

# **Big Data and Text Analysis**



## Contents



# Generalities and Map Reduce

## 1 Generalities and Map Reduce

I problemi "big data" fanno riferimento a quella tipologia di problemi dove il carico dei dati da elaborare è MOLTO grande. Per risolvere un problema di questo tipo, una volta, avremmo fatto affidamento su un unico "super computer". Oggi, si utilizzano strutture di più dispositivi uniti chiamate **computer clusters**.

### Motivation: Google Example

- 10 billion web pages
- Average size of webpage = 20KB
- 10 billion \* 20KB = 200 TB
- Disk read bandwidth = 50 MB/sec
- Time to read = 4 million seconds = 46+ days
- Even longer to do something useful with the data

L'esempio principale lo abbiamo da Google: elaborare dati con poche macchine sarebbe infattibile. Quello che faremo sarà quindi andare a mettere insieme più server ma la loro gestione rimane un problema grande. Cosa succede se ho bisogno di un'informazione ma si rompe una macchina che permetteva di accedervi?

### 1.1 Introduction to Distributed File Systems

Consiste nello store dei dati in maniera *redundant*: ovvero frammentando l'informazione in più chunks per potervi risalire in qualunque momento. In questo modo riesco ad avere più copie approssimative della mia informazione sparse per la mia rete. La dimensione dei chunk e il grado di riproduzione del dato sono decisi dall'utente.

*Come gestisco i chunk?* Il **Master node** è il contenitore del filesystem tree, delle metadata e le directories. Attraverso esso gestiamo i chunks. Anche il master node viene replicato.

Ma quindi, come elabro i dati nel Distributed File System? Con il metodo **Map Reduce**.

## 1.2 Map Reduce

Ci permette di leggere sequenzialmente grandi quantità di dati. Si compone principalmente di tre step:

1. Map
2. Group by key
3. Reduce

### map

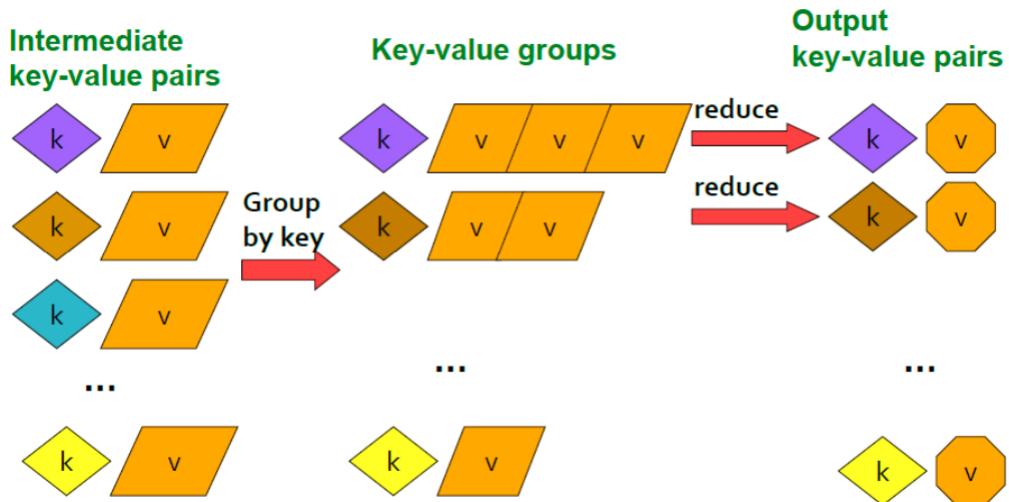
Il "map" è il primo step dove essenzialmente vado alla ricerca delle informazioni utili. Corrisponde di fatto ad una query dove isolo elementi secondo una certa caratteristica. *Extracting something of interest*

### group by key

Fare "group by key" significa letteralmente "raggruppare per chiave". Aggrego cose simili tra di loro, ma tengo il conto di quante ne ho raggruppate. Di fatto, metto insieme la chiave e tutti i valori a lei associati. *Sort and shuffle*

### reduce

Mette insieme tutto alla fine, aggrega per avere dei dati più compatti. Risparmiando quindi memoria semplicemente associando, ad ogni elemento diverso del documento, il numero di volte che si ripete. *Aggregate, summarize, filter, transform*



Piccolo approfondimento per pura curiosità: è nato per risolvere il problema della ricerca e conteggio di parole per Google.

### 1.3 Distributed File System Architecture w/ Map Reduce

Possiamo immaginare il Master Node come colui che gestisce gli altri nodi, identifica le operazioni che gli altri devono compiere. Ad esempio, nella ricerca e conteggio delle parole, una parola che si ripete lui la invia a chi di dovere per farla ridurre, ovvero identifica un gruppo di nodi adibiti a questa funzione. Nell'immagine seguente è possibile notare tutti i pezzi di un Distributed File System e la loro funzione.

## Distributed File System

- **Chunk servers**
  - File is split into contiguous chunks (16-64MB)
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks
- **Master node**
  - a.k.a. Name Node in Hadoop's HDFS
  - Stores metadata about where files are stored
  - Might be replicated
- **Client library for file access**
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data

Inoltre, il Master Node controlla e pinga periodicamente i workers per trovare delle failures (malfunzionamenti).

**Cosa succede se avviene un guasto?** Dipende da che parte della struttura si rompe:

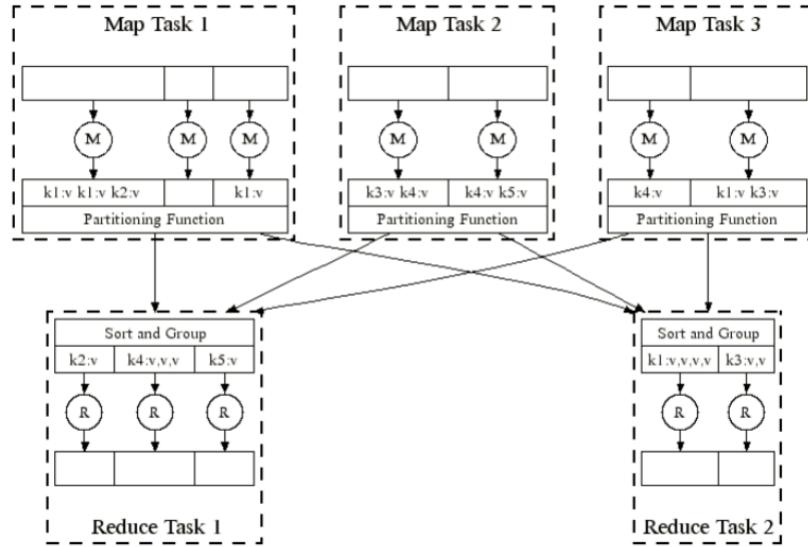
- Map worker: le task del worker sono resettate, i reduce workers vengono notificati quando la task viene compiuta da un altro worker
- Reduce worker: solo le task in-process del worker vengono resettate, la task di reduce viene restartata
- Master failure: MapReduce task abortita e contattato il client

La funzione di Reduce è associativa e commutativa, quindi va utilizzata in casi compatibili ad essa: se devo contare o sommare, posso utilizzarla. Ad esempio invece se dovessi fare la media mi sarebbe impossibile farlo!

I valori possono essere combinati a piacimento che danno lo stesso risultato. I valori delle key/value di input devono essere dello stesso tipo dei valori delle key/output. (Commutativa e Associativa)

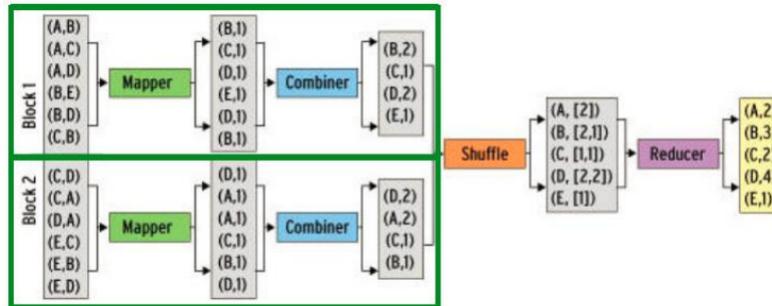
## 1.4 MapReduce in parallelo

Vengono mappati più valori e viene eseguita la reduce su più dati separati secondo un determinato criterio:



## 1.5 Combiners

I combiners sono elementi che ci aiutano a combinare il valore di tutte le chiavi di un singolo mapper (un singolo nodo) così non è necessario copiare e mescolare tutti questi dati.



