo già visto che ci garantiamo sia al massimo  $\varepsilon$

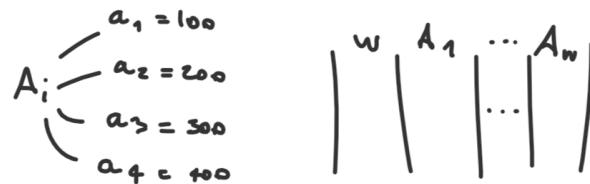
$$\text{EFFICIENZA} = \frac{\# \text{incorrect single} + \# \text{correct single}}{\# \text{Both}}$$

### Split dei decision tree usando $\chi^2$

Per scegliere la split di decision tree – o anche il boosting – posso usare altre misure oltre all'IG! Una di queste idee è di usare il chi quadro  $\chi^2$ .

Potrei voler decidere il miglior attributo su cui spartire (decision tree) o aumentare il peso (boosting).

Avendo 1000 sample, come per il p-value, guardo i valori marginali sull'attributo  $i$ :



Ortogonalmente contiamo le classi in ogni attributo, che presumibilmente saranno meschiate. Più la distribuzione delle classi è lontana dai valori marginali, più quel valore è significativo; il  $\chi^2$  misura quanto si discosta.

In generale un'ipotesi nulla mi aspetterei delle classi equamente distribuite, quindi in questo scenario avrei una cosa simile a questo esempio:

	$a_1$	$a_2$	$a_3$	$a_4$	Total (REALE)
$C_1$	30.2				308
$C_2$			10% di 308		272
$C_3$					420
Total (REAL)	100	200	300	400	1000
Dato che $a_1$ è il 10% del set totale, mi aspetto che il 10% dei sample $C_1$ abbiano $A=a_1$ (se $A$ è uniforme)					

Ovviamente la realtà non è uniformemente distribuita, quindi potrei avere un qualunque numero tale per cui sommando righe e colonne ottengo il numero sulle righe. Il punto è proprio che più i valori reali si discostano dal valore atteso, più quell'attributo deve esser ritenuto significativo!

Il  $\chi^2$  misura proprio questa differenza per ogni cella, secondo la formula

$$\chi^2 = \frac{(valore\ reale - valore\ atteso)^2}{valore\ atteso}$$

Sommo tutte le celle della tabella di cui sopra per avere la somma del  $\chi^2$  dell'attributo  $A_i$ , e poi scelgo quello con  $\chi^2$  maggiore.

## Discretizzazione usando $\chi^2$

Ho un set di attributi numerici e ho un classificatore che vuole numeri discreti, oppure voglio raggruppare i valori in cluster sensati. Il problema è che, se ho numeri molto lontani e con delta molto piccoli:

- Vorrei poter creare degli intervalli in base alle classi
- Vorrei misurare quanto gli intervalli sono interessanti rispetto alla classificazione

### Discretizzazione locale

E' l'equalizzatore delle vecchie radio. Il caso semplice è che prendo l'attributo e lo trasformo nel seguente modo:



! I classificatori molto simbolici (scemi) non colgono la relazione di  $>$  e  $<$ ; vedono solo il diverso. In questo caso non sarebbero molto contenti.

Tutto il resto è lasciato fare al classificatore.

### Local discretization

Altrimenti, partiamo da un classificatore numerico, ma non vogliamo semplicemente normalizzare: infatti se ho numeri molto sparsi ma con delta molto piccoli, rischio di perdere qualcosa a causa dell'aritmetica di macchina. Ad esempio:

$$10^{-6} \longleftrightarrow 10^6$$

↓ normalizzando

$$0.00\dots 01 \longleftrightarrow 0.999\dots 9$$

Un modo più carino di fare questa cosa è usando la local discretization. Posso ridurre a bit sia le entries che le classi, usando il one shot encoding:

$x$	$y$	$\geq 5$	$\geq 8$	$\geq 10$	$\geq 12$	$c_1$	$c_2$
8 $C_1$	8	1	1	0	0	1	0
5 $C_2$	10 $\Rightarrow$	1	1	1	0	0	1
10 $C_1$	5	1	0	0	0	1	0
12 $C_2$	12	1	1	1	1	0	1

In questo modo riesco a ridurre a bit l'intero problema.

