Big Data and Text Analysis

Contents

1 Generalities and Map Reduce **1**
- 1.1 Introduction to Distributed File Systems . . . 1
- 1.2 Map Reduce . . . 2
- 1.3 Distributed File System Architecture w/ Map Reduce . . . 3
- 1.4 MapReduce in parallelo . . . 4
- 1.5 Combiners . . . 4
- 1.6 Remarks . . . 4
- 1.7 Algoritmi che usano Map Reduce . . . 5
  - 1.7.1 Matrix-Vector multiplication - Vettore che sta in memoria . . . 5
  - 1.7.2 Matrix-Vector multiplication - Vettore che non sta in memoria . . . 6
  - 1.7.3 Relational algebra operations - Selection . . . 6
  - 1.7.4 Relational algebra operations - Projection . . . 6
  - 1.7.5 Relational algebra operations - Union . . . 6
  - 1.7.6 Relational algebra operations - Intersection . . . 6
  - 1.7.7 Relational algebra operations - Difference . . . 7
  - 1.7.8 Relational algebra operations - Natural Join . . . 7
  - 1.7.9 Relational algebra operations - Grouping and Aggregation . . . 7
  - 1.7.10 Matrix multiplication . . . 7
- 1.8 Remarks . . . 8
  - 1.8.1 Reducer size q . . . 8
  - 1.8.2 Replication rate r . . . 8
  - 1.8.3 Similarity Join . . . 8

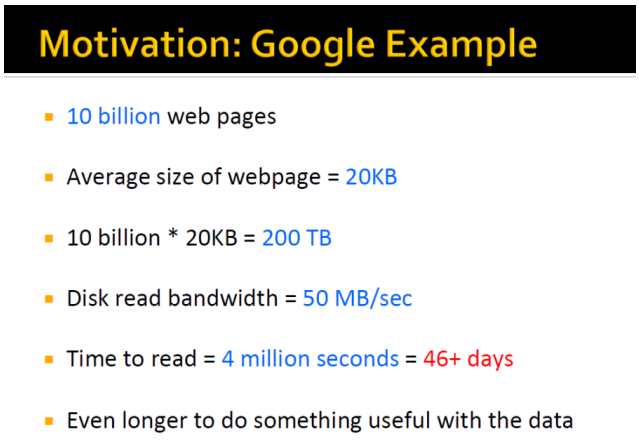
2 Data Mining **9**
- 2.1 Principio di Bonferroni . . . 9
  - 2.1.1 Esempio dei malfattori . . . 9

3 Machine Learning **10**
- 3.1 Perchè utilizzare il Machine Learning? . . . 10
- 3.2 Tipologie di apprendimento . . . 10
- 3.3 Risultato . . . 11
- 3.4 Tipologie di modelli . . . 11
- 3.5 Regressione . . . 12
  - 3.5.1 Regressione lineare . . . 12
  - 3.5.2 Decision Tree . . . 13
  - 3.5.3 Costruzione di un Decision Tree . . . 14
  - 3.5.4 Simple Algorithm . . . 15
- 3.6 Overfitting . . . 15
  - 3.6.1 i.i.d. assumption per Statistical Modeling . . . 15
- 3.7 Instance based learning . . . 16
- 3.8 One rule algorithm . . . 16
- 3.9 Naive Bayes Classifier . . . 17
  - 3.9.1 Regola di Bayes . . . 18
- 3.10 Unsupervised Learning . . . 19
  - 3.10.1 Clustering . . . 19

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**.



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 elaboro 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