## 1.6 Remarks

Per raggiungere il massimo parallelismo, potremmo usare un Reduce Task per eseguire ogni reducer oppure eseguire ogni Reduce task in un nodo diverso. Ma tutte queste ipotesi genererebbero solo dei problemi in più:

- potrebbero esserci più chiavi dei nodi che abbiamo.
- potrebbe esserci una variazione eccessiva della lunghezza delle liste di valori per chiavi diverse
- c'è un overhead associato ad ogni task che creiamo

## 1.7 Algoritmi che usano Map Reduce

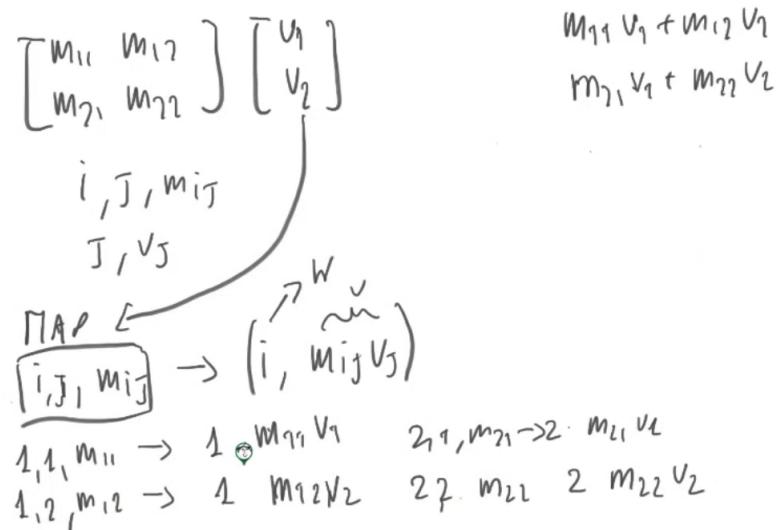
*Elevata probabilità di essere chiesto.* Esistono diversi algoritmi che sfruttano il map reduce, come la moltiplicazione vettoriale oppure le operazioni di algebra relazionale.

### 1.7.1 Matrix-Vector multiplication - Vettore che sta in memoria

Matrice  $M = n \times n$  ( $m_{ij}$ ),  $v$  = vettore di  $n$  componenti. Il prodotto matrice vettore fornisce come risultato:

$$x_i = \sum_{j=1}^n m_{ij}v_j$$

$M$  e  $v$  sono salvati entrambi nel DFS (Distributed File System) come coppie  $(i, j, m_{ij})$  e  $(j, v_j)$ . Per fare il map consideriamo sempre che  $v$  stia fisicamente in memoria. Ogni Map Task opera su un chunk di  $M$ . Per ogni  $m_{ij}$  che legge, genera un  $(i, m_{ij}v_j)$ , che banalmente è la moltiplicazione del valore  $ij$ -esimo della matrice per il valore  $j$ -esimo del vettore, e gli associa un altro indice  $i \rightarrow$  la Reduce Task invece somma tutti i valori associati alla stessa key  $i$ , ottenendo  $(i, x_i)$ . Si può vedere perfettamente questo passaggio dalle slide del professore:



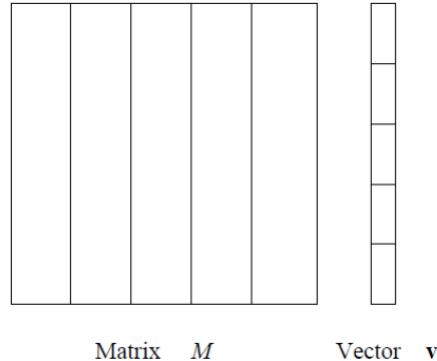
Ora abbiamo il Group By Key Task, che funziona esattamente come spiegato: mette insieme gli elementi che hanno stessa chiave. La chiave, in questo caso, è la **riga della matrice**. Quindi, dopo la Reduce Task che fa la somma e osservando sempre l'immagine, il risultato che otterremo sarà:

$$[1, m_{11}v_1 + m_{12}v_2], [2, m_{21}v_1 + m_{22}v_2]$$

Che è esattamente il risultato dell'operazione matrice per vettore.

### 1.7.2 Matrix-Vector multiplication - Vettore che non sta in memoria

Con  $v$  che non sta in memoria, semplicemente separiamo gli elementi di  $v$  in modo da ottenerne una quantità fattibile per la memoria. La cosa importante è che la parte della matrice che sarebbe da moltiplicare per quegli elementi, venga associata correttamente. Quindi dividiamo il vettore e la matrice in bande, e le associamo l'un l'altra, ottenendo una struttura ben associata e definita.



### 1.7.3 Relational algebra operations - Selection

$\sigma_C(R)$

Mapping → per ogni tupla  $t$  che soddisfa  $C$ , emetti  $(t, t)$

Reducing → emette l'identità. NON fa altre operazioni.

### 1.7.4 Relational algebra operations - Projection

$\pi_S(R)$

Mapping → per ogni tupla  $t$  costruisci  $t'$  rimuovendo i componenti  $i$  cui attributi non sono in  $S$  emetti  $(t', t')$ .

Reducing → emetti i key value pairs  $(t', t')$  rimuovendo i duplicati.

### 1.7.5 Relational algebra operations - Union

$UNION(R, S)$

Mapping → per ogni tupla  $t$  in input, emetti  $(t, t)$

Reducing → emetti  $(t, t)$ . Associati alla chiave ci sono uno o due valori. L'unione di fatto elimina i duplicati.

### 1.7.6 Relational algebra operations - Intersection

$INTERSECTION(R, S)$

Mapping → per ogni tupla  $t$  in input, emetti  $(t, t)$

Reducing → emetti  $(t, t)$  solo se ci sono 2 valori uguali (se le tabelle quindi si intersecano nella tupla).

### 1.7.7 Relational algebra operations - Difference

*Difference*( $R - S$ )

Mapping → per ogni tupla  $t$  di  $R$  in input, emetti  $(t, R)$  e per ogni tupla  $t$  in  $S$  emetti  $(t, S)$   
 Reducing → Per ogni key  $t$ , se la lista dei valori associati è  $R$  emetti  $(t)$  altrimenti nulla.

### 1.7.8 Relational algebra operations - Natural Join

$R(a, b) JOIN S(b, c)$

Mapping → per ogni tupla  $(a, b)$  in  $R$  emetti  $(b, (a, R))$  e per ogni tupla  $(b, c)$  in  $S$  emetti  $(b, (c, S))$   
 Reducing → Ogni key value  $b$  è associato a una lista di coppie  $(a, R)$  e  $(c, S)$  da cui puoi costruire tutte le coppie prendendo un elemento da  $R$  e uno da  $S$ .

### 1.7.9 Relational algebra operations - Grouping and Aggregation

$R(A, B, C) \gamma_{A, \theta(B)}(R)$

Mapping → per ogni tupla  $(a, b, c)$  produce il key value  $(a, b)$

Reducing → applica l'aggregazione  $\theta$  alla lista  $[b_1, b_2, \dots, b_n]$  associata ad  $a$ ; il risultato è una coppia  $(a, x)$  dove  $x$  è il risultato dell'aggregazione  $\theta$ .

Se ci sono attributi di aggregazione multipli il key value della map function è la lista dei valori e la reduce si applica ad ognuno di essi.

### 1.7.10 Matrix multiplication

Il risultato di una moltiplicazione matriciale date due matrici  $m$  e  $n$  di dimensione  $r_m \times c_m$  e  $r_n \times c_n$  è una matrice  $r_m \times c_n$ , i cui valori:

$$p_{ik} = \sum_j m_{ij} n_{jk}$$

$$\begin{matrix} & 4 \times 2 \text{ matrix} \\ \left[ \begin{matrix} \color{red}{a_{11}} & \color{red}{a_{12}} \\ \cdot & \cdot \\ \color{orange}{a_{31}} & \color{orange}{a_{32}} \\ \cdot & \cdot \end{matrix} \right] & \left[ \begin{matrix} & 2 \times 3 \text{ matrix} \\ \cdot & \color{purple}{b_{12}} & \color{blue}{b_{13}} \\ \cdot & \color{purple}{b_{22}} & \color{blue}{b_{23}} \end{matrix} \right] = \left[ \begin{matrix} \cdot & x_{12} & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & x_{33} \\ \cdot & \cdot & \cdot \end{matrix} \right] \end{matrix}$$

$$x_{12} = (\color{red}{a_{11}}, \color{red}{a_{12}}) \cdot (\color{purple}{b_{12}}, \color{purple}{b_{22}}) = \color{red}{a_{11}} \color{purple}{b_{12}} + \color{red}{a_{12}} \color{purple}{b_{22}}$$

$$x_{33} = (\color{orange}{a_{31}}, \color{orange}{a_{32}}) \cdot (\color{blue}{b_{13}}, \color{blue}{b_{23}}) = \color{orange}{a_{31}} \color{blue}{b_{13}} + \color{orange}{a_{32}} \color{blue}{b_{23}}.$$

**Come viene risolta con Map Reduce?** Viene svolta con due Map Reduce in cascata. Nella Map Task prendiamo ogni elemento della matrice  $m$  e  $n$  e genero delle coppie chiave valore:

$$(j, (M, i, m_{ij})), (j, (N, k, n_{jk}))$$

Dove  $M$  e  $N$  sono i nomi delle tue matrici. Ora nella Group by key Task andremo ad associare gli elementi con stessa chiave, producendo un elemento con chiave  $(i, k)$  e valore  $(m_{ij} n_{jk})$  e nella Reduce Task, faremo la somma.

## 1.8 Remarks

Per raggiungere il parallelismo massimo, possiamo: usare una Reduce task per eseguire tutti i reducer, oppure svolgere tutte le Reduce task ad un computer node diverso. Ma non è una linea guida ottimale, perché ad esempio potremmo ottenere più keys dei compute nodes disponibili. Ulteriore problema è dato nell'esecuzione delle Reduce Tasks separate, che va ad aumentare il tempo di ogni operazione.