Tale Darwiche sfrutta questa idea nel seguente modo: prende qualunque sample, lo trasforma in un vettore di bit, prende il classificatore come fosse una black box e cambia 1 bit di ciascun sample, 1 bit alla volta, per osservare come cambia il comportamento del classificatore al variare dei bit.

Da questo procedimento riesce a tirare fuori una formula logica – associando gli 0 e 1 a True e False.



Il diagramma che ottiene da questo procedimento si chiama BDD, e attraverso questo riesce a definire delle **sufficient reasons**: ad esempio, “mi basta guardare i primi N bit per saper già dire la classificazione”.

In teoria questo procedimento è esponenziale, e effettivamente lo è per le neural network, ma per strutture più semplici di cui ho informazioni sulla struttura diventa un problema facile.

Contribuisce all'explainability: ci dice che le uniche features su cui si basa il classificatore. In generale, più features sono prese davvero in considerazione nella neural network, più è difficile fregarla.

## Discretizzazione generale: $\chi^2$ merge

Voglio avere dei bucket e produrre une hot encoding anziché schifo encoding (?)

Potrei farlo con del normalissimo clustering, ma farlo con il  $\chi^2$  mi permette di tener conto della distribuzione delle classi.

Questa tecnica può essere usata anche per verificare se una mia classificazione è significativa o meno: se non lo è ottengo dei cluster non sensati.

Anche in questo caso procediamo costruendo una tabella delle frequenze, e ordino i valori osservati per grandezza e calcolando la frequenza di ogni valore. Quella che ho ottenuto è la peggiore discretizzazione possibile: tratto i singoli valori come classi. Quello che posso fare è collassare iterativamente i valori!

L'idea si applica iterativamente, calcolando il  $\chi^2$  per ogni coppia di righe e collassando insieme quelle con  $\chi^2$  minore, perché di minor interesse.

Come decido come fermarmi? Ci sono delle tabelle pre-calcolate, che in base alla confidenza desiderata ci dicono quale valore deve assumere il  $\chi^2$  per essere significativo; ad esempio, ho una significatività del 90% con un  $\chi^2 = 4.61$ . In questo caso, vado avanti finché ottengo che tutti i  $\chi^2 > 4.61$ . Eventualmente, posso aggiungere un bound inferiore e superiore di bucket da creare.

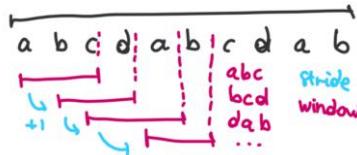
! Technicality: se ho un denominatore minore di 0.5, devo alzarlo a 0.5. !

## ASSIGNMENTS

### Esercizio 1: Association Rules

Data una sequenza di eventi, definisco due parametri:

- **Stride:** deve essere  $\geq 1$ . Specifica di quanto deve spostarsi avanti la sliding window ad ogni iterazione
- **Window:** deve essere  $\geq 1$ . Specifica quanto deve essere grande la sliding window.



Una volta che ho generato **tutte le sottosequenze** in questo modo, **stabilisco una sequenza di dimensione  $\leq$  a window e conto quante volte essa è presente nelle sottosequenze del passo precedente**.

! vale anche se non è contigua !

Quindi posso calcolare il supporto della sequenza come

$$\frac{\# \text{ sottosequenze che hanno la mia sequenza}}{\# \text{ totale sottosequenze trovate}}$$

#### Dataset

Come dataset possiamo sia usare il dataset del diabete in una versione più complessa dell'esercizio (la finestra, anziché essere lunga un certo numero di eventi, è grande un certo numero di giorni). O altrimenti prendiamo il dataset skating dalla repo del profe e usiamo quella 😊

### Esercizio 2: Shapley Value

Voglio valutare **quanto un elemento sia influente su una regola**, relativamente a una misura di bontà che posso scegliere a piacere. Per questo esercizio sceglio la **confidenza**, che è definita come

$$\text{confidence}(AB \rightarrow C) = \frac{\text{Sup}_X(ABC)}{\text{Sup}_X(AB)}$$

Per calcolare questo valore devo **calcolare tutte le permutazioni degli antecedenti**, e per ciascuna permutazione calcolare la **differenza di confidence all'aggiunta di ciascun elemento degli antecedenti** (nell'ordine dato dalla permutazione). Dopo di che, faccio la media sulle colonne tutte le differenze di ciascun elemento. Questa media è lo shapley value.