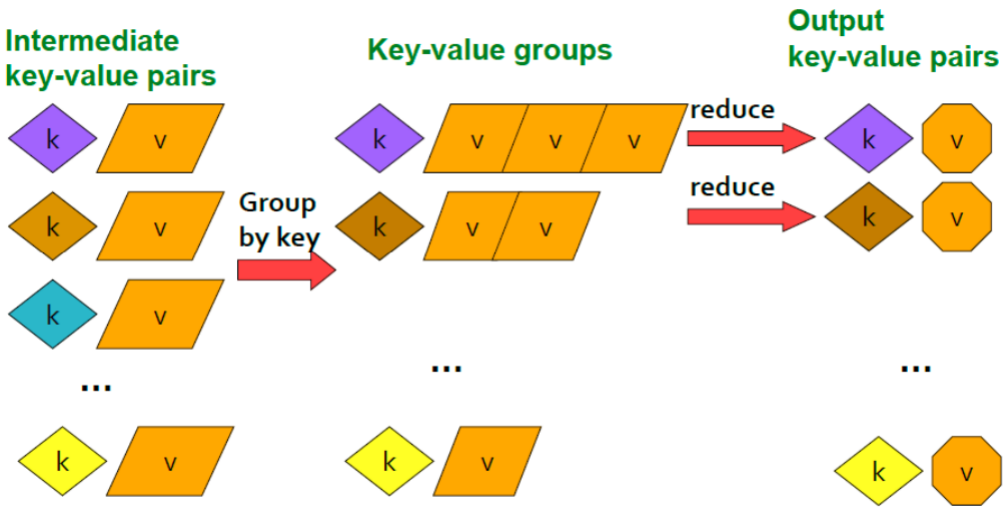
Il ”map” è il primo step dove essenzialmente vado alla ricerca delle informazioni utili. Corrisponde di fatto ad una query dove isolo elementi secondo una certa caratteristica. *Extracting something of interest*

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

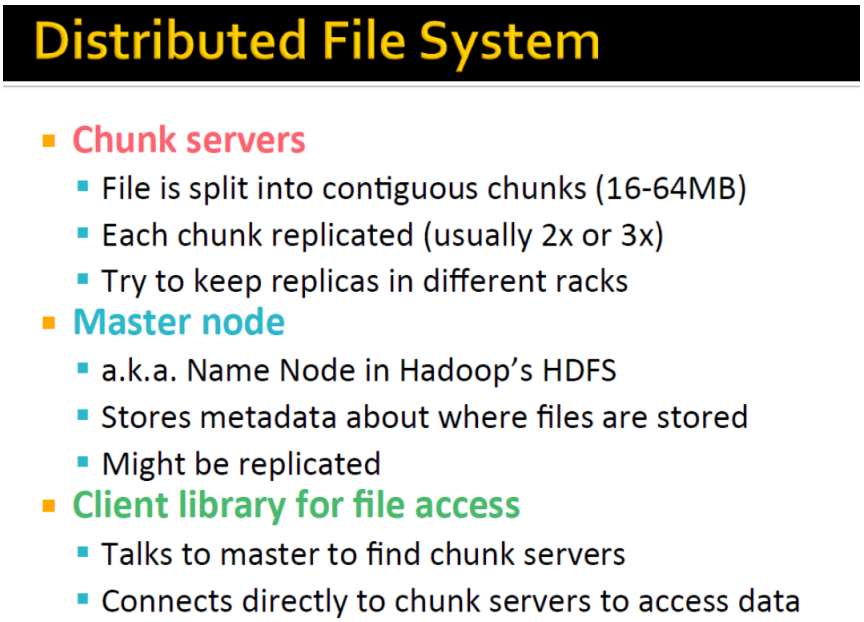
Mette insieme tutto alla fine, aggrega per avere dei dati più compatti. Risparmiando quindi memoria semplicemente associando, ad ogni elemento diverso del documento, il numero di volte che si ripete. *Aggregate, summarize, filter, transform*



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.



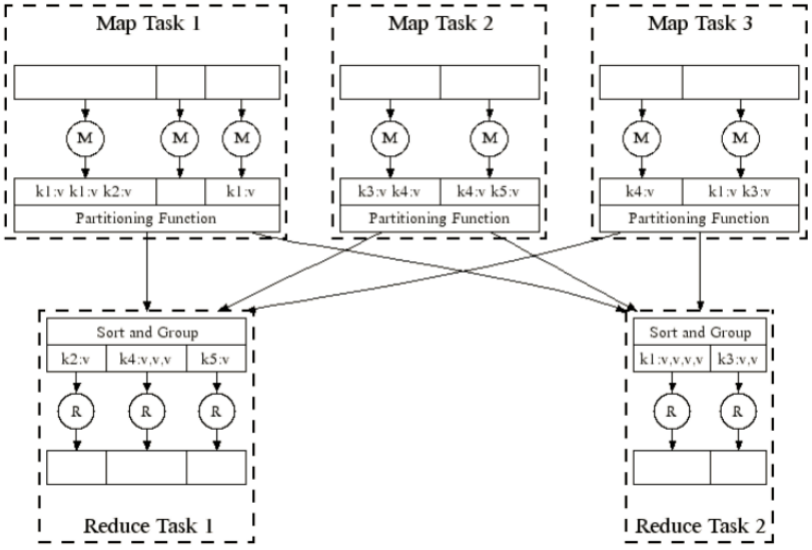
Inoltre, il Master Node controlla e pinga periodicamente i workers per trovare delle failures (malfunzionamenti).
Cosa succede se avviene un guasto? Dipende da che parte della struttura si rompe:

- 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 utilizzati 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)

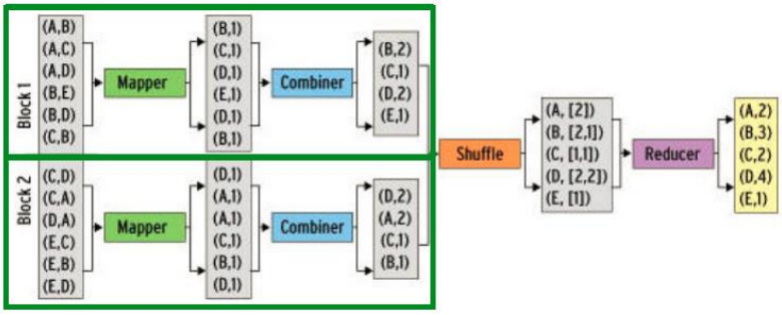
1.4 MapReduce in parallelo

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

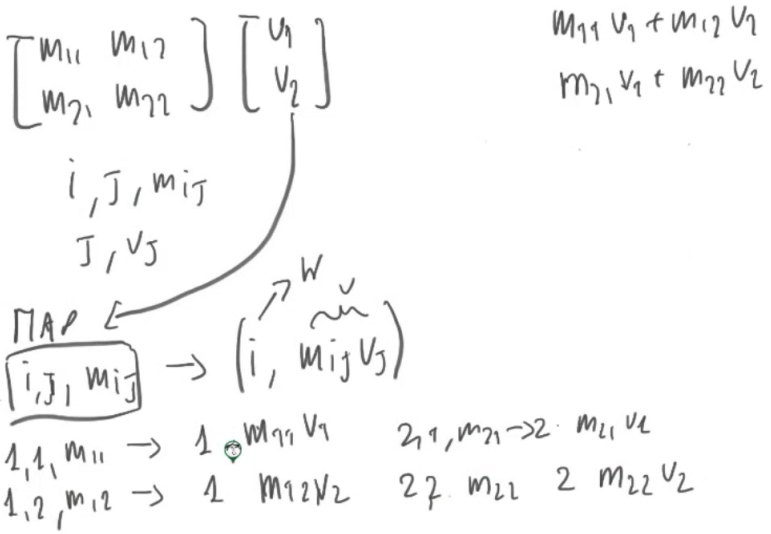
Elevata probabilità di essere chiesto. Esistono diversi algoritmi che sfruttano il map reduce, come la moltiplicazione vettoriale oppure le operazioni di algebra relazionale.

1.7.1 Matrix-Vector multiplication - Vettore che sta in memoria

Matrice M = n × 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 → 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:



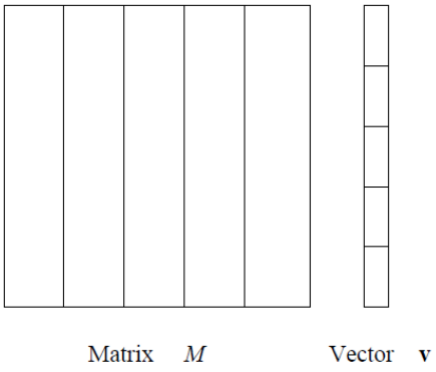
Ora abbiamo il Group By Key Task, che funziona esattamente come spiegato: mette insieme gli elementi che hanno stessa chiave. La chiave, in questo caso, è la **riga della matrice**. Quindi, dopo la Reduce Task che fa la somma e osservando sempre l'immagine, il risultato che otterremo sarà:

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

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

$R(a,b)JOINS(b,c)$
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 θ alla lista [b1,b2,...,bn] associata ad a; il risultato è una coppia (a,x) dove c è il risultato dell'aggregazione θ.
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:

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

$$\underset{4\times2 \text{ matrix}}{\begin{bmatrix} a_{11} & a_{12} \\ \cdot & \cdot \\ a_{31} & a_{32} \\ \cdot & \cdot \end{bmatrix}} \underset{2\times3 \text{ matrix}}{\begin{bmatrix} \cdot & b_{12} & b_{13} \\ \cdot & b_{22} & b_{23} \end{bmatrix}} = \underset{4\times3 \text{ matrix}}{\begin{bmatrix} \cdot & x_{12} & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & x_{33} \\ \cdot & \cdot & \cdot \end{bmatrix}}$$

$$x_{12} = (a_{11}, a_{12}) \cdot (b_{12}, b_{22}) \quad = a_{11}b_{12} + a_{12}b_{22}$$
$$x_{33} = (a_{31}, a_{32}) \cdot (b_{13}, b_{23}) \quad = a_{31}b_{13} + a_{32}b_{23}.$$