Per guardare il **costo** del Map Reduce, devo guardare il quantitativo dei dati, perché l'operazione più costosa è il trasferimento di essi. A questo punto una soluzione sicuramente potrebbe essere quella di tenere una sola macchina (rimuovendo il parallelismo) necessitando così di una macchina sola che contenga tutti i miei dati.

**Dobbiamo bilanciare queste due cose.** Lo facciamo basandoci su due parametri, la **Reducer size** (che indichiamo con  $q$ ) e il **Replication rate** (che indichiamo con  $r$ ).

### 1.8.1 Reducer size $q$

Questo parametro indica il numero di valori associabili ad una key. Nell'architettura Combiner, questo valore è fisso a 1.

### 1.8.2 Replication rate $r$

Questo parametro indica il numero di coppie key-value che la lettura di un input mi genera. Esiste un algoritmo (non trattato a lezione) che è quello della moltiplicazione matriciale in uno step solo, che ha come  $r$  il numero  $k$  delle colonne quindi ad ogni input genero  $k$  coppie. Ma di base lo abbiamo sempre considerato 1.

#### Relazione tra $r$ e $q$

Sono inversamente proporzionali. Non si è espresso più di tanto riguardo a questi due valori perché basta semplicemente sapere che giocando sulla loro proporzionalità è il modo giusto per bilanciare il costo delle operazioni.

### 1.8.3 Similarity Join

*Argomento importante.* L'esempio classico che lui utilizza per spiegare questo argomento, è un ds con  $10^6$  immagini dove vogliamo cercare delle similitudini. Se utilizzassimo Map Reduce e applicassimo la Map Task ad un input  $(1, I_1)$ , mettendolo in relazione con la generica immagine  $(j, I_j)$ , otterremmo una coppia di questo tipo  $(i,j)(I_i, I_j)$  il che vorrebbe dire ottenere per ogni immagine, 999'999 valori. Replication rate  $r$  elevatissimo, moltiplicato per 1 MB di immagine, e per ogni immagine presente (perchè il ragionamento è da iterare su tutte) otteniamo:

$$999'999 \times 10^8 \times 10^6 = 10^{18}$$

Che sono EB. Con una rete Gbit,  $10^8$  bit/s, ci vogliono 300 anni. L'approccio migliore da usare è quello della similarity join per cui separo il dataset in gruppi. Ottengo un  $r = g - 1$  ( $g$  è il numero di gruppi) quindi più gruppi faccio, più riduco il costo delle operazioni. Devo sempre tenere conto però che riducendo  $r$ ,  $q$  aumenta. La  $q$  infatti diventa  $\frac{n}{g}$  ovvero il numero di dati diviso il numero di gruppi, che indica il valore massimo che può assumere una chiave (se fosse simile a tutte le immagini del suo gruppo).



# Datamining and Machine Learning

## 2 Data Mining

Nel Machine Learning, come vedemo prossimamente, non si scrivono istruzioni ma è la macchina che impara direttamente dai dati. Il **Data Mining** è quella parte dell'informatica che si occupa di cogliere delle relazioni tra grandi quantitativi di dati. L'assunzione che noi facciamo è che il dato dica qualcosa di vero. Ma questa cosa non è sempre vera!

### 2.1 Principio di Bonferroni

Questo principio è esattamente la rappresentazione di ciò che abbiamo appena detto. Esso dice che è possibile imparare un'informazione sbagliata completamente a caso guardando in grandi distribuzioni di dati e trovando uno di quelli che in letteratura vengono chiamati *falsi positivi*.

#### 2.1.1 Esempio dei malfattori

Per capire ancora meglio il discorso introdotto, è utile soffermarsi su un esempio discussso a lezione, dove si vuole analizzare la seguente casistica: si vuole prevedere il numero di coppie di malfattori che si trovano negli hotel per organizzare attentati. Questo valore lo troviamo cercando le coppie di persone, su un db di un miliardo di nomi, che si trovano nello stesso hotel almeno 2 volte nell'arco di 3 anni.

$$\begin{array}{l}
 10^9 \text{ persone} \\
 1 \text{ notte su } 100 \quad 100 \text{ notti} \quad 10^{-18} \\
 10^5 \text{ Hotel} \\
 1000 \text{ giorni} \\
 \\ 
 5 \times 10^{17} \quad 5 \times 10^5 \quad 10^{11} \\
 \text{coppie} \quad \text{giorni} \quad 25 \times 10^4 \\
 \text{persone} \quad \text{giorni} \quad 250'000
 \end{array}$$

Otteniamo, facendo delle stime di probabilità:

$$10^{-2} \text{ probabilità che una persona passi una notte su 100 in hotel}$$

$$10^{-2} \times 10^{-2} \text{ probabilità che due persone passino una notte su 100 in hotel}$$

$$\frac{10^{-4}}{10^5} \text{ probabilità che due persone passino una notte nello stesso hotel}$$

$$10^{-9} \times 10^{-9} \text{ probabilità che due persone passino due notti nello stesso hotel nel periodo considerato}$$

E quindi salviamo un  $10^{-18}$ . Ora calcoliamo:

$$\frac{10^9 \times 10^9}{2} = 5 \times 10^{17} \text{ numero combinazioni coppie} \quad \frac{10^3 \times 10^3}{2} = 5 \times 10^5 \text{ numero combinazioni giorni}$$

E con questi due valori, uniti alla probabilità di trovare una coppia sospetta, calcoliamo il numero atteso di coppie sospette:

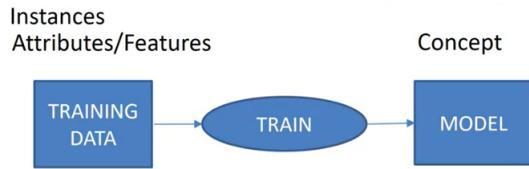
$$5 \times 10^{17} \times 5 \times 10^5 \times 10^{-17} = 250'000$$

Ma su un miliardo di persone, è normale trovare anche più coppie che si troveranno nello stesso hotel in quel lasso di tempo, potrebbero essere bambini, persone sposate, persone fidanzate ecc... Quindi è necessario un lavoro sui dati da studiare.

## 3 Machine Learning

### 3.1 Perchè utilizzare il Machine Learning?

Parliamo quindi ora del Machine Learning, e nello specifico, della classica pipeline quando si parla di ML.



Il Machine Learning è la scienza che si occupa di programmare un computer affinchè *impari dai dati*. Gli esempi da cui esso impara vengono chiamati "training instances" e la parte del sistema che impara e fa le prediction è chiamata modello. Neural Networks e Random Forest sono esempi di modelli.

**Ma perchè utilizzarlo?** Solitamente, in un qualsiasi codice "automatico" avresti necessità per certi aspetti di elencarglieli uno ad uno: facciamo un esempio più pratico. Scriviamo il codice dello *spam filter* e elenchiamo tutte le possibili parole chiave che ci aiutino a riconoscere gli spam. Se dovessimo sbagliare a scrivere, o queste key dovessero cambiare, saremmo costretti a cambiare il codice. Un modello di ML invece, imparerebbe direttamente dai dati, sopportando a tutti problemi elencati; potrebbe aggiornarsi per rimanere al passo con gli spam più evoluti e non farebbe errori di trascrizione o riconoscimento.

### 3.2 Tipologie di apprendimento

Tra le più famose elenchiamo (in ordine sparso):

- Classificazione
- Regressione
- Clustering
- Data Reduction
- Association Learning
- Similarity Matching
- Profilazione
- Link Prediction
- Causal Modeling

### 3.3 Risultato

Il risultato del ML è un modello che mi genera una predizione con un grado elevato di riuscita. Migliori saranno i dati di input, migliori saranno le predictions. Le istanze di train devono essere ben distribuite e soprattutto devono rappresentare fedelmente la feature su cui si vuole fare una predizione.

In sostanza, il Machine Learning è utile per:

- problemi la cui soluzione richiede molto fine-tuning e un codice spropositato.
- problemi complessi per cui non è possibile utilizzare un approccio tradizionale
- environments dove i dati sono soggetti a cambiamenti
- ottenere informazioni sulle distribuzioni di big data

### 3.4 Tipologie di modelli

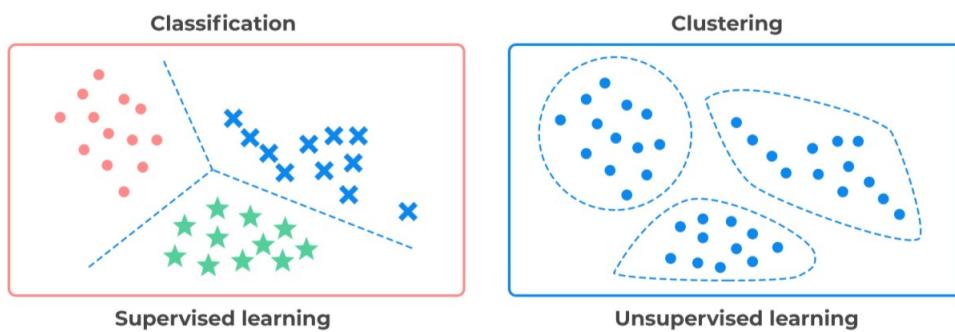
Un modello di ML quando si distingue per vari criteri:

- il tipo di *supervisione* durante il training
- se possono aggiornarsi col tempo
- se lavorano semplicemente sui dati conosciuti, trovando la relazione tra nuovi data e quelli vecchi, o se imparano dei pattern

Durante questo corso, abbiamo parlato principalmente di modelli di ML che apprendono in maniera supervisionata, e che non si aggiornano col tempo.