(img)

Quando sapete contare sapete fare tutto!

Questa misura è **molto usata in explainability**, anche in ambito di classifier: quando un classifier classifica un sample, vengono restituiti gli shapley value di ciascuna delle feature del sample per capire quale è stata più determinante nella classificazione.

Calcolare gli shapley value, però, è NP-completo (infatti ho le permutazioni). In ambito delle association rule riusciamo comunque a farlo, dato che normalmente non ho più di 5 antecedenti, ma in un classifier generico esplode. Si riesce comunque a usare questa misura calcolandone il valore atteso al posto del valore effettivo.

#### Dataset

Come dataset, posso prendere il risultato dell'esercizio 1: la funzione confidenza sarà

$$\text{confidence}(CAD \rightarrow B) = \frac{\# \text{ tuple che contengono } CADB}{\# \text{ tuple che contengono } CAD}$$

In alternativa, posso prendere qualunque sequenza e trasformarla nell'insieme delle etichette uniche – per non avere problemi con le ripetizioni.

### Esercizio 3: Sequence tree

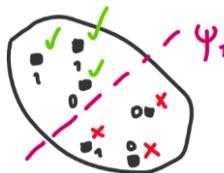
Come si faccia quest'esercizio è già stato specificato durante le lezioni. La cosa nuova interessante è la seguente: dato che il diabetes dataset dà solo le sequenze, senza una classificazione, il professore propone di farla in questo modo:

- Per ogni

### Esercizio 4: Boosting e ensemble classifier sulle sequenze

Richiamiamo l'esercizio sugli alberi per elaborare sequenze. Vogliamo usare lo stesso principio, ma con il boosting su ogni singolo test. L'idea è che cerco tra le varie durate e sequenze il test che mi spezza meglio il dataset (es. il test di evento che minimizza l'errore).

La  $\psi$  è il mio test, ed è solo una parte del mio weak classifier:



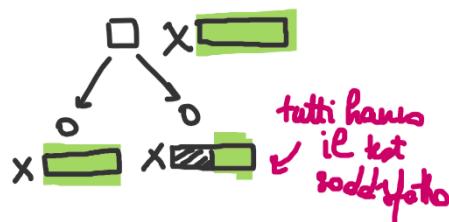
Il test  $\psi_1$  mi divide i samples, ma l'ottimalità del taglio di  $\psi_1$  dipende dalla  $Y$ . Ovviamente, data una  $\psi$ , possiamo sia decidere di dare classe 1 a True e 0 a False che, se più efficace, l'opposto.

$$\Psi_i = (\psi_i, 1)$$

↗ value to assign if  $x \models \psi$   
 ↗ test  
 ↓ actual function

Ad ogni passo del boosting provo tutte le possibili combinazioni di label  $L$ , durata  $d$  e valore  $v$  per trovare il test che mi dà maggiore information gain.

Quando usavamo gli alberi, avevamo i vantaggio di splittare il dataset ogni volta che avevo un test di evento: se ho un test di evento, tutti i samples che vanno nel ramo “true” – ovvero soddisfano l’evento – hanno sicuramente quella caratteristica.



Il boosting, invece... è sempre effettuato sull'intero dataset. Quello che succede, quindi, è che se provo a fare un test sul valore dopo aver fatto un test di evento, mi trovo ad avere che alcuni samples avranno il value che mi aspetto, mentre altri avranno null (se non hanno mai fatto test di evento a true) o addirittura valori non correlati (se hanno fatto test di evento su altre label a true).

Per esempio, prendiamo il seguente dataset:

$$x_1 \left[ \begin{array}{l} ((0, L_1, 1), (2, L_1, 2), (3, L_3, 10) \dots) \\ ((0, L_1, 1), (2, L_2, 10), (4, L_5, 12) \dots) \end{array} \right] \left[ \begin{array}{l} \text{null} \\ \text{null} \end{array} \right] \left[ \begin{array}{l} V \\ ST \end{array} \right]$$

Decidiamo di fare un test di evento del tipo  $\psi = (L_1, 2)$  – ovvero voglio trovare un evento di label  $L_1$  con una distanza al più di 1. Possiamo formalizzarlo come

$$\Psi = \exists i : vt_i - vt \leq 2 \wedge L_i = L_2$$

E quindi il test vero e proprio sarà

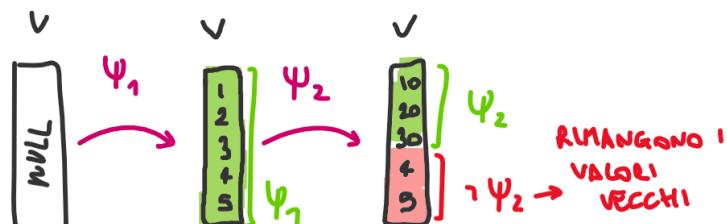
$$\Psi = (\Psi, 1)$$

Quello che succede è che il primo sample viene riprodotto proprio tale e quale, perché non ho nessuna  $L_2 < 2$ ; mentre nel secondo sample viene rilevato un elemento della sequenza che soddisfa la proprietà, quindi la sequenza viene “mangiata” e gli array  $V$  e  $ST$  vengono aggiornati.

$$x_1 \left[ \begin{array}{l} ((0, L_1, 1), (2, L_1, 2), (3, L_3, 10) \dots) \\ ((2, L_2, 10), (4, L_5, 12) \dots) \end{array} \right] \left[ \begin{array}{l} \text{null} \\ 10 \end{array} \right] \left[ \begin{array}{l} V \\ ST \end{array} \right]$$

Ecco il problema: se ora facessi il test di valore, mi troverei ad avere che  $x_1$  non ha nessun valore!!!

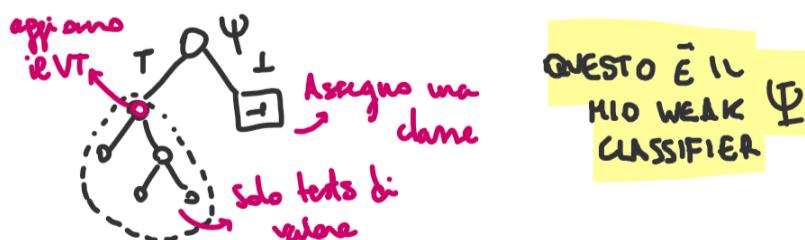
Come sistemiamo? Il problema si pone solamente dopo un test di evento. Ed è molto brutto, perché potrei avere dei valori vecchi di alti test di evento.



Quindi dobbiamo trovare un altro approccio.

Possiamo fare che invece che portarci dietro i valori e incastrare n test “di riserva”, la cosa migliore è che ogni classificatore weak del boosting sia un albero dell’esercizio precedente in cui:

- La radice è  $\psi$  (test di label)
  - Sottoalbero vero: posso avere solo ulteriori test di valore
  - Sottoalbero falso: è una classe



A ogni passo rifaccio la stessa cosa.

(pseudocodice)

La differenza è che invece di dare sempre lo stesso sample e moltiplicare come facevo nel boosting normale, adesso li vedo in sequenza: il sample che do al test i-esimo dipende da cosa ho consumato ai passi (i-1)-esimi.

Il test  $\psi$  può essere usato in due modi:

- $\psi_i.predict$  torna 1 o -1 applicando il sample all'albero di test
- $\psi_i(\hat{X})$  ritorna un  $VT_i, X_i$  consumando la sequenza.

Il ricacolo dei pesi è esattamente quello del boosting normale:

$$\begin{aligned} \varepsilon_i &= 1 - \sum_{j=1}^{i-1} w_j \underbrace{\psi_i.predict(VT_j, X_j) = y_j}_{\text{= quelli dannificati giunti}} \\ &= \sum_{j=1}^{i-1} w_j \underbrace{\psi_i.predict(VT_j, X_j) \neq y_j}_{\text{= quelli dannificati sbagliati}} \end{aligned}$$

Con questo possiamo calcolarci  $\alpha_i$ :

$$\alpha_i = \frac{1}{2} \ln \left( \frac{1-\varepsilon_i}{\varepsilon_i} \right)$$

E poi ricalcolo il peso come

$$w_i[j] = \frac{\text{peso vecchio}}{\text{Soma per i dannificati correttamente al passo } i-1}$$

Esempio

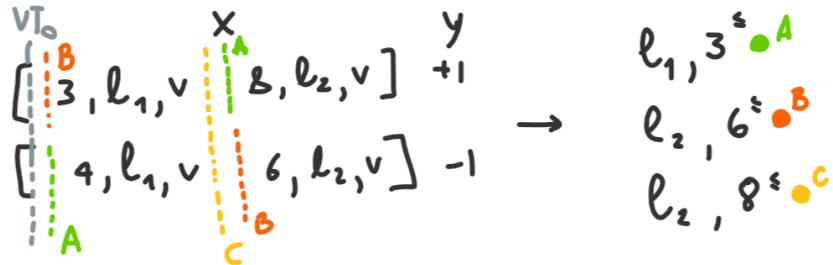
peso al passo i		predizione attese?	
$w_1$		✓	
$w_2$		✗	
$w_3$		✓	
$w_4$		✗	
$w_5$		✓	
$\sum w_i = 1$			
			$\text{per i dannificati} = W_T = w_1 + w_3 + w_5$
			$\text{per i sbagliati} = W_L = w_2 + w_4$
			$w_2^{i+1} = \frac{w_2}{2 \cdot W_T} \quad W_2^{i+1} = \frac{w_2}{2 \cdot W_L}$

Criterio di stop

Ho molte opzioni fra cui scegliere:

- $\varepsilon$  scende sotto una certa soglia
- Le classificazioni sono tutte corrette
- Ho esaurito tutte le sequenze
- $\varepsilon_i = 0.5$ , perché in questo caso l'ensemble classifier non funziona
- # massimo di step

Esempio di applicazione di Best Tree

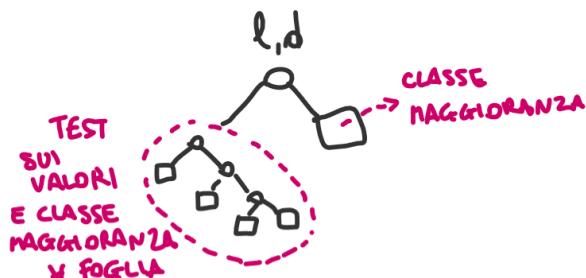


Voglio testare tutte le coppie di  $l, d$  possibili. Chiaramente così avrei una marea di coppie: posso migliorare la cosa emettendo solo quelle coppie che mi generano degli alberi / "slices" diversi.

Voglio tutti i possibili sottoinsiemi che possiamo enumerare con un solo test.

Sarebbe esponenziale, ma avendo i vincoli sui test mi diventa polinomiale.

Preso ciascuna possibile coppia  $l, d$ , per essa esiste un solo albero ottimo. Lo costruisco con la stessa struttura

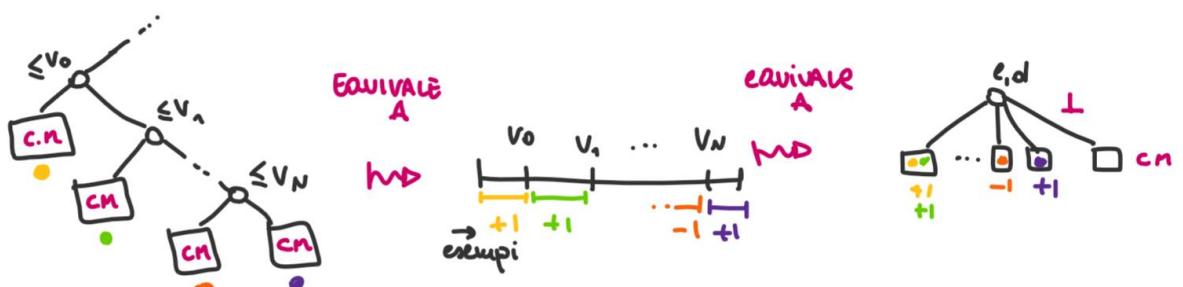


Dove

- Calcoliamo la classe maggioranza come

$$\text{sign}\left(\sum_{w_T} w - \sum_{w_L} w\right)$$

- Il sottoalbero dei value è fatto da un nodo value per ogni sample, dato che prendo solo la prima occorrenza di  $l$  della sequenza. Ordino i valori, per ognuno genero un nodo e creo la seguente struttura per rappresentare gli intervalli:



Come nel caso più a destra, nulla mi vieta di ottimizzare la struttura ponendo direttamente tutte le foglie sullo stesso livello in un solo test, e unendo gli intervalli adiacenti che hanno classificazione alla stessa classe.