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, perchè 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, perchè 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 perchè 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

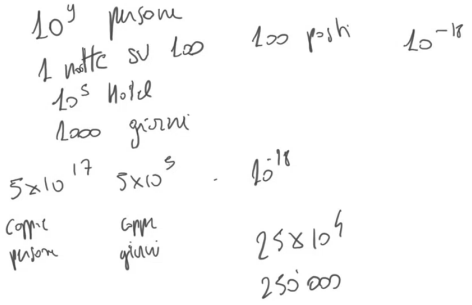
Nel Machine Learning, come vedemo prossimamente, non si scrivono istruzioni ma è la macchina che impara direttamente dai dati. Il **Data Mining** è quella parte dell'informatica che si occupa di cogliere delle relazioni tra grandi quantitativi di dati. L'assunzione che noi facciamo è che il dato dica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



Otteniamo, facendo delle stime di probabilità:

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

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

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

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

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

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

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

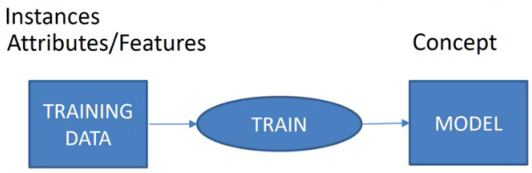
$$5 \times 10^17 \times 5 \times 10^5 \times 10^{-17} = 250'000$$

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, sopperendo 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:

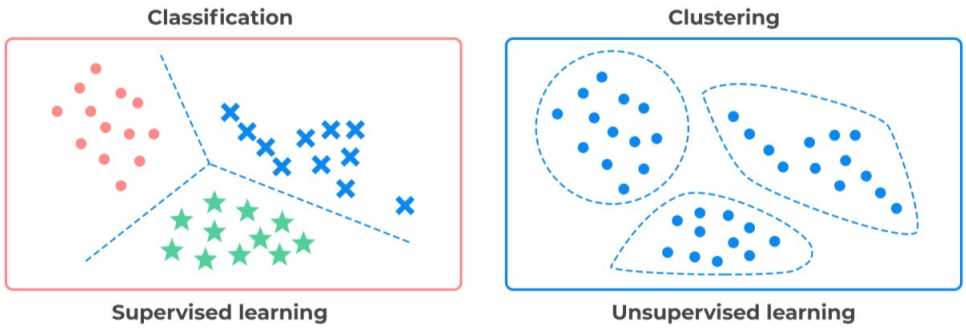
- problemi la cui soluzione richiede molto fine-tuning e un codice spropositato.
- problemi complessi per cui non è possibile utilizzare un approccio tradizionale
- enviroments dove i dati sono soggetti a cambiamenti
- ottenere informazioni sulle distribuzioni di big data

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 con il 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

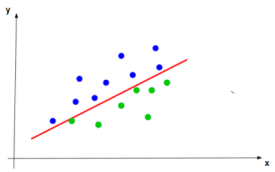
Immaginiamo di voler stimare la qualità della vita di un individuo, a partire dal PIL pro capite. Potremmo quindi indicare la $life\_satisfaction$:

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

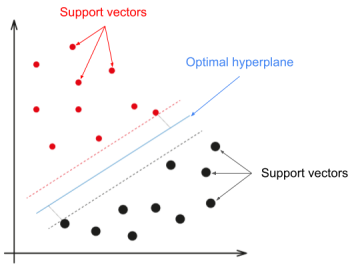
E avremmo così una relazione lineare tra i due valori (al crescere del PIL, la qualità della vita migliora). Generalmente, un modello lineare fa una prediction calcolando semplicemente una somma pesata delle input features, più una costante chiamata *bias*.

- θ_0 è il bias term, o intercept term
- θ_1 è il 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.