**Cosa si intende per studio supervisionato?** Si parla comunemente di *supervised* ML quando il modello si allena su un training set che contiene già le soluzioni desiderate chiamate *label*. Un esempio sarebbe un dataset dei dati clinici di alcuni pazienti e una feature `Heart_Failure` a valori binari [0, 1] che indica se questi pazienti hanno avuto un infarto. Possiamo allenare un modello a prevedere le probabilità di infarto di nuovi pazienti su questo dataset, considerando le cartelle cliniche dei pazienti che l'hanno avuto e la loro sintomatologia.

Si fa distinzione tra *supervised learning* e *unsupervised learning*. Nel caso di un apprendimento non supervisionato, come si può dedurre banalmente, il training viene eseguito su un dataset senza una label desiderata: il modello prova ad imparare "senza un istruttore" e quindi le operazioni più frequenti sono quelle di clustering, dove si cerca una relazione di similitudine tra i dati.



## 3.5 Regressione

### 3.5.1 Regressione lineare

Immaginiamo di voler stimare la qualità della vita di un individuo, a partire dal PIL pro capite. Potremmo quindi indicare la *life\_satisfaction*:

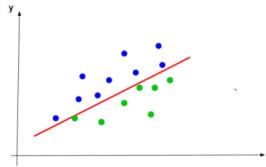
$$\text{life\_satisfaction} = \theta_0 + \theta_1 \times \text{PIL}$$

E avremmo così una relazione lineare tra i due valori (al crescere del PIL, la qualità della vita migliora). Generalmente, un modello lineare fa una prediction calcolando semplicemente una somma pesata delle input features, più una costante chiamata *bias*.

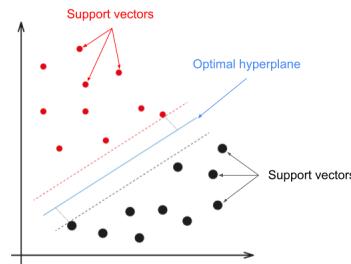
- $\theta_0$  è il bias term, o intercept term
- $\theta_1$  è il peso
- PIL (se vogliamo esprimere un generico valore, metteremo  $x$ ) indica la nostra input feature
- life\_satisfaction ( $y$ ) sarebbe il valore predetto

E ricadiamo nella categoria di **regressione**.

**Un modello lineare può essere utilizzato per classificazione?** Certamente. Esistono diversi modelli lineari, che attraverso la separazione dei dati, effettuano classificazione.



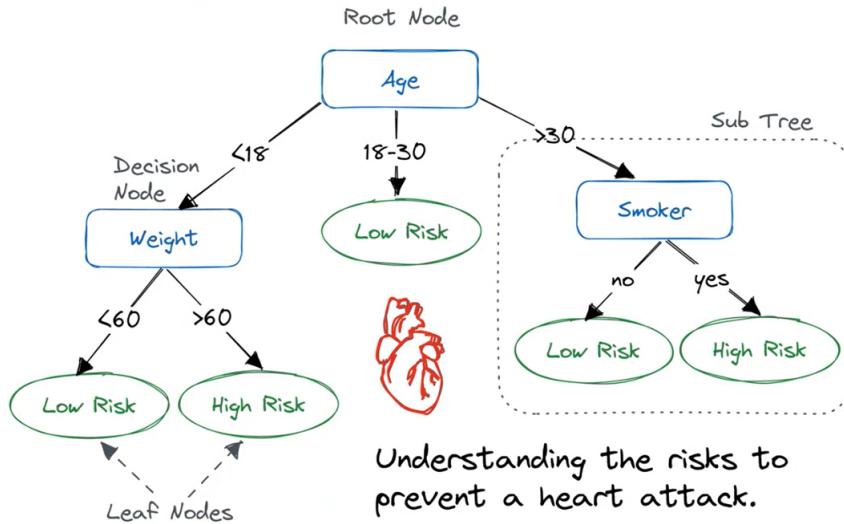
Un esempio pratico è il modello SVM, che sfrutta il concetto di *support vector* per separare linearmente i dati. Entrando nello specifico (ma non troppo) si tratta di *large margin classification*: immagina il classificatore SVM come un modello che cerca di creare la più grande zona di separazione tra le due classi in una distribuzione di dati, usando delle rette parallele.



**Attenzione!** La logistic regression è un classificatore binario che non fa rette, ma semplicemente calcola la probabilità che una classe sia 1 rispetto alla probabilità che sia 0 ed è diverso dalla linear regression. (*Il prof bene o male ha detto solo questo sulla logistic regression.*)

### 3.5.2 Decision Tree

Uno dei modelli più versatili in ML, il Decision Tree (o albero decisionale) può essere utilizzato per classificazione e regressione, ma anche in casi con multiple outputs. Sono algoritmi molto potenti che possono essere sfruttati per dataset complessi.



Uno dei suoi punti di forza è la sua leggibilità: un altro aspetto molto ricercato nei modelli di ML è la facilità di comprensione del modello.

Il Binary Decision Tree è sicuramente uno dei più semplici da leggere e viene detto *white box model*, ovvero modello a "scatola bianca", indicando l'opposto di una scatola nera. La problematica dell'interpretabilità nel ML è ancora oggi affrontata e ha l'obiettivo di capire sempre meglio il tipo di ragionamento fatto dai modelli e tradurlo in qualcosa che l'umano possa capire; si ha questa necessità per influenzare sempre meglio le scelte che vengono fatte dagli algoritmi, soprattutto influenzarli per una questione di *fairness*.

Nel Decision Tree si ha un Root node iniziale, da cui partono i primi rami, che arrivano a degli altri nodes (se a loro volta hanno delle diramazioni) o a delle leaves (o foglie, se sono nodi terminali). E sicuramente un altro pro dei Decision Trees è che non richiedono una grossa *data preparation*.

**Esempio di lettura.** Partendo dall'immagine qui sopra, cerchiamo di capire il ragionamento di un modello ad alberi decisionali. Si vuole prevedere il rischio di un paziente di avere un infarto (Sì lo so siamo tornati su questo brutto esempio).

L'albero parte da una feature, Age:

*In quale gap di età si trova il paziente?*

E genera tre leaf nodes:  $<18$ , 18-30,  $>30$ .

Poi scopre dal dataset che la maggior parte delle persone con età compresa tra 18 e 30 anni, sono a basso rischio, a prescindere da altri fattori (che in letteratura vengono chiamati feature).

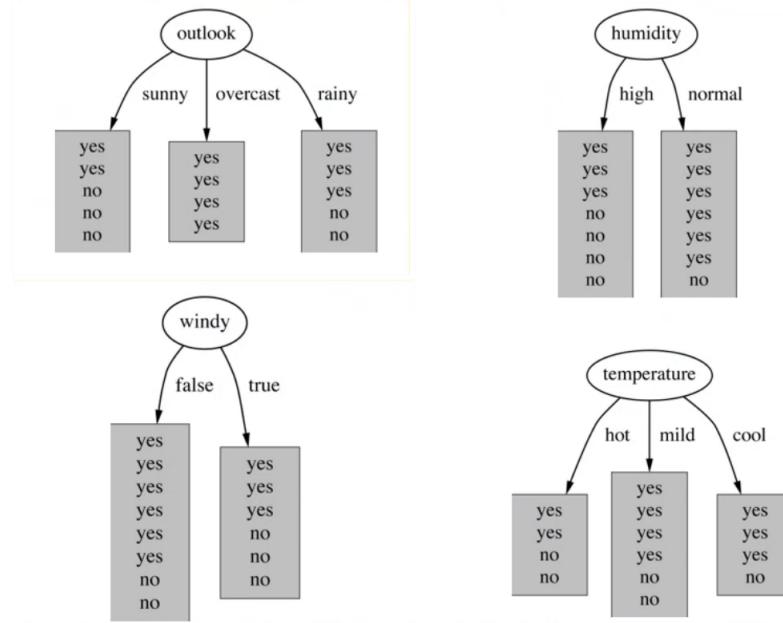
Analizzando in età minore di 18, nota che c'è una feature che fa pendere l'ago della bilancia: Weight (peso): crea quindi due leaf nodes con rischio alto e basso sulla base di tale valore.

Discorso analogo si fa per le persone con più di 30 anni dove però la feature critica è Smoker.

**N.B. Decision Tree può essere anche letto come un insieme di regole, ognuna che ci porta una risposta.**

### 3.5.3 Costruzione di un Decision Tree

Per costruire un Decision Tree, scegliamo il primo nodo e dividiamo le istanze di quell'attributo. Come si sceglie l'attributo di partenza?



Tra questi attributi, la scelta ottimale è Outlook. Questo perchè è l'unico a darci l'informazione migliore su una delle sue istanze: overcast. Overcast ha una probabilità del 100%. Ogni volta che c'è tempo nuvoloso, si gioca a calcetto. Non è necessario pruning o altre operazioni sull'albero, quella è già una foglia, a differenza degli altri che si dirameranno. Ma quando creo un albero decisionale devo formarlo il più efficiente possibile: ogni nodo deve massimizzare l'information gain.