## Esercizio 5: Boosting su sequenze multiclass con conformal classifier

---

Creiamo una conformity function e la testiamo sul boosting appena creato.

Prendiamo il boosting classifier dell'esercizio precedente e lo definiamo come

$$\varphi = (\alpha_1, \psi_1, \dots, \alpha_n, \psi_n)$$

Dove  $\psi_i : vt_i, \hat{x}_i \rightarrow \{+1, -1\}$

Da questa struttura costruiamo la probabilità: dato che tutti gli  $\alpha$  sono  $> 0$ , possiamo definire

$$p = \frac{\sum_{\psi_i=1} \alpha_i}{\sum \alpha}$$

Da qui definiamo la funzione di non conformità come

$FNC = 1 - p(\hat{x})$  per la classe +1,  $FNC = p(\hat{x})$  per la classe -1.

Vogliamo sfruttare la conformity per passare a un setting multiclass in cui ho delle classi  $\{0 \dots C - 1\}$

Dato  $\varphi$  posso sempre tirare fuori un classificatore calibrato che prende in input  $\hat{x}, \varepsilon$ .

$$\varphi(\hat{x}) \in [-1, 1]$$

$\downarrow$  con  $X_C, Y_C$

$$\Phi(\hat{x}, \varepsilon) \subseteq \{-1, 1\}$$

Supponiamo di avere già  $C$  classificatori conformi che chiamiamo  $\Phi_{CL}(\hat{x}, \varepsilon)$ , uno per ogni classe: è allenato a rispondere 1 se appartiene a quella classe, e -1 se appartiene a qualunque altra.

## SEZIONE DI LABORATORIO: CODICE VISTO IN CLASSE

### Algoritmo Apriori

#### Association rules: Air Quality dataset

##### 1. Caricamento dei dati dal file CSV:

```
import pandas as pd
import numpy as np
aq = pd.read_csv('air+quality/AirQualityUCI.csv', sep=';')
```

Questo codice utilizza la libreria Pandas per caricare i dati dal file CSV 'AirQualityUCI.csv' nel dataframe aq. Il separatore delle colonne è impostato su ;.

##### 2. Selezione delle colonne e rimozione delle righe con valori mancanti.

```
aq = aq[aq.columns[2:-2]].dropna()
```

##### 3. Selezione delle colonne di classe e delle colonne di caratteristiche:

```
classes = list(filter(lambda x: x[-4:] == '(GT)', aq.columns))
features = list(set(aq.columns) - set(classes))
```

Viene creata una lista `classes` contenente le colonne che terminano con '(GT)'.

La variabile `features` contiene le colonne di caratteristiche, ottenute sottraendo le colonne di classe da tutte le colonne disponibili nel dataframe.

`lambda x: x[-4:] == '(GT)':` Questa è una funzione lambda, una funzione anonima, che prende un argomento x (che è un nome di colonna) e restituisce True se gli ultimi quattro caratteri di x sono '(GT)', altrimenti restituisce False.

##### 4. Ulteriore pulizia del dataframe

```
# Prendi solamente le colonne delle features
aq = aq[features]

# Per i valori indicati converti la stringa in formato numerico.
aq["T"] = aq["T"].apply(lambda x: float(x.replace(',', '.')))
aq["RH"] = aq["RH"].apply(lambda x: float(x.replace(',', '.')))
aq["AH"] = aq["AH"].apply(lambda x: float(x.replace(',', '.')))
```

```
# Rinomina le colonne con nomi decenti.
aq = aq.rename(columns={
    'PT08.S1(CO)': 'CO',
    'PT08.S5(03)': 'OO',
    'PT08.S3(NOX)': 'NOX',
    'PT08.S2(NMHC)': 'NMHC',
    'PT08.S4(NO2)': 'NO2',
    'T': 'Temperature'})
```

```
# Se il numero è negativo sostituisci con NaN
for c in aq.columns:
    aq[c] = aq[c].apply(lambda x: x if x >= 0 else np.nan)
```

## 5. Cluster