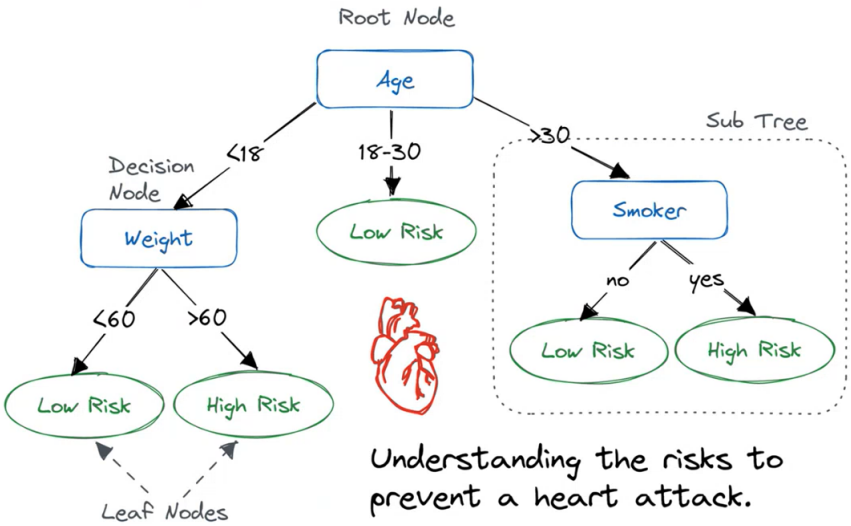
Un esempio pratico è il modello SVM, che sfrutta il concetto di *support vector* per separare linearmente i dati. Entrando nello specifico (ma non troppo) si tratta di $large\ margin\ classification$: immagina il classificatore SVM come un modello che cerca di creare la più grande zona di separazione 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 quindi 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`:

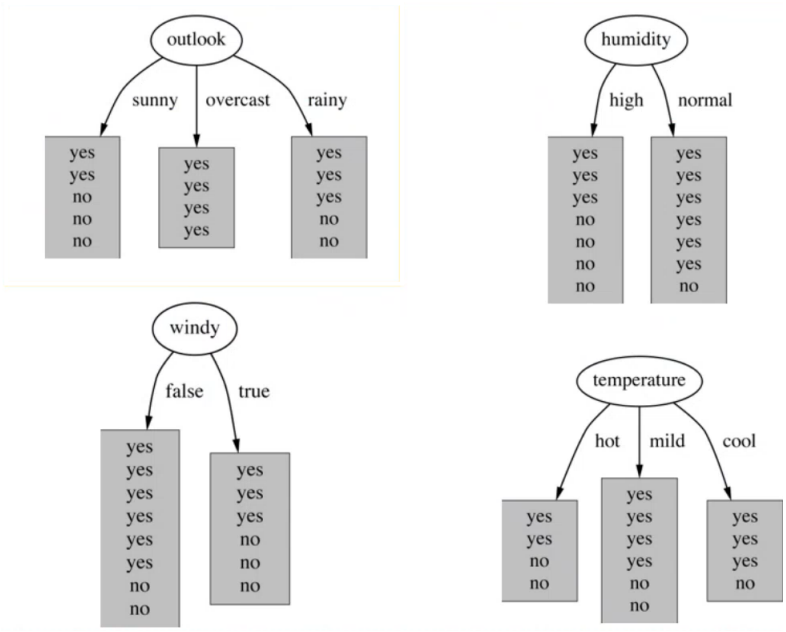
In quale gap di età si trova il paziente?

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 trodotto in linguaggio matematico nel calcolo dell'entropia per ogni attributo:

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

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

$$info_{Overcast} = entropy(4, 0) = -\frac{4}{4}log(\frac{4}{4}) - \frac{0}{4}log(\frac{0}{4}) = 0$$

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

- L'informazione attesa è:

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

- Mentre l'info gain effettiva di Outlook è:

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

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.

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

Cosa prendiamo? La regola che massimizza il rapporto. Quindi prenderemo Tear production rate = normal oppure astigmatism = yes, e ne scegliamo 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 efftuare 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
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

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

* 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*	*Yes*	*No*
Sunny	2	3	Hot	2	2	High	3	4	False	6	2	9	5
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	9/	5/
Overcast	4/9	0/5	Mild	4/9	2/5	Normal	6/9	1/5	True	3/9	3/5	14	14
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 × 3/9 × 3/9 × 3/9 × 9/14 = 0.0053

For "no" = 3/5 × 1/5 × 4/5 × 3/5 × 5/14 = 0.0206

Conversion into a probability by normalization:

P("yes") = 0.0053 / (0.0053 + 0.0206) = 0.205

P("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 perchè 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 scompone 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

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

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.