Questo ragionamento, viene tradotto in linguaggio matematico nel calcolo dell'entropia per ogni attributo:

- Entropia del nodo Outlook, data dalla somma tra le entropie (Sunny, Overcast, Rainy):

$$\text{info}_{\text{Sunny}} = \text{entropy}(2/5, 3/5) = -\frac{2}{5}\log(\frac{2}{5}) - \frac{3}{5}\log(\frac{3}{5}) = 0,971\text{bits}$$

$$\text{info}_{\text{Overcast}} = \text{entropy}(4, 0) = -\frac{4}{4}\log(\frac{4}{4}) - \frac{0}{4}\log(\frac{0}{4}) = 0$$

$$\text{info}_{\text{Rainy}} = \text{entropy}(3/5, 2/5) = -\frac{3}{5}\log(\frac{3}{5}) - \frac{2}{5}\log(\frac{2}{5}) = 0,971\text{bits}$$

- L'informazione attesa è:

$$\text{info}[(2, 3), (4, 0), (3, 2)] = (\frac{5}{14}) \times 0,971 + (\frac{4}{14}) \times 0 + (\frac{5}{14}) \times 0,971 = 0,693\text{bits}$$

- Mentre l'info gain effettiva di Outlook è:

$$\text{gain}(\text{Outlook}) = \text{info}([9, 5]) - \text{info}([2, 3], [4, 0], [2, 3]) = 0,247\text{bits}$$

Il risultato è che facendolo con anche gli altri, il valore con info gain più elevato è proprio Outlook. Per costruire l'albero, itero questo procedimento negli attributi sotto.

### 3.5.4 Simple Algorithm

Come trasformo il Decision Tree in una regola? Esiste una procedura iterativa, dove cerchiamo di massimizzare l'accuracy della stessa. Tenendo in considerazione il numero di istanze a cui si applica la regola e al numero di quelli a cui si applica correttamente e massimizzando il loro rapporto.

**Esempio.** Immaginiamo di avere un dataset con i dati sulla vista di alcune persone. La regola? Vogliamo capire qual è il tipo corretto di lenti per una persona non vedente. Sappiamo diverse sue caratteristiche fisiche. Potenzialmente, una regola potrebbe essere "Altezza > 1.80". Andiamo a contare nel dataset per quante persone servono le lenti, che sono sopra a l'1.80, e si fa il rapporto tra casi positivi / casi considerati dalla regola. Potremmo avere 4 persone con necessità su 45 considerate. La regola sarebbe molto debole.

<code>Age = Young</code>	<code>2/8</code>
<code>Age = Pre-presbyopic</code>	<code>1/8</code>
<code>Age = Presbyopic</code>	<code>1/8</code>
<code>Spectacle prescription = Myope</code>	<code>3/12</code>
<code>Spectacle prescription = Hypermetrope</code>	<code>1/12</code>
<code>Astigmatism = no</code>	<code>0/12</code>
<code>Astigmatism = yes</code>	<code>4/12</code>
<code>Tear production rate = Reduced</code>	<code>0/12</code>
<code>Tear production rate = Normal</code>	<code>4/12</code>

Cosa prendiamo? La regola che massimizza il rapporto. Quindi prenderemo Tear production rate = normal oppure astigmatism = yes, e ne sceglio una delle due (hanno lo stesso valore). Andiamo avanti come nel Decision Tree con le altre righe del dataset e le nuove regole e le combino tutte. Però non devo renderlo troppo specifico, altrimenti non generalizza!

## 3.6 Overfitting

Uno dei problemi più importanti dell'intero ML e consiste in un modello che performa bene sui dati di training, ma non è capace di generalizzare, sbagliando completamente nel test. Questo succede per dataset troppo specifici, o per noise all'interno di essi che il modello interpreta come pattern e impara. Il modello in sè non è capace di distinguere questi noises.

**Esempio.** Immaginiamo di allenare un modello a prevedere da un dataset, l'altezza di un cittadino cinese dalle sue caratteristiche. Il test però lo eseguiamo su un dataset di persone provenienti dalla Svezia. Sicuramente, l'accuracy sarà bassissima, ma questo è dovuto non al modello scelto, ma alla scelta del dataset su cui effettuare train e test.

**Quindi come lo evito?** Basta *discretizzare* l'intervallo delle temperature in insiemi, generalizzando.

### 3.6.1 i.i.d. assumption per Statistical Modeling

Abbiamo due particolari assunzioni da fare nel caso dello statistical modeling:

- tutti gli attributi sono ugualmente importanti (identically distributed)
- tutti gli attributi sono statisticamente indipendenti (independent distributed)

Ma sappiamo che la seconda è impossibile da garantire, basta guardare il dataset metereologico e pensare che gli attributi di tempo e umidità sono dipendenti.

### 3.7 Instance based learning

Viene descritta come la più semplice forma di apprendimento: non si impara nulla, si fa tutto a memoria, con una funzione di istanza (quanto l'elemento A è simile a B o C ecc..) un esempio sono gli algoritmi simili al nearest neighbours.

Sono una serie di modelli molto utilizzati per fare profilazione e cercare le preferenze di un utente. La tipologia più conosciuta di instance learning è il nearest neighbors, un modello dove cerco "tutte le istanze più vicine". Ci sono alcuni accorgimenti da fare per poterlo utilizzare, uno di essi è la necessità di normalizzare il dataset altrimenti overfitta.

### 3.8 One rule algorithm

Inventato nel '94, è l'algoritmo più banale di tutti. Da un dataset, estrae una sola regola. L'idea alla base è molto semplice, tuttavia non è da sottovalutare: spesso le idee semplici sono quelle più efficaci.

Outlook	Temp	Humidity	Windy	Play	Attribute	Rules	Errors	Total errors
Sunny	Hot	High	False	No	Outlook	Sunny → No	2/5	4/14
Sunny	Hot	High	True	No		Overcast → Yes	0/4	
Overcast	Hot	High	False	Yes		Rainy → Yes	2/5	
Rainy	Mild	High	False	Yes	Temp	Hot → No*	2/4	5/14
Rainy	Cool	Normal	False	Yes		Mild → Yes	2/6	
Rainy	Cool	Normal	True	No		Cool → Yes	1/4	
Overcast	Cool	Normal	True	Yes	Humidity	High → No	3/7	4/14
Sunny	Mild	High	False	No		Normal → Yes	1/7	
Sunny	Cool	Normal	False	Yes	Windy	False → Yes	2/8	5/14
Rainy	Mild	Normal	False	Yes		True → No*	3/6	
Sunny	Mild	Normal	True	Yes				
Overcast	Mild	High	True	Yes				
Overcast	Hot	Normal	False	Yes				
Rainy	Mild	High	True	No				

\* Random choice between two equally likely outcomes

Analizziamo questo dataset e il potenziale risultato del one rule. Immaginiamo di dover prevedere se giocare a calcetto, sulla base del meteo; contiamo per ogni l'attributo **Outlook** i valori distinti di play. Ad esempio, per **Outlook = Sunny** notiamo che è No 3 volte su 5, il che significa che commette 2 errori su 5. E li contiamo per ogni valore di ogni feature. Alla fine prendiamo come rule la feature su cui commettiamo meno errori totali: in questo esempio, una a scelta tra **Humidity** e **Outlook**.

**Quale errore commette il one rule?** Il motivo per cui è poco utilizzato, risiede proprio nella sua semplicità. Seguendo il suo modus operandi, è utile notare che: per valori numerici ad intervalli grandi, diventa impossibile contare gli errori. Ad esempio, se aggiungessimo l'attributo temperatura, con tanti valori di temperature in un intervallo compreso tra i 15 e i 25 gradi, lui imparerebbe una rappresentazione troppo dettagliata, causando *overfitting*.

### 3.9 Naive Bayes Classifier

Come funziona un classificatore probabilistico? Calcola la probabilità che un evento si verifichi, basandosi sulla distribuzione di probabilità del dataset di training.

Outlook		Temperature		Humidity		Windy		Play			
	Yes	No		Yes	No		Yes	No		Yes	No
Sunny	2	3	Hot	2	2	High	3	4	False	6	2
Overcast	4	0	Mild	4	2	Normal	6	1	True	3	3
Rainy	3	2	Cool	3	1						
Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5	False	6/9	2/5
Overcast	4/9	0/5	Mild	4/9	2/5	Normal	6/9	1/5	True	3/9	3/5
Rainy	3/9	2/5	Cool	3/9	1/5						

Tornando alla tabella del calcetto, calcolo la probabilità di fare un calcetto per ogni valore di ogni attributo.

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

Likelihood of the two classes

For "yes" =  $2/9 \times 3/9 \times 3/9 \times 9/14 = 0.0053$

For "no" =  $3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = 0.0206$

Conversion into a probability by normalization:

$P(\text{"yes"}) = 0.0053 / (0.0053 + 0.0206) = 0.205$

$P(\text{"no"}) = 0.0206 / (0.0053 + 0.0206) = 0.795$

Osservando il nuovo giorno e sulla base dei dati di training è possibile calcolare la probabilità che si faccia il calcetto o no. Utilizzo la probabilità che sia sì, e quella che sia no, e ricavo la likelihood facendo:

$$p_{newday}(Yes) = \frac{p(Sunny)}{p(Yes)} + \frac{p(Cool)}{p(Yes)} + \frac{p(High)}{p(Yes)} + \frac{p(True)}{p(Yes)}$$

$$p_{newday}(No) = \frac{p(Sunny)}{p(No)} + \frac{p(Cool)}{p(No)} + \frac{p(High)}{p(No)} + \frac{p(True)}{p(No)}$$

Le normalizzo a uno:

$$p(Yes) = \frac{p_{newday}(Yes)}{p_{newday}(Yes) + p_{newday}(No)}$$

$$p(No) = \frac{p_{newday}(No)}{p_{newday}(Yes) + p_{newday}(No)}$$

E ottengo lo score delle due risposte. Eseguo quindi una classificazione probabilistica.

### 3.9.1 Regola di Bayes

$$Pr[H|E] = \frac{Pr[E|H]Pr[H]}{Pr[E]}$$

Questa regola viene descritta nel modo seguente:

*La probabilità di un evento H, data un'evidenza E, è uguale alla probabilità dell'evidenza dato un evento, moltiplicata per la probabilità dell'evento e diviso la probabilità dell'evidenza.*

Che spiegato in termini semplici, ritornando quindi all'esempio del calcetto:

- L'evidenza sarebbe la nuova riga di ds da classificare.
- L'evento sarebbe: si gioca o non si gioca?
- Quindi, il primo termine nella frazione sarebbe la probabilità che ci siano certe condizioni quando si verifica l'evento: se la nuova riga da classificare dice che c'è soleggiato, umido, una certa temperatura... quel termine indica la probabilità che se si gioca a calcetto, ci siano esattamente quelle condizioni, che è diverso dal chiedersi se con quelle condizioni si gioca a calcetto.
- Questo termine è poi moltiplicato per la probabilità che si giochi (ricavabile dai dati di training)
- Il termine al denominatore è l'unica incognita in questa formula. Consiste nella probabilità a priori dell'evidenza. Ovvero, la probabilità che si verifichi una certa giornata nel nostro caso (ovvero che ci sia una giornata soleggiata, con una certa temperatura ecc...). Valore non calcolabile perché ci servirebbero tutti i valori delle giornate del mondo!
- La soluzione a questo problema consiste nell'evitare il denominatore. Alla fine, è un valore per cui vengono divise entrambe le probabilità: sia che l'evento H si verifichi, sia che non si verifichi, il valore al numeratore cambia ma il denominatore no: lo consideriamo un valore inutile e non lo ricaviamo.

Il classificatore Bayesiano sfrutta questo concetto. Il Naive Bayes invece va oltre:

- Andando a pescare tra le i.i.d. assumption, considera le feature indipendenti tra loro, quindi il primo termine al numeratore, lo scomponete nella produttoria tra tutte le probabilità delle evidenze dato H. Considerando le feature indipendenti, consideriamo la probabilità dell'evidenza E dato l'evento H come l'unione delle probabilità di eventi indipendenti, ovvero la produttoria dei singoli elementi disgiunti. Il numeratore diventa quindi

$$Pr[H|E] = \frac{Pr[E_1|H]Pr[E_2|H]Pr[E_3|H]\dots Pr[E_n|H]Pr[H]}{Pr[E]}$$

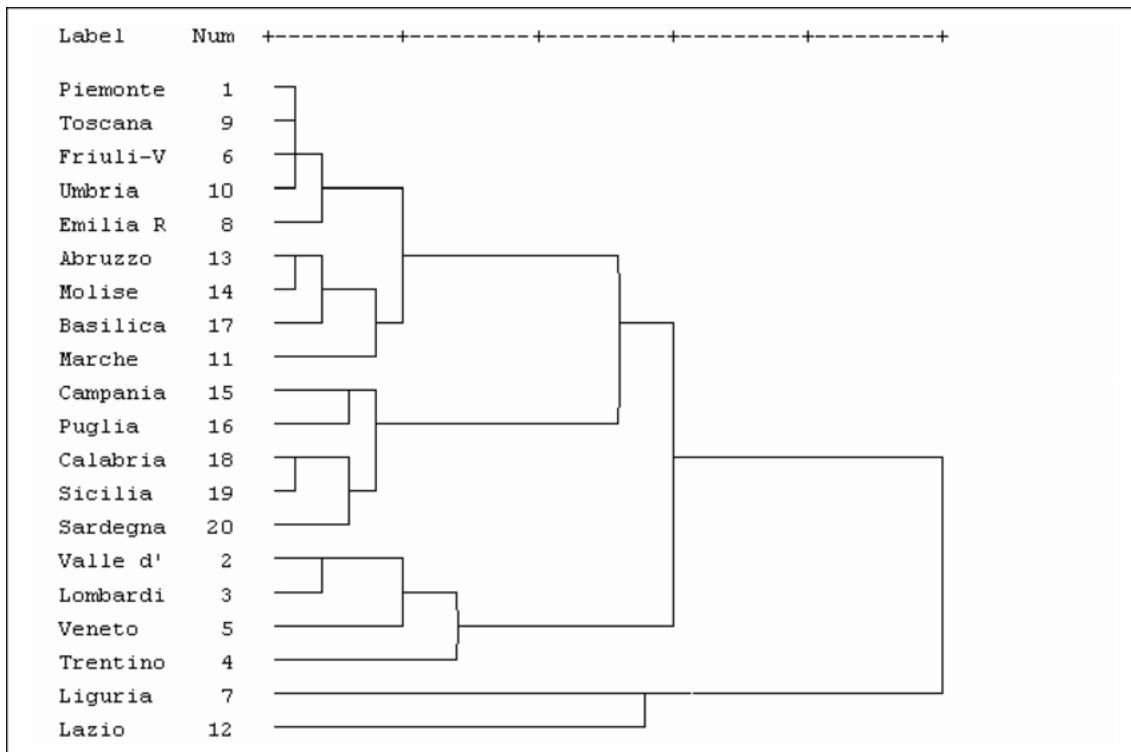
**Qual è il problema di questo approccio?** Che se la probabilità di un'evidenza dato l'evento è 0, mi porta tutto a 0.

**Come posso evitare questo problema?** Aggiungo 1 al conteggio per ogni combinazione valore attributo-classe.

## 3.10 Unsupervised Learning

### 3.10.1 Clustering

Si parla di learning unsupervised quando non si utilizzano label. Quindi non è nota la classe da predire. Quello che avviene nel clustering quindi è una divisione naturale delle istanze in gruppi. Come funziona l'algoritmo gerarchico? Finchè non ci fermiamo, cerchiamo i migliori due cluster da unire e li uniamo. Altrimenti esiste anche probabilistico: basta trovare la probabilità che un certo punto appartenga al cluster  $i$ -esimo. Come si trova la distanza tra due cluster? Sfrutti i loro centroidi (sono punti medi) e cerchi quelli più vicini. Possiamo capire quanti cluster creare, basta organizzare un dendrogramma e separarlo dove avviene la separazione principale.



### 3.11 Come trovo un modello che non overfitta?

Per calcolare l'overfitting del modello, separiamo il dataset in due parti, training e test.

Per vedere una misura dell'overfitting, basta guardare la misura dell'errore, sia nella simulazione reale che durante la fase di training. Per capire effettivamente quando un modello overfitta, è necessario guardare la sua complessità: più un modello è complesso, più *tenderà ad overfittare*.

Il Decision tree è un modello che tende ad overfittare: se immaginiamo un albero molto dettagliato, con tanti rami e nodi che descrivano ogni possibile feature del dataset, allora overfitteremo sul training set.

### 3.12 Come posso aiutare il modello contro l'overfitting?

Si può suddividere il dataset in tre parti: train, validation, e test set. Il validation è utilizzato per un test prima del test, poi lo traino nuovamente su anche quello. In questo modo separo correttamente training e testing ma soprattutto posso usare i validation data per aggiornare i parametri di learning. Esiste un altro metodo chiamato comunemente **Cross Validation**. Che spiegato molto brevemente, consiste nella suddivisione del dataset in k subsets grandi uguali. A turno, uno solo viene usato per il testing, gli altri per il training. Anche questa tipologia di struttura per la suddivisione del dataset viene utilizzata per fare il tuning dei parametri del modello.

### 3.13 Valutazione del modello

Come si misura l'errore in un modello di ML? Si usa una matrice chiamata *Confusion Matrix* o matrice di confusione; essa mette a confronto le predizioni corrette e quelle sbagliate, toccando il tema dei falsi positivi e falsi negativi, distinguendo tra i due tipi di errore.

		Predicted class	
		Yes	No
Actual class	Yes	True positive	False negative
	No	False positive	True negative

Sulla base di questa tabella, possiamo definire due parametri:

$$TP_{rate} = \frac{TP}{TP+FN}$$

$$FN_{rate} = \frac{FN}{TN+FP}$$

Che rappresentano i rate dei veri positivi e dei falsi negativi. Quindi sarebbero l'accuracy sul Sì, e l'altro è 1 - l'accuracy sulla predizione di No.

Attenzione! Un errore di falso positivo e di falso negativo hanno pesi diversi, e dipendono molto dall'ambiente in cui ci troviamo.

Altre misure che posso ricavare dalla confusion matrix sono la *precision* e la *recall*:

$$precision = \frac{TP}{TP+FP}$$

$$recall = \frac{TP}{TP+FN}$$

La precision è un valutare quante di quelle che sono positive, lo sono effettivamente. Non è sufficiente, serve anche la recall che misura quali sono quelli effettivamente scoperti come positivi?

### 3.13.1 F1 score

Le due misure indicate nel punto precedente possono essere combinate in una unica:

$$F1_{score} = \frac{(\beta^2+1)PR}{\beta^2P+R}$$

Dove il parametro  $\beta$  funge da ago della bilancia: quando è  $< 1$ , prevale la precision, quando è maggiore la recall e se è uguale vengono considerate in egual modo.

### 3.13.2 Multilabel Confusion Matrix

Quello visto finora è il caso di classificazione binaria, un conto dei Sì e No. Ma se la label da predire fosse a più valori, quindi non ci trovassimo più in un caso binario, il tipo di matrice sarebbe il seguente:

		Predicted			
		urgent	normal	spam	
Actual	urgent	8	10	1	$recall_u = \frac{8}{8+10+1}$
	normal	5	60	50	$recall_n = \frac{5}{5+60+50}$
	spam	3	30	200	$recall_s = \frac{3}{3+30+200}$
		$prec_u = \frac{8}{8+5+3}$	$prec_n = \frac{60}{10+60+30}$	$prec_s = \frac{200}{1+50+200}$	

Dove precision e recall sono calcolati singolarmente per ogni valore. Questo sistema può essere utilizzato per valutare la logistic regression.

## 3.14 Ensemble learning

Normalmente i modelli visti si combinano. Ed esistono 2 modi per metterli insieme.

### 3.14.1 Bagging

Separo il dataset in *bags*. Immaginiamo di avere n bags, utilizziamo n modelli e li applico in parallelo su una sola frazione del dataset.

### 3.14.2 Boosting

Se il bagging era in parallelo, questo consiste nel mettere in serie i modelli, e quindi tutti i dati passano all'interno di un modello dopo l'altro.

# Frequent Itemsets

## 4 Discovering frequent itemsets

Ora ci occupiamo della ricerca dei *frequent itemsets*. Immaginando il caso di un supermercato, andiamo a cercare gli elementi che la gente acquista insieme.

TID	Items
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

E dopo aver trovato queste coppie frequenti, posso estrarre delle regole da essi:

- Se qualcuno compra diaper e milk, dovrebbe comprare anche beer
- Discorso analogo per bread

Ma attenzione a non pensare a regole biunivoche! Sono definite in un solo verso.

### 4.1 Market-Basket model

Il modello market-basket è utilizzato per descrivere una relazione di tipo many-many tra due tipi di oggetti. Tutti i basket hanno al loro interno degli items. Un insieme di elementi che appare **frequentemente** è un set che si ripete più volte all'interno dell'insieme dei basket. Ovvero, ci saranno n basket dove questa coppia sarà presente. Quandol è abbastanza per essere frequente? Quando  $n > s$ , dove  $s$  è un parametro chiamato *soglia di supporto*.

Questo parametro è importante perché determina la capacità di estrapolare delle regole da un dataset.

- Se  $s$  è molto basso, ovvero ad esempio la soglia minima è che  $s = 1$ , allora riuscirò a scrivere tantissime regole, ma risulteranno completamente inutili
- Se  $s$  è molto alto, avviene esattamente il contrario, potrei descrivere troppo poco il dataset e imparare poco sulle relazioni tra i dati

Un esempio di valore per  $s$  corrisponde all'1% dei valori. **Attenzione!** Soglia di supporto e supporto sono diversi. Il supporto è il numero di volte che una coppia appare nel dataset, mentre la soglia è impostata per decidere se una coppia è frequente.

#### 4.1.1 Principio di monotonicità

Regola che limita il numero di frequent itemsets:

*In un dataset qualunque, se sono presenti due elementi x e y frequent, avranno sicuramente una frequenza maggiore o uguale alla coppia (x,y)*

Ovviamente, è uguale se i due valori si ripetono sempre in tutti i basket di cui fanno parte, maggiore nel caso in cui la coppia non sia costante.

### 4.1.2 Confidence

Un modo per misurare quanto una regola sia affidabile rispetto ad un'altra è il concetto di *confidenza*.

$$\text{conf}(I \rightarrow j) = \frac{\text{support}(I \cup j)}{\text{support}(I)}$$

Definita anche come la probabilità di  $j$  dato  $I$ . La utilizzo per trovare le regole che meglio descrivono i miei dati. Infatti, prendendo solo le regole che hanno confidenza più alta, sto prendendo le regole che mi forniscono più informazioni. Un'altra misura che ci interessa mantenere alta nel caso dei frequent itemset è l'*interesse*:

$$\text{Interest}(I \rightarrow j) = |\text{conf}(I \rightarrow j) - \Pr(j)|$$

## 4.2 Representation of the Market-Basket data

Tipicamente, i market basket data sono salvati in un file, basket-by-basket: { 22, 33, 44 } ad esempio potrebbe essere la rappresentazione di un basket.

Il numero massimo di elementi che abbiamo sono le coppie, perchè abbiamo detto che per essere frequenti un insieme di elementi deve avere delle coppie frequenti. Gli algoritmi che vedremo ora funzionano solo su coppie o terne frequenti perchè non ha senso ragionare su valori non frequenti.

**Domanda da esame: se io avessi un dataset gigantesco dove ogni riga è uno scontrino, come conto le coppie frequenti?** La cosa più banale che ci viene in mente sarebbe chiedere una lista di tutti gli elementi e con un contatore, fare il conteggio. Poi con un ciclo for per le possibili coppie, terne, ecc... Ma in un mondo big data, questo approccio è impossibile. Bisogna sfruttare il concetto di monotonicità introdotto in precedenza, e lo faccio leggendo il dataset e cercando **solo** gli elementi frequenti (diminuendo il conteggio) per poi creare delle coppie da essi e vedere quelle frequenti, e così via. Con questo approccio, ad ogni passaggio, diminuisco il numero totale di elementi da leggere.

Se avessi continuato secondo la linea d'azione originale, avrei dovuto creare (con il contatore) per ogni scontrino, un intero per vedere quante volte si ripete con tutti gli altri: per ogni scontrino avrei creato dei vettori di contatori grandi GB.

Tutti gli approcci che andremo a vedere, sono caratterizzati dal numero di volte che leggo i dati.

### 4.2.1 Triangular Matrix e Triplets

Sono due tecniche che utilizziamo per fare lo storage in memoria del valore del contatore (altra operazione che ha un costo computazionale smisurato):

- **Triangular Matrix:** posso considerare il vettore dei valori associando ad ogni posizione di quel valore uno specifico riferimento ad una coppia. Sui singoletti è facile, perchè l'elemento 0 è il primo elemento e quello in posto 1000 è il 1001esimo. Per le coppie è più complesso. Come faccio con le coppie? Mi serve una funzione che mi associa ad ogni posizione, una coppia di riferimento.

$$k = (i - 1)(n - \frac{i}{2} + j - i)$$

Dove  $k$  è la posizione nel vettore,  $n$  è il numero massimo di prodotti,  $i$  e  $j$  è la coppia che voglio trovare.

*Esempio.* Immaginiamo di avere 5 elementi, da 1 a 5. Dove sarà la coppia {1, 4}?

$$k = (1 - 1)(5 - \frac{1}{2} + 4 - 1)$$

$k = 3$ . La coppia scelta è il terzo elemento del vettore. **Importante!** Dobbiamo considerare che i sia minore di  $j$  così da evitare i duplicati.

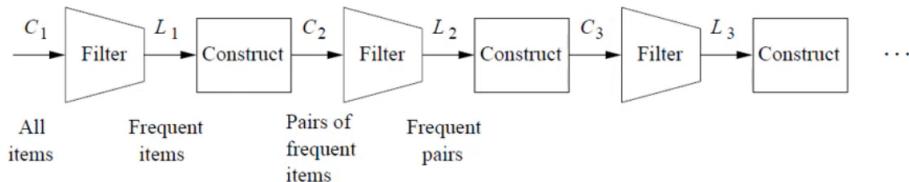
- **Triplets:** la tecnica delle triplettte consiste nella creazione di una tripletta composta da i due valori numerici e il conteggio:  $\{1, 2, 3\}$  ma è svantaggioso rispetto alla matrice poichè per una coppia salva 3 interi invece di uno solo. Ha però un vantaggio enorme sulla matrice: tratta le coppie non esistenti. Nella matrice, a prescindere che una coppia si verifichi o meno, ha un posto all'interno del vettore quindi occupo la memoria con uno 0. Nel caso invece delle triplettte, le coppie con conteggio nullo vengono scartate! Ed è quindi molto utile quando si usano dataset con poche coppie frequenti, altrimenti si preferisce la triangular matrix.

#### 4.2.2 A-Priori Algorithm

Creato per ridurre il numero di coppie da dover contare, fa due passaggi sui dati:

1. creiamo 2 tabelle, dove la prima traduce i nomi degli items in interi. Mentre l'altra tabella è un array di conteggi. Fondamentalmente il primo step è quello utilizzato per ricercare gli elementi frequenti, candidandoli
2. cerchiamo i singoletti non frequenti e li eliminiamo. Quali sono le coppie candidate? Tutte le coppie degli elementi rimanenti, per il principio di monotonicità!

E alla fine valuta quali sono le coppie frequenti dopo il secondo passaggio (e non è detto che siano tutte frequenti). Sicuramente efficace, ma è l'algoritmo più lento.



#### 4.2.3 Park, Chen, Yu Algorithm

Simile al algoritmo precedente, sfrutta la possibilità di candidare direttamente delle coppie. Quindi nel primo passaggio avviene la generazione di coppie candidate. Utilizzando una funzione di hash non facciamo lo storage delle coppie (sarebbe troppo costoso) ma associamo ad ogni coppia un valore numerico. Quali saranno le coppie frequenti? Quelle formate da singoletti frequenti e quelle in bucket frequenti.

*Esempio.* Immaginiamo di avere questa serie di elementi:

$$\{1,2,3\}, \{1,3,4\}, \{1,2\}, \{1,2,4\}, \{1,2,3,4\}, \{1,4\}$$

Creiamo  $n$  funzioni di hash, che chiameremo  $h$ , che contano le coppie in questo modo:  $h_1$ , conta le coppie con 1,  $h_2$  quelle con 2, ecc... Il risultato è che grazie alle coppie generate dalle funzioni di hash possiamo escludere le coppie non frequenti.

#### 4.2.4 Limited Pass Algorithm

Tecnica approssimata che non trova tutti gli item frequenti. Ragiona in un modo completamente diverso dagli altri algoritmi perchè estrae una parte degli items che possano stare in memoria e li estrae a sorte. Modifico la soglia di supporto, poichè con un dataset più piccolo, non posso usare ancora l'1% del dataset ma prendere l'1% del dataset più piccolo. Una volta ridotto il set, calcolo con un qualunque algoritmo visto finora. Il risultato che ottengo, impongo che sia il risultato del dataset completo, ma ovviamente lavorare solamente su un subset produrrà molti errori. Soprattutto, il problema principale saranno i falsi positivi: come posso ridurli? Faccio una seconda passata sul dataset leggendo gli elementi e verifico se quelli da me trovati sono frequenti! Elimino i non-frequent. I falsi negativi non sai di averli, e purtroppo è inutile cercare di correggerli, perchè vorrebbe dire applicare uno degli altri algoritmi... Posso pensare di ridurre i falsi negativi campionando un'altra parte del dataset e ripetendo le operazioni su di essa. Se dovessi trovare altri valori frequent che non lo erano nel primo subset, basterebbe aggiornarli.

C'è un'altra operazione, un po' meno dispendiosa: abbasso la soglia.

#### 4.2.5 Toivonen's Algorithm

Trova i risultati in queste due condizioni:

- Trova tutti i risultati, senza falsi positivi o falsi negativi
- Non genera alcuna risposta

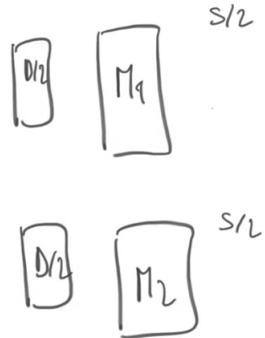
Come funziona? Sfrutta il concetto di *negative border*. Gli itemset non frequenti formati da elementi frequenti (più corretto sarebbe dire itemset non frequenti, i cui sottoinsiemi sono frequenti). Prendiamo solo una porzione, aggiorniamo la soglia, e calcoliamo gli elementi frequenti come visto nel Limited Pass. Andiamo a costruire il negative border e lo confrontiamo con il dataset nella sua interezza: se ci sono insiemi del negative border che sono frequent nel dataset, allora non produce risultato. Altrimenti il risultato ottenuto è corretto.

#### 4.2.6 SON Algorithm (Savasere, Omiecinski and Navathe)

Dividiamo l'input in chunks e faccio andare l'algoritmo su ogni chunk. Ovviamente è necessario aggiornare la soglia!

Abbiamo diviso il carico computazionale tra n macchine. Ogni macchina mi darà un insieme di elementi frequenti: paragoniamo il risultato con quello ottenuto dalle altre macchine. Possiamo dire che un elemento è frequent solo se supera la soglia anche nelle altre macchine (quindi se è frequent nelle altre).

*Esempio.* Immaginiamo di avere due chunks:



Con le relative soglie.  $M_1$  mi estrae 4 elementi frequenti  $F_1, F_4, F_7$  e  $F_9$  mentre  $M_2$  mi estrae  $F_A, F_1, F_4$ . Ora prendo l'insieme degli elementi e faccio controllare se sono frequenti in entrambe le macchine. Praticamente, passo ad  $M_1$  e  $M_2$  l'insieme di valori complessivo e conto le occorrenze in uno e nell'altro caso, se superano la soglia li posso tenere.

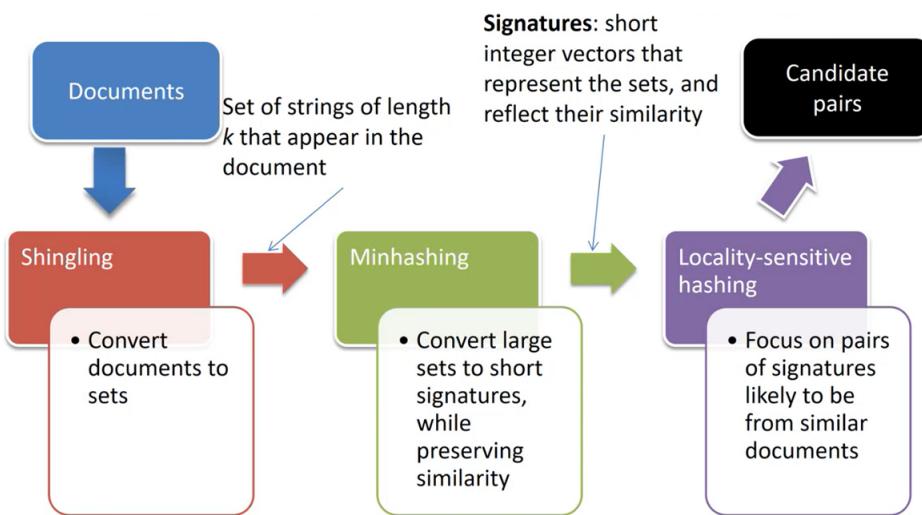
**Approccio con MapReduce.** Il primo map mi analizza un pezzo e mi trova quelli frequenti. Mi emette una coppia chiave valore con il valore frequente e una chiave associata: per esempio. potrebbe venire una cosa di questo tipo  $(F_1, 1), (F_2, 1)$ . Il reducer non fa assolutamente niente, butta fuori le coppie esattamente in questo modo per fare prima. Conviene piuttosto fare due cicli piuttosto che fare tutto in una volta.

Secondo step: altro mapreduce, mettiamo  $M_1$  e  $M_2$  ma questa volta con gli elementi frequenti: con il combiner possiamo contare quante occorrenze abbiamo in ogni macchina.

### 4.3 Finding Similar Items

Abbiamo visto come trovare gli items più frequenti, ora dobbiamo cercare di capire come trovare gli items simili. Dobbiamo trovare una misura per stabilire una similarità tra due elementi: **similarità di Jaccard**. Ossia, va analizzato un oggetto secondo un insieme di proprietà e poi è necessario guardare le proprietà che sono in comune rispetto a tutte le proprietà con cui qualifichiamo gli oggetti. Per una persona, potrebbero essere altezza, colore degli occhi, voce... Su un documento ad esempio, le parole uguali.

Perchè è utile? Per trovare documenti uguali, quindi plagiati, oppure per trovare i suggerimenti di acquisto per le persone; si vede bene la pipeline guardando la seguente immagine:



Tutti i passaggi verranno spiegati nello specifico qui di seguito.

#### 4.3.1 Shingling

Prima tecnica: convertiamo i documenti in insiemi. Indivuiamo una finestra di  $k$  caratteri chiamati *shingle*. Vado a costruire una matrice dove sulle colonne ho sempre un milione di documenti mentre sulle righe tutte le possibili combinazioni tra caratteri.

*Esempio.* Cerchiamo di capire meglio il concetto di shingle da un esempio. Frase: Oggi piove Se prendiamo  $k$  uguale a 3, otteniamo come shingle ogg, ggi, gi- ecc...

**Ma perchè utilizzare degli shingle?** Mi dà un ordine alle parole con questa finestra che scorre. E come scelgo  $k$ ? Con un  $k$  basso, tutti i documenti risultano uguali, non faccio abbastanza distinzione se guardo in finestre unitarie di singole lettere; se  $k$  è troppo alto, ottengo al contrario una ricerca troppo pignola dove trovo solo documenti esattamente uguali (considero interi discorsi in una finestra).

**Quante righe posso fare con  $k=3$ ?** Si calcolano come i caratteri che scelgo, elevati alla  $k$ :  $(caratteri)^k$  e devo fare attenzione perchè per  $k$  troppo elevati, non staranno mai in memoria.

DISCORSO MATRICCICICICIC

#### 4.3.2 Minhashing

Seconda tecnica: tecnica per mantenere la similarità con una tabella ridotta. Riduce il numero di righe mettendo per ogni vettore della tabella originale una signature o firma che mantiene la similarità.

Prendiamo il documento originale: