**22/09/2020**

**Big Data**: tanti dati incongruenti, sporchi, dati che eccedono la capacità di processo di un database convenzionale (molto aleatoria, dipende dalla potenza del PC su cui viene applicata, oggi tutti i PC li contengono). Dati che hanno 3 caratteristiche principali: Volume, Velocità e Varietà (+ Veracity + Value + Double V). Creano trasparenza perché accessibili in modo veloce da chiunque, sono disponibili dati su cui poter eseguire simulazioni, segmentazioni della popolazione rendendolo utile.

**Volume**: grande quantitativo di dati che impediscono di essere processati in maniera veloce, obiettivo realizzare nuovi sistemi per memorizzare dati (NoSQL, JSON, Mappe Key-Value). Oggi si utilizzano sistemi distribuiti, memorizzando dati in diversi PC, introducendo il problema della suddivisione dei dati, distribuiti in *modo orizzontale* (suddivido in ordine alfabetico su due PC, quindi so su quale PC devo cercarli) oppure in *modo verticale* (in caso di divisione della tabella per colonne).

**Velocità**: molto importante sia acquisire che processare i dati in maniera veloce se no inutili.

**Varietà**: eterogenei, diversi tra di loro, gestione di dati con formalismi diversi (Data Lakes: dati in formato nativo, in passato venivano normalizzati ma si perdeva troppo tempo).

**Market Busket Analysis**: processo di analisi di affinità che analizza le abitudini di acquisto dei clienti nella vendita al dettaglio, trovando associazioni su diversi prodotti comprati, è utile per l'adozione di strategie di marketing ad hoc.

**Pipeline**: insieme di istruzioni che ci portano dai dati grezzi ad una analisi.
- **Acquisition**: i dati sono presi da una fonte e devono essere memorizzati in qualche modo (es. sensori della macchina), quindi capire quali sono i dati che mi importano davvero, oppure devono essere acquisiti da un DB online perché non salvati nella memoria interna.
- **Extraction**: operazioni che mi consentono di elaborare i dati grezzi e li portano in un formato utile per l'analisi, perché i sensori potrebbero sbagliarsi e quindi devo essere in grado di trovare i dati che sono discostanti dal dato atteso, rimuovo quindi gli *OutLier*, quindi considerato errato, per evitare questo fenomeno si raddoppiano i sensori e vengono confrontati i dati ottenuti, e infine gestisco i *Missing Lier* e normalizzo i dati.
- **Integration**: dati provenienti da diverse tabelle quindi devo integrarli tra loro.
- **Analysis**: modello per analizzare i dati, problema di classificazione.
- **Interpretation**: controllo se il modello applicato mi da risultati in base a ciò che avevo previsto.

**Provenance**: ovvero capire da dove derivano questi dati per definire la sua qualità in base al sito da cui sono stati ricavati.

**Inconsistenza e Incompletezza**: più o meno comprensibili, non importa la loro esattezza perché non sono classificati in maniera standard e incompleti, quindi si utilizzano algoritmi per unirli.

**CrowdSourcing**: quando ci sono molte persone che aiutano a trovare un determinato dato (Amazon's Mechanical Turk, processo manuale effettuato da utenti utilizzato per controllare dei dati e classificarli, "pulire i dati").

**Scale**: lavorare sui Big Data deve essere facile anche su una scala di dati molto grandi e non ci interessa il risultato esatto ma una stima, ci interessa una stima di comparazione.

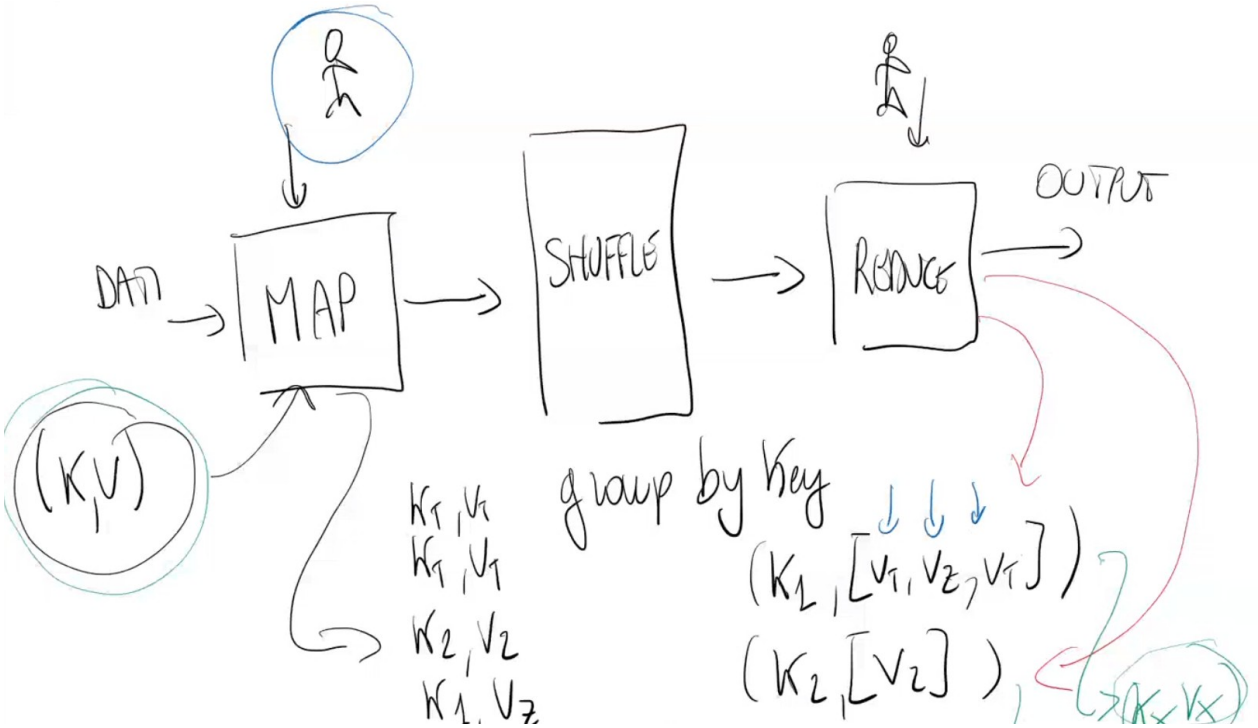
**TimeLiness**: la risposta del dato e le tecniche real-time devono essere tempestive a scapito della precisione.

**Privacy**: bisogna stare attenti a ciò che si pubblica e mantenere i dati privati, oscurandoli.
**Visualizzazione**: come devo visualizzare i dati, tecniche per visualizzare i dati in maniera corretta, facile da capire e senza generare errori, utilizzando una scala corretta. Il dato deve raccontare una storia e rappresentare qualcosa di concreto.

**Data Science**: scienza che studia i dati e la creazione di prodotti basati sui dati, il dato è il risultato del prodotto (es. suggerimenti di acquisto). Promuove la creazione di dati stessi.

**<u>24/09/2020</u>**

**Map-Reduce**: sistema/tecnica per processare i dati in un file system distribuito, software di basso livello, per questo oggi si utilizzano piattaforme che incorporano tecniche di questo tipo, quindi utilizzato in modo indiretto.
- Fase di **Map**: dove entrano i dati (input) in formato chiave-valore ed emette tante coppie chiave-valore, le chiavi possono essere duplicate;
- Fase di **Shuffle**/Group by key: fase eseguita automaticamente dal sistema, mette insieme elementi con la stessa chiave ottenendo nel campo valore un array contenente i dati della stessa chiave;
- Fase di **Reduce**: lavora a livello di singola coppia, un reduce per ogni chiave, e genera una nuova coppia chiave-valore, in un nuovo formato.



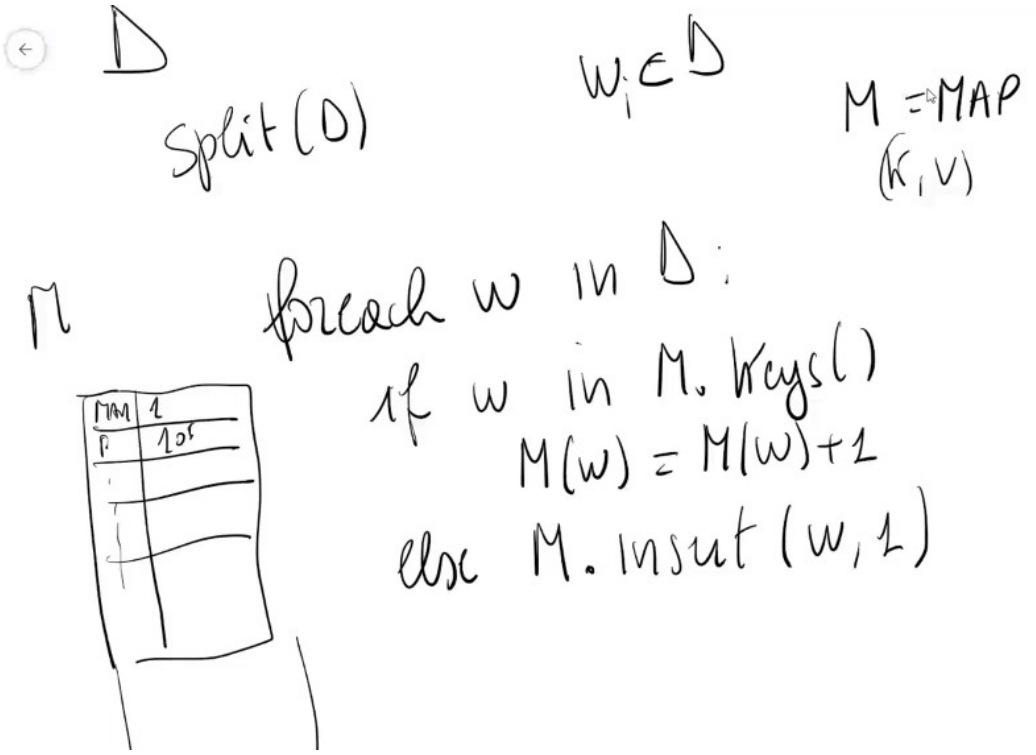

**Computer Clusters**: insieme dei computer connessi nella rete del file system distribuito, ognuno contiene dati ma i dati devono essere memorizzati in diversi dischi per permetterne l'accesso anche in caso di guasto all'interno di uno di essi, rendendolo accessibile quindi attraverso un altro switch, con tecniche di ridondanza.

**Chunks**: porzioni di dati, con dimensione prestabilita (16-64MB), replicati in diversi nodi della rete (2 o 3 volte), cercando di mantenere i replicati in diversi racks (contenitori), questo processo viene eseguito dai **Chunk servers**.

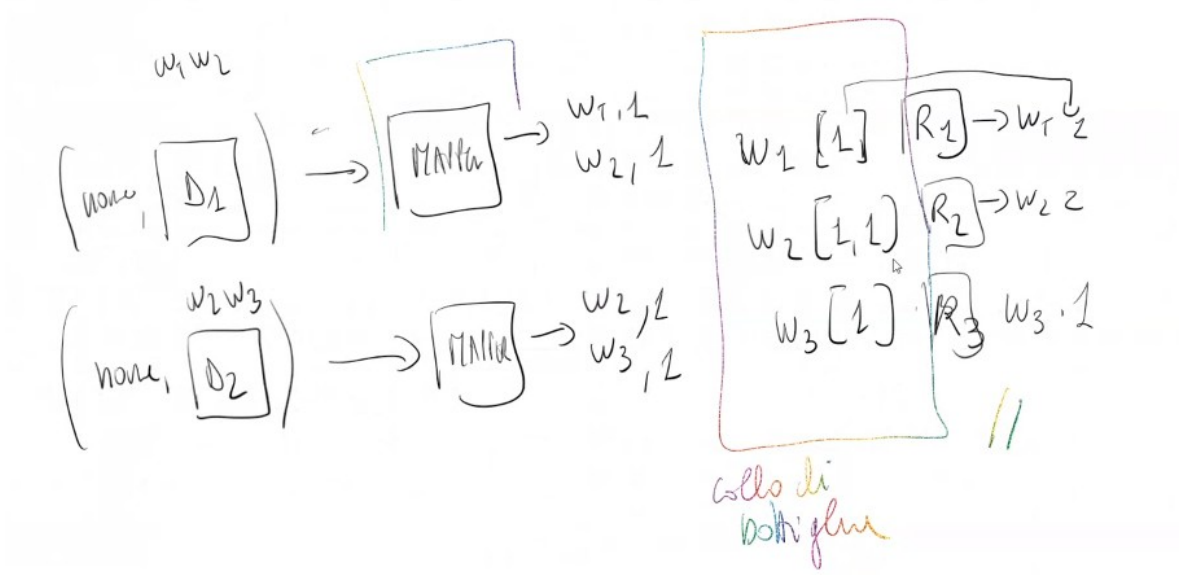
**Master node**: si preoccupa della coordinazione, nodo principale che mantiene la tabella dei dati e sa dove e in quanti chunks è suddiviso un file, anche questo nodo deve essere replicato. Si occupa

inoltre di dire quali mapper hanno finito il loro compito e infine di “pingare” i diversi server per sapere se sono funzionanti o meno.

**Esempio**: nel caso in cui devo contare le parole contenute all’interno del mio documento di testo e inserirle in un file dove è racchiuso il conteggio delle parole (di dimensioni molto grandi), andrei in contro al problema del tempo perché ogni volta che devo eseguire un confronto o una scrittura all’interno della tabella ci metterei tanto tempo proporzionalmente alle dimensioni della tabella.




Per rendere questo processo più veloce posso applicando il Map-Reduce, divido il testo in chunks e darlo in pasto a diversi PC che ne effettuano la fase di Map, successivamente li raggruppo per la fase di Shuffle (Bottleneck) dove devo eseguire il conteggio totale e infine eseguo il Reduce in parallelo, associando i processi in base al tempo che ci mettono a diversi PC.



**28/09/2020**

**Map-Reduce in Parallelo**: preleva direttamente i dati nodi di partenza che gli servono, quindi il master conosce quale riduttore gestisce quelle determinate chiavi e quindi le porta direttamente alla destinazione corretta (indirizzamento).

**Map-Task**: macchina che esegue l'operazione di Map e contenitore di uno o tanti mapper, idealmente una macchina contiene un mapper.

I **risultati intermedi** *non* vengono collocati nel filesystem distribuito in quanto ci vorrebbe troppo tempo per ridondare i dati, ma salvati in locale.

(Il Reduce interroga ogni filesystem locale dei mapper che hanno generato i dati e attraverso la chiave vengono restituiti i dati corrispondenti, ma può verificarsi anche il metodo contrario.)

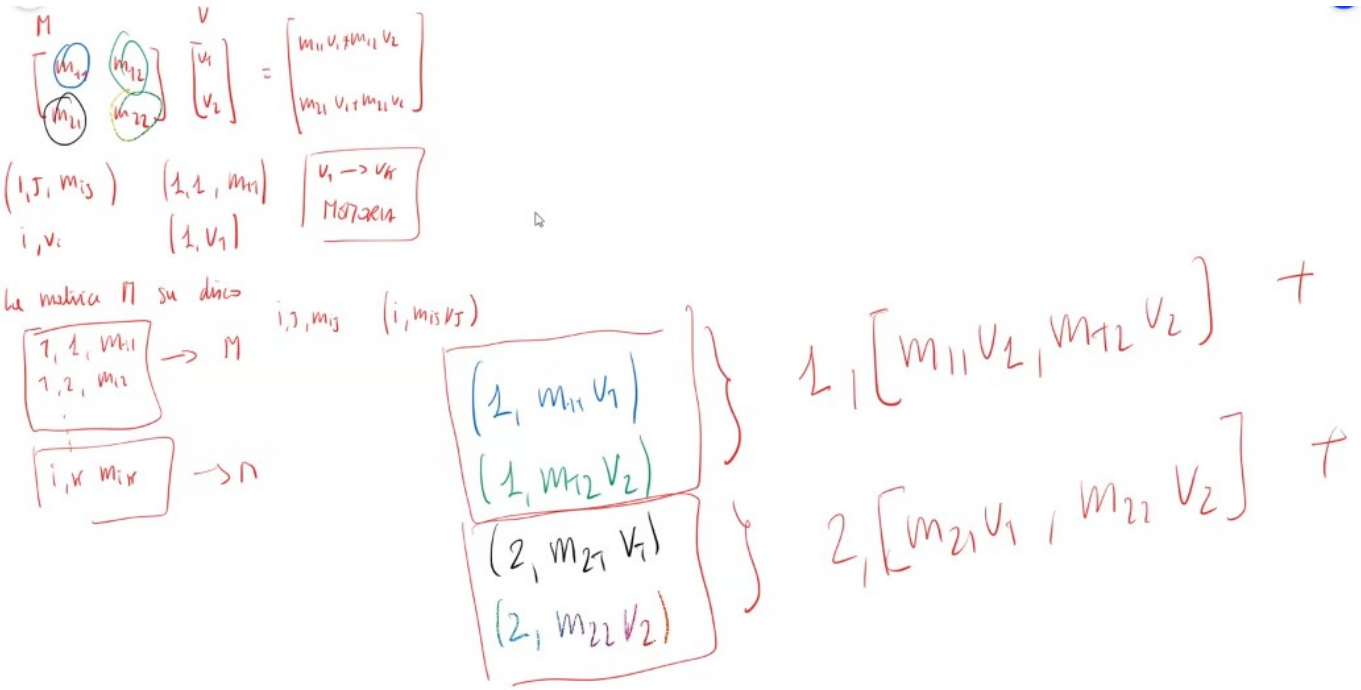
**Fallimenti**:
- **Mapper**, il master node se ne accorge e il lavoro viene assegnato ad un altro server e bisogna notificare il Reducer per sapere dove richiedere i dati quindi il nuovo reducer, oppure lo riassegna a quel server se è un problema transitorio.
- **Reducer**, solo la chiave su cui si sta eseguendo il compito è interessata e viene riavviato.
- **Master**, devo ripartire da capo perché è stato un fallimento globale.

Quando l'operazione del reduce è di tipo **Associativa** o **Commutativa** posso eseguire la stessa operazione nel mapper, permettendo di passare meno dati e rendendo più veloce il processo.

Il **Combiner** somma il valore di tutte le chiavi di un mapper, riassumendo direttamente al suo interno il compito del reducer, permettendo di trasferire meno dati e aumentando la velocità.

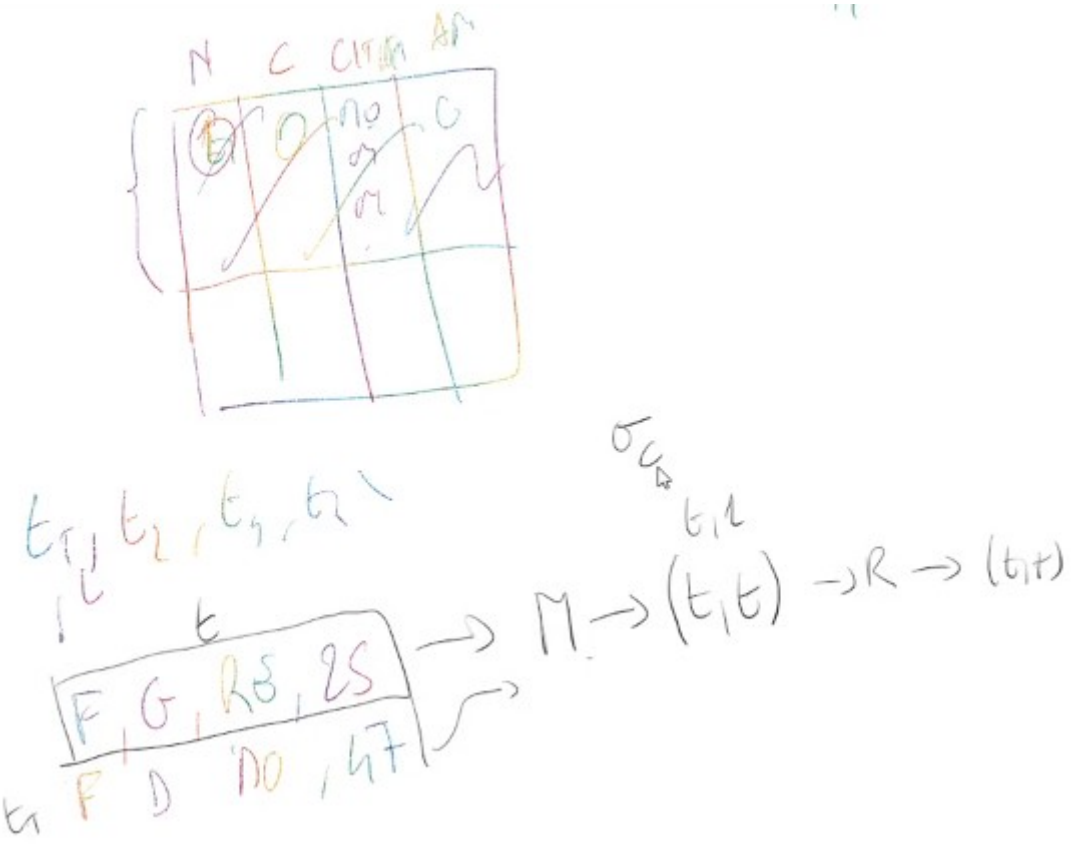
Il massimo parallelismo si ottiene quando ogni macchina di reduce contiene una sola operazione. Non sempre questo porta all'ottimizzazione, ma con tecniche di bilanciamento del carico su diverse macchine si possono ottenere risultati simili.

**Matrix-vector**: algoritmo che utilizza il map-reduce, moltiplicazione matrice per vettore.

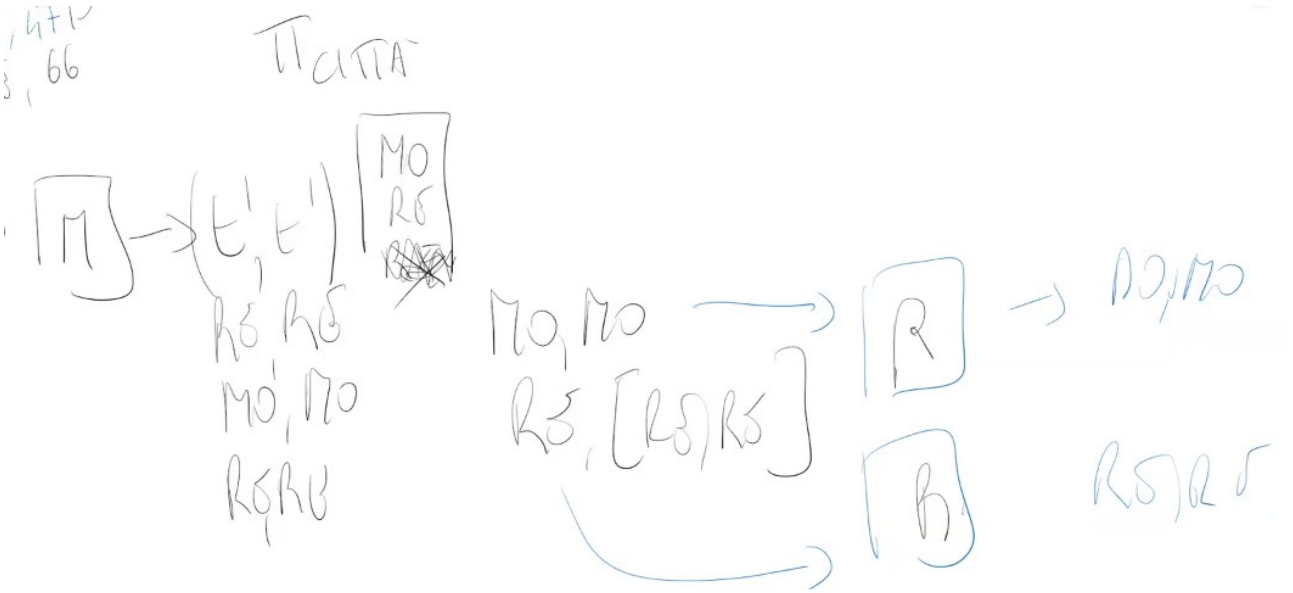


Ogni volta che leggo i valori della matrice devo leggere un vettore corrispondente ai valori da moltiplicare, questo comporta un enorme dispendio di tempo e si potrebbe incorrere nel “Trashare”. Per ovviare a questo problema posso dividere il vettore in piccoli pezzi per poterlo contenere nella memoria e stanzio un mapper per ogni pezzo di vettore.

**Operazione di Selezione**



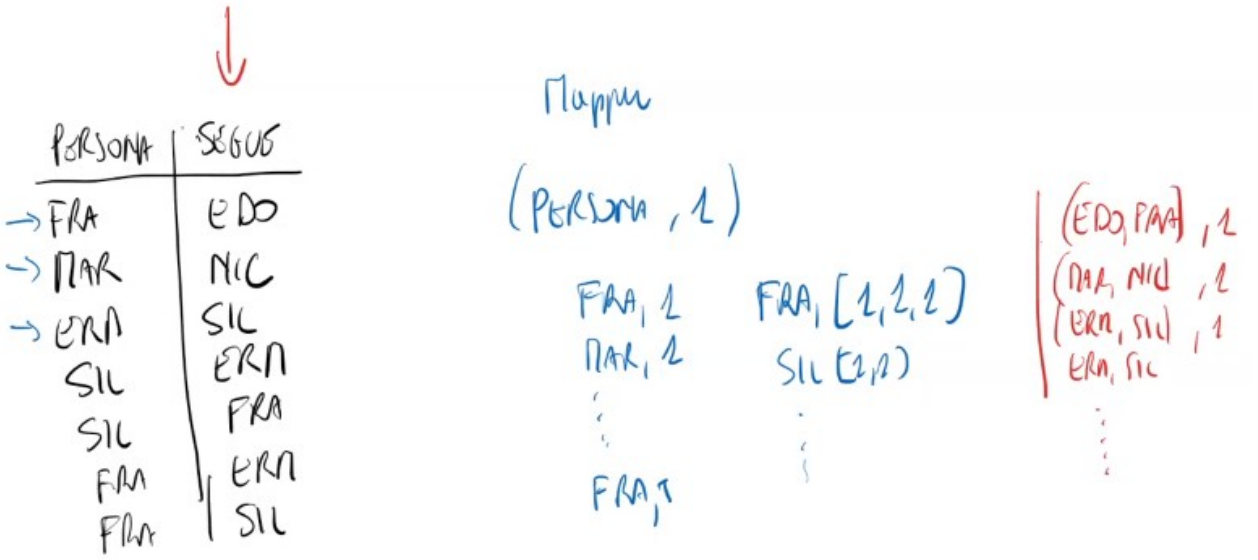
**Operazione di Proiezione**: seleziona una porzione della tabella e eliminati i duplicati.



! Tutte le volte che c’è un conteggio si può utilizzare un combiner

**29/09/2020**

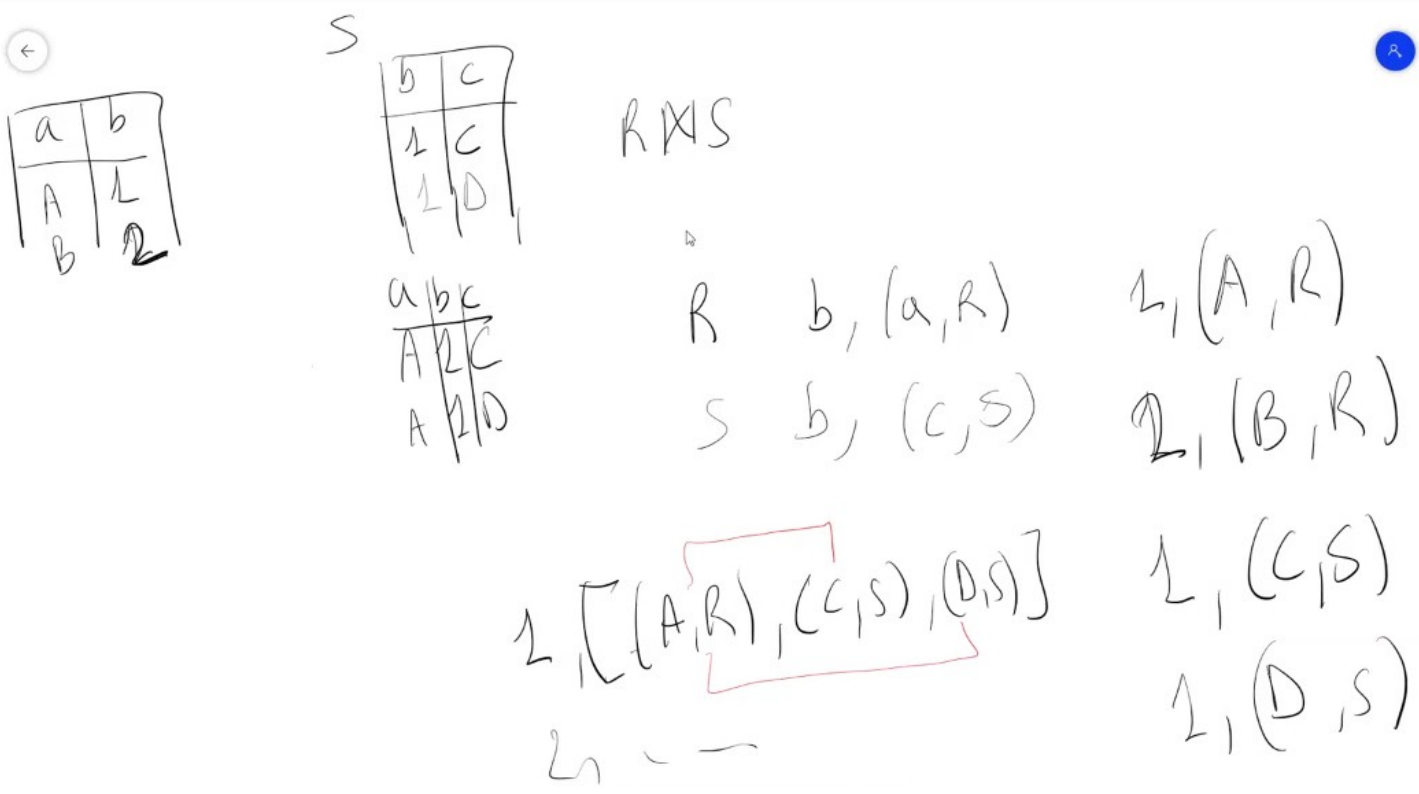
Esempio per gestire i follower (metto nella chiave direttamente la coppia)



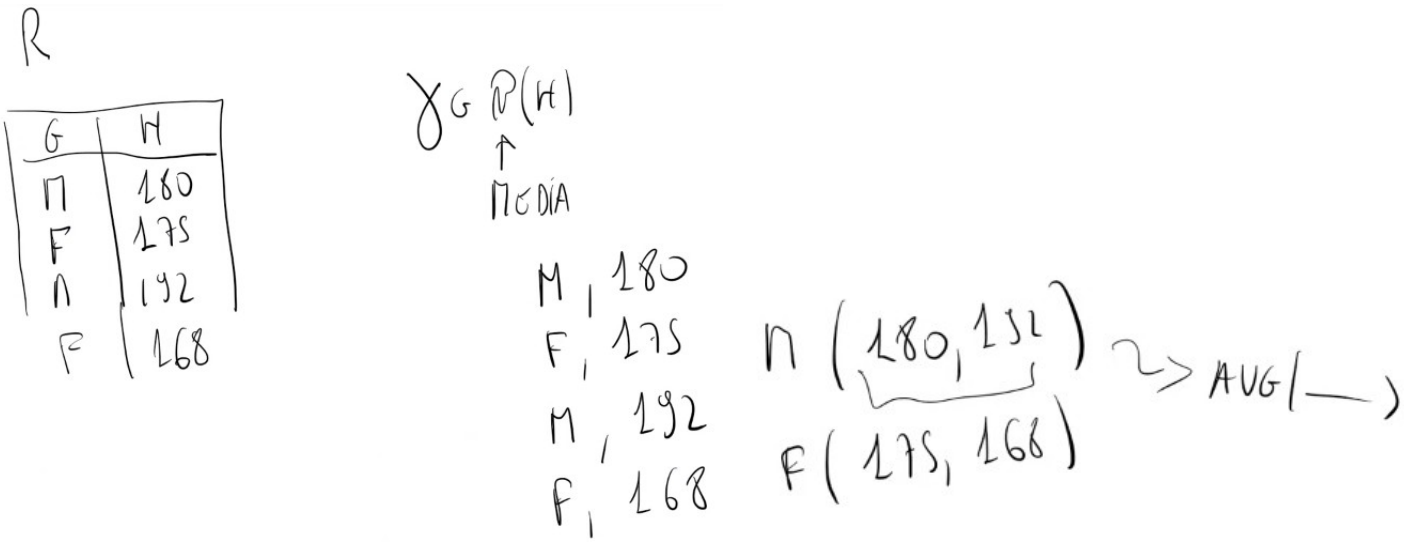

**Operazione di Unione**: prende tabelle con lo stesso schema e le unisce in una tabella unica, eliminando i duplicati (nella fase di shuffle).

**Operazione di Differenza**: da una tabella tolgo gli elementi di un'altra tabella. Per fare ciò associo un nuovo valore ad ogni valore di entrambe le tabelle e il risultato sarà ottenuto leggendo i dati che contengono solo il valore corrispondente alla tabella iniziale, quelli da togliere conterranno 2 valori.

**Operazione di Join Naturale**: prende due o più tabelle e concatena i dati in comune.

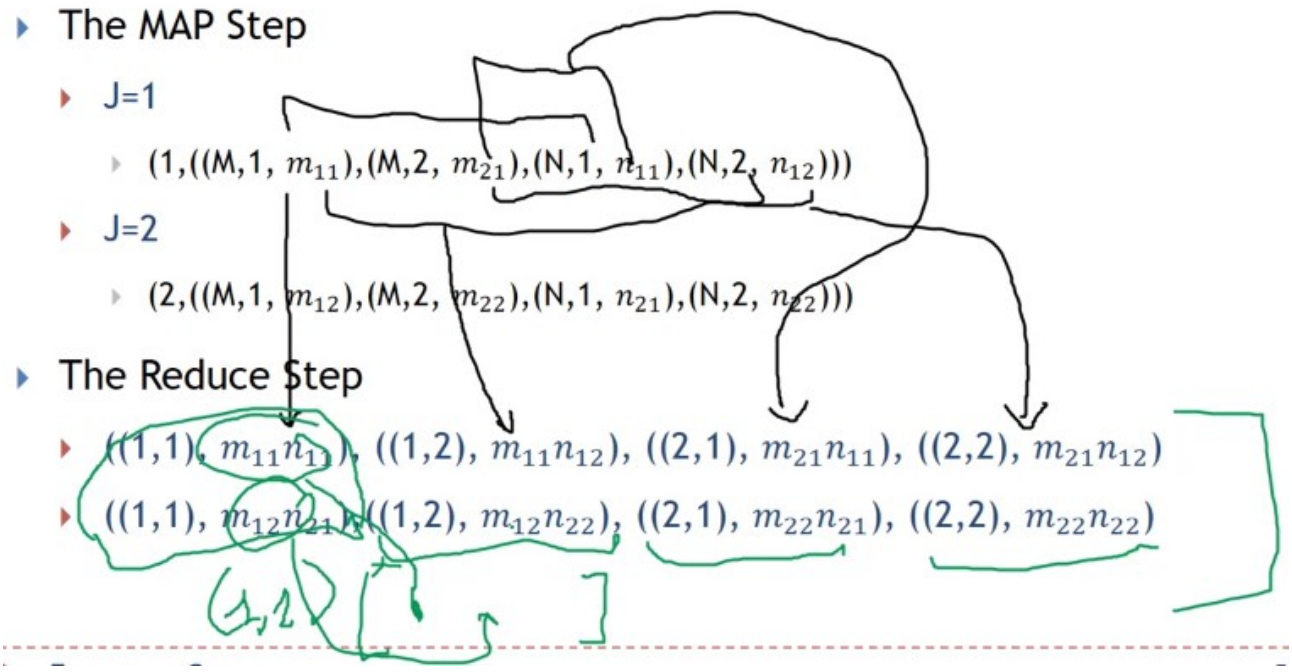


**Operazione di Raggruppamento e Aggregazione**: quando si hanno due o più colonne e si vuole raggruppare secondo il valore di una delle colonne e effettuare un'aggregazione rispetto ad un'altra colonna. (esempio: ordino per anno di nascita [raggruppamento] e conto i nomi sull'altra colonna [aggregazione])




**Moltiplicazione di matrici**

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix}$$




Il **costo di comunicazione** di un map-reduce è il fattore che *incide* di più in questo tipo di algoritmo, perché i costi computazionali al suo interno sono calcoli banali e quindi risultano bassi. Si ha trasporto di dati:
- fase iniziale, quando ricevo i dati su cui lavorare  e li porto ai [mapper] (non molto alto)

- fase di riduzione, quando produco l'output e devo stampare il mio risultato e portarlo al filesystem distribuito [reducer]
- fase di trasferimento dal mapper al reducer, costo molto alto perché i dati si possono trovare su macchine diverse, e possono essere ridotti attraverso un combiner

! Più coppie chiave-valore creo nel mapper più sarà lento il mio passaggio di dati

**Wall-Clock Time**: tempo che ci mette un algoritmo in parallelo a finire.

**Reducer Size [q]**: numero massimo di valori che possono essere associati ad ogni chiave, utilizzando il combiner questo valore sarà pari a 1.

**Replication Rate [r]**: quante coppie chiave-valore (uguali) vengono generate/replicate da tutti gli input, diviso il numero di input.

! I valori r e q sono inversamente proporzionali e possono essere espressi come: $q*r >= 2n^2$

**05/10/2020**

* Formule a slide 68

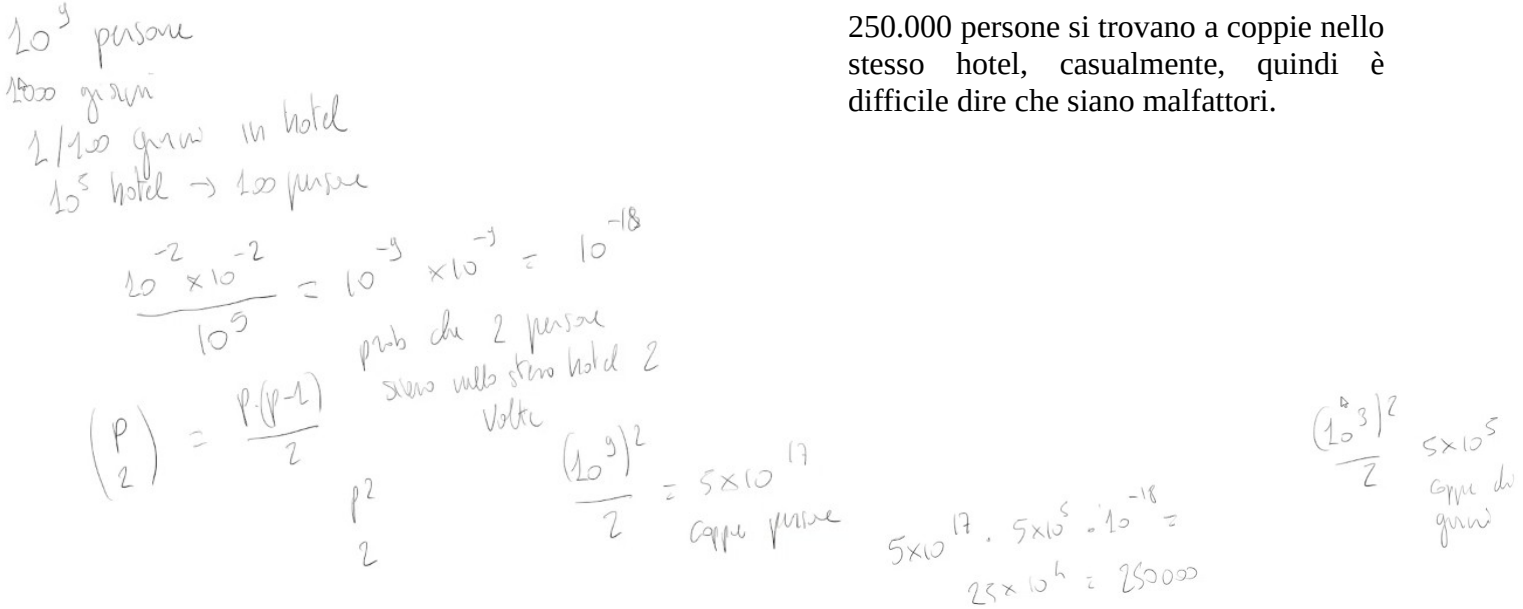
Problema di confrontare una immagine con 1000 immagini e quindi si ha un tempo di risposta molto grande in quanto i dati trasportati all'interno della rete sono moltissimi.
Non confronto le immagini all'interno dello stesso gruppo perché operazione molto veloce all'interno dello stesso chunk.

* Saltare slide da 70 a 77 + molto velocemente le rimanenti

**Data mining**: estrarre informazioni/contenuto/conoscenza da un insieme di dati, impara questo metodo attraverso il machine learning. Quando si ha a disposizione un'insieme molto grande di dati analizzati si possono predire un risultato atteso.

**Principio di Bonferroni**: non tutte le informazioni possono essere estratte dai dati, in quanto sono presenti dati spuri/casuali, ovvero che sono stati registrati in modo errato (es. esame del sangue che può essere eseguito due volte nel caso si abbia un risultato strano). Alcune situazioni quindi possono essere frutto del caso e quindi nei dati devo cercare qualcosa che avvenga con una frequenza superiore alla frequenza casuale.
Esempio:

250.000 persone si trovano a coppie nello stesso hotel, casualmente, quindi è difficile dire che siano malfattori.



**Data mining process**:
- analizzare il problema
- capire se ho abbastanza dati e se sono corretti
- preparo i dati (tabellare, normalizzare), questo processo occupa circa l'80% del tempo
- applico il data mining, testando i diversi algoritmi
- valuto i risultati ottenuti su dati reali
- deposito/metto in produzione il programma nel caso funzioni correttamente, questa fase dura anni perché il modello deve essere aggiornato e reso auto aggiornabile perché se no si sarebbe applicato un insieme di regole e non un algoritmo di machine learning, quindi devo continuare a modificare il mio modello che deve essere in grado di controllare i nuovi dati ricevuti in input

**06/10/2020**

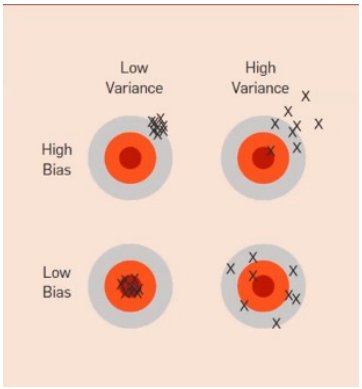
! Il modello va mantenuto aggiornato in quanto se è stato allenato su un certo set di dati, non sarà possibile usarlo su nuovi casi, dove cambiano i casi in ingresso.

**Rappresentazione**: come i dati in input vengono rappresentati, rappresenta l'operazione eseguita.

**Valutazione**: indica quanto funziona correttamente l'algoritmo, a differenza dell'accuratezza che misura quante volte classifico correttamente, ma in un dominio molto grande e un evento molto raro otterrò lo stesso un'accuratezza molto alta anche se la valutazione è negativa.

**Ottimizzazione**: indica la velocità nell'apprendimento, in che modo è efficiente nell'apprendimento.

**3 Regole fondamentali**
- la **generalizzazione** dell'algoritmo è il punto fondamentale, in quanto non deve funzionare in casi particolari ma in generale;
- i **dati non sono mai abbastanza**, ci saranno sempre dei casi che non sono presenti negli esempi;
- **overfitting**, quando funziona perfettamente con i dati di esempio, ma non funziona bene con altri.




**Bias**: tendenza di imparare o no, è alto se non imparo le cose corrette, viceversa è basso se imparo le cose giuste

**Varianza**: quando l'apprendimento è focalizzato o disperso, range stretto di valori o casuali.

**Classificazione**: quando si hanno delle classi e voglio sapere a quale classe appartiene l'istanza analizzata, ho delle "etichette" negli esempi (supervisionata).
**Associazioni**: quali sono gli oggetti presi insieme, studia le relazioni/combinazioni tra elementi.
**Regressione**: quando devo predire un numero, come altezza o costo.
**Clustering**: raggruppare elementi, non è supervisionato e quindi il gruppo lo crea l'algoritmo.

! Una rappresentazione tabellare non sempre è la soluzione migliore perché si perdono informazioni e non è sempre possibile capire la logica contenuta. (esempio tabella sorella con i soli nomi)

**Flattening data**: sui dati viene utilizzata la denormalizzazione, in quanto capire quali sono le colonne importanti nell'analisi tabellare è difficile, quindi creo un unico dataset contenente tutte le ripetizioni.

**Attributi**: descrivono un'istanza
- nominali, tipo di dato categorico, come il genere (uomo/donna)
- ordinali, quando è presente una relazione di ordinamento tra un dato e l'altro (freddo tiepido caldo)
- intervalli, gruppi di elementi
- ratio, valori numerici come altezza o peso

! Per tradurre i risultati degli attributi categorici li converto con associazioni booleane (0/1), oppure attraverso il "onenote encoding" che permette di creare una colonna per ogni valore che può capitare e effettuo associazioni booleane

**Output**: ogni algoritmo restituisce un risultato che può essere in due forme principali
- modello, rappresenta un albero decisionale
- uso, trasformazione del modello in un risultato per effettuare una predizione

**Rappresentazione dei dati ottenuti**
- *Tabelle decisionali*: metodo di rappresentazione dei risultati dati i diversi input.
- *Modelli lineari*: servono per rappresentare dati suddivisi in modo lineare su un piano.
- *Alberi decisionali*: rappresentazione dei dati su una struttura ad albero e possono essere di regressione se rappresentano dei numeri, oppure di classificazione nel caso vengano rappresentate delle classi. Posso utilizzare degli intervalli di valori per scegliere su quale ramo procedere. Nel caso missing value (nel caso in cui non conosco il valore contenuto), posso scegliere in modo randomico o in base a una stima oppure scegliendo a priori il ramo da scegliere.
Quando le regole seguite nell'albero sono ordinate e numerate si parla di "decision list", nel caso opposto si parla di regole non ordinate quando non ha importanza l'ordine.
- *Basate su esempi*, non imparo regole e non elaboro i dati, ma confronto gli esempi più vicini al mio caso di studio che voglio predire.

**12/10/2020**

**Cluster**: insieme dei dati
- *overlapping*, i dati possono fare parte di più insiemi.
- *probabilistici*, ogni elemento ha una probabilità di appartenenza ad un cluster o un altro.
- gerarchici/*dendrogramma*, elementi raggruppati a 2 a 2 in un albero e trovati attraverso una "soglia di similarità".

**Algoritmo One Rule**: si basa su un singolo attributo per prevedere il risultato, calcolando il tasso di errore composto dalla somma dei possibili valori di esso fratto il numero di possibilità totali e scelgo l'attributo su cui il tasso di errore è più basso. Nel caso i valori sono rappresentati da numeri potrebbe avere problemi perché avrei pochi esempi per ogni caso portando a un insieme sbagliato di regole e quindi creando overfitting. Per risolvere questo problema creo dei gruppi di valori scegliendo un numero di ricorrenze che devono essere presenti nel mio gruppo e quindi associo a quel gruppo una determinata regola.

**Algoritmo Naive Bayes**: tutti gli attributi hanno la stessa importanza e sono statisticamente indipendenti corrispondente alla probabilità congiunta, calcolata con la moltiplicazione delle diverse probabilità e divido per la somma dei risultati per trovare la probabilità finale. Il grafico risultante sarà rappresentato da diversi "intorni" dei punti. Per applicarlo con valori numerici, assumo che i valori seguano un andamento gaussiano e calcolo la densità per ogni valore.

**Smoothing**: tecnica utilizzata nel caso la probabilità di un caso in cui un valore sia possibile 0 volte, portando a 0 la probabilità finale, per risolvere questo problema aggiungo 1 a tutti i valori.

! Nel caso in cui ho un missing value non c'è nessun problema e basta non considerarlo nella moltiplicazione.

**Algoritmo C4.5**: utilizzato per costruire un decision tree da un dataset, devo riuscire a costruire un albero il più "basso" possibile quindi meno innestato e meno regole contiene, questo è possibile scegliendo il giusto attributo. Viene scelto l'attributo che genera il maggior numero di insiemi puri ovvero con valori che hanno lo stesso valore (polarizzati).

**Entropia**: l'informazione/formula misurata in bit richiesta per descrivere/rappresentare un dataset. Quanto è vario, quindi più bit uso, più è complesso il mio dataset. Il valore massimo è 1 e più cala meglio è. Dove nel calcolo "p" identifica la frazione di valori che assumo un determinato valore, diviso il numero totale di valori.

**13/10/2020**

**Guadagno di informazione**: calcolato come la differenza tra l'informazione prima e dopo lo split.

! Nell'analisi dei dati di training viene esclusa la colonna della chiave (ID) perché overfitterebbe e non imparerebbe nulla.

**Gain ratio**: calcolato come il rapporto tra il guadagno di informazione e l'intrinsic info, che rappresenta in quanti valori diversi/unici viene suddivisa l'informazione.

**Algoritmo PRISM**: si cerca di coprire tutti i dataset generando diverse regole che generano riquadri, ma questo molto spesso porta all'overfitting.

**Support Vector Machines (SVM)**: si pone l'obiettivo di costruire un margine il più ampio possibile per dividere i due insiemi. Si usa l'errore al quadrato perché penalizza gli errori molto elevati.

**Outlier**: caso particolare che appartiene a un gruppo ma le sue caratteristiche specifiche lo fanno appartenere ad un altro, si tratta di un caso anomalo.

**Logistic regression**: tecnica che consente di classificare gli elementi, sfrutta un'equazione lineare per trovare la classificazione, quindi la probabilità che un elemento sia di un tipo o dell'altro. *Odds* rappresenta il rapporto tra la probabilità che l'elemento appartenga alla classe predetta e la probabilità inversa. Quando la funzione è elevata appartiene quasi sicuramente alla classe, mentre più è bassa meno sono le probabilità che appartenga alla classe.

**19/10/2020**

**Instance-based learning**: prendo un esempio con le condizioni il più vicine (distanza minima) possibili alle condizioni del caso che sto analizzando e predico quindi sulla base di esempi, non imparando mai. Sono presenti dei problemi come quello di avere valori con distanza molto elevata gli uni dagli altri che possono assumere importanza diversa (come nel caso del reddito che può variare da zero a milioni), è possibile risolverlo *normalizzando* facendo assumere a tutti i valori un determinato valore compreso tra 0 e 1. Un altro problema viene riscontrato quando ho un dataset molto ampio e quindi trovo un'insieme di valori con distanza molto piccola ma non so quale

scegliere, scelgo un'insieme di punti e calcolo i "*voti*" attraverso l'algoritmo K-NN, ovvero quante classe sono uguali, in modo casuale nel caso siano pari oppure con media pesata rispetto la distanza.

! Generalizzerà meno un K basso (= 1), viceversa per un K elevato dove generalizzando meno cala di conseguenza l'overfitting.

**Clustering**: tecnica che si utilizza per effettuare raggruppamenti senza classi prefissate (noi al massimo stabiliamo quanti gruppi ottenere), quindi in gruppi naturali. Si suppone di avere un dataset numerico ed ogni punto può essere collocato nello spazio attraverso coordinate del punto e li classifico in cluster attraverso la loro distanza.
- disjoint/disgiunti: un elemento può appartenere ad una sola classe
- overlapping: un elemento può appartenere a diverse classi
- deterministic: ottengo un risultato di appartenenza ad una classe
- probabilistic: ottengo una probabilità di appartenenza ad una classe
- flat: semplice raggruppamento/assegnamento del punto
- hierarchical: posso costruire un albero di appartenenza, considero i punti più vicini (a distanza più bassa) e li posso unire/fondere fino a quando raggiungo le mie regole/tecniche.

**Centroide**: elemento che identifica un'insieme di punti nello spazio euclideo, è calcolato come la media delle distanze dei valori assunti dalle coordinate dei punti che ho raggruppato all'interno del cluster. Da questo può essere costruito un dendrogramma per rappresentare i cluster.

**Clustroide**: elemento più vicino al centroide, quindi rappresenta il punto esistente più vicino al punto rappresentato dal centroide che non esiste ma è calcolato.

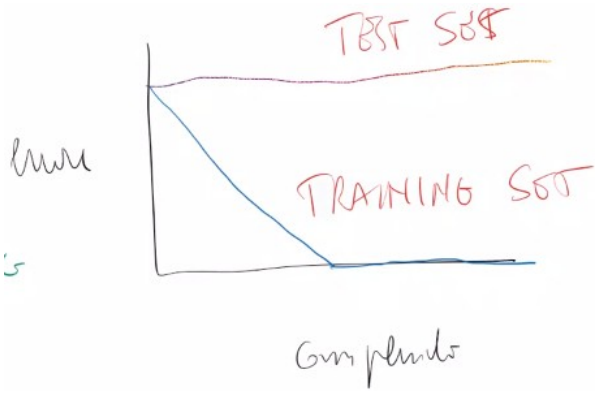
**Algoritmo K-means**: si parte da un numero K che rappresenta il numero di gruppi che voglio ottenere, assegno quindi casualmente K punti nel mio piano e li considero come centroidi dei cluster ipotetici e assegno il centroide al punto più vicino ad esso, creando un cluster e continuando a riassegnare il centroide fino a quando non mi muovo più, o mi muovo meno di una soglia preimpostata.

**Tecniche per evitare l'Overfitting**
- Generalizzazione
- *Fitting Graphs*: grafico utilizzato per valutare il tasso di errore rapportato alla complessità del modello, devo riuscire ad ottenere il giusto compromesso.

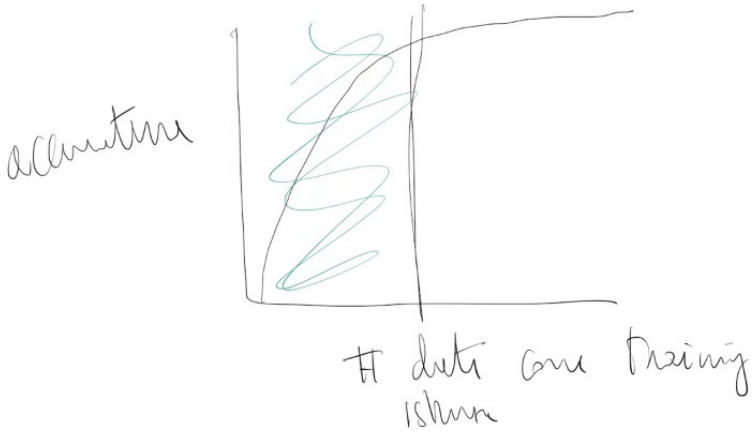
**Hold out**: metodo che toglie un 30% dei dati di training e li utilizzo in fase di test.

**20/10/2020**

**Valutazione del modello**: più il il modello diventa complesso, più gli errori calano ma fino a un certo punto, dove si andrà incontro all'overfitting, lo stesso vale anche per gli alberi.

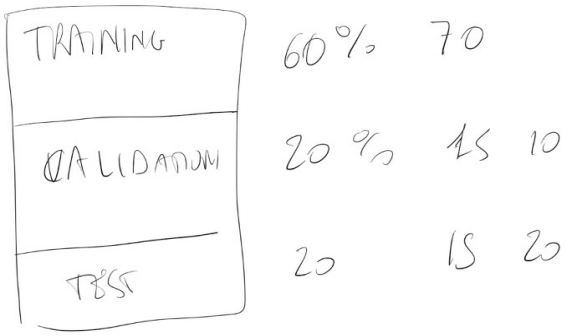
Problemi della **suddivisione del dataset** in dati di Training e Test perché potrebbero essere ordinati ad esempio in ordine cronologico e quindi devo prenderli in modo *randomico*, la giusta percentuale di suddivisione (60/40) *dipende dalle dimensioni del mio dataset*, perché dopo un po' non imparo più e quindi non mi servono tantissimi dati anche se il mio dataset è enorme, mi conviene utilizzarli per il test, come rappresentato nella curva di apprendimento:




**Test di significatività**: rappresenta la stabilità del nostro algoritmo, ovvero se il mio algoritmo può esteso alla realtà oppure no, nel caso fosse poco significativo, come nel caso in cui ho pochi casi di test da analizzare. I costi assegnati ad ogni errore sono differenti come nel caso dei falsi positivi e falsi negativi (inaccettabili) nei tamponi.

**Resubstitution error**: tasso di errore ottenuto sul training set.

**Test set**: istanze indipendenti che non hanno preso parte alla formazione del classificatore.




Nel **Validation set**, vado a valutare se la porzione di dati estratti dal training è corretto rispetto a ciò che voglio ottenere, quindi rappresenta la validazione del mio algoritmo, come il conteggio dei nodi del mio albero, e non la validazione dei risultati ottenuti dal training. Spesso omesso o confuso con il test set.

**K Fold Cross Validation**: metodo che serve per valutare il mio algoritmo, dividendo il dataset in fold e valutando il risultato ottenuto preso un singolo fold.
- **Selezione stratificata**: se nel *folder* è presente la stessa percentuale di classi contenute nel dataset originale, quindi non un folder che contiene dati tutti uguali ma diversificati. Il massimo numero di folder corrisponde al numero di istanze minore, il numero ottimale sarebbe 10 ma non obbligatorio.
- **Leave-One-Out**: provo N volte l'algoritmo, N corrisponde alle righe, prendendo un elemento come test e i rimanenti come training, utilizza nel miglior modo i dati per il training ma computazionalmente molto costoso.
- **Bootstrap**: nel caso di pochi dati, creo nuovi dataset da quello originale estraendo dati a caso, posso trovarmi quindi a riutilizzare gli stessi dati in dataset diversi. Le istanze hanno probabilità (1/n) di essere prese e (1 – 1/n) di non essere prese, le istanze che non prendo mai saranno quelle che utilizzerò nella fase di test, queste corrispondono circa al 36,8% del dataset.

**Confusion Matrix**: sulle righe le classi reali e sulle colonne le classi prodotte, matrice che serve per calcolare quante classi predico correttamente e non, utilizzata in fase di Validation. Un falso positivo è meno importante di un falso negativo nel caso di risultato medico, perché nel caso di falso positivo posso eseguire altri test per avere conferma, non si può dire lo stesso del caso contrario.

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

Overall success (accuracy)= (TP + TN) / (TP+TN+FP+FN)

TP rate = TP / (TP+FN)

FP rate = FP / (FP+TN)

! FP può essere anche calcolato come [1 - Accuracy]

**26/10/2020**

! Non sempre l'accuratezza è sinonimo di qualità perché calcolata in totale (nella formula sopra riportata), in quanto se ho un tasso di falsi negativi molto alto, non è un buon risultato in quanto rischio di tralasciare un caso, quindi è meglio avere un falso positivo.

**Area Under Curve (AUC)**: area sotto la curva che descrive i valori sopra descritti, più è vicina al punto (0,1) più è corretto il mio algoritmo, ciò significa che non sbaglia gli elementi di classe 1 e 0.

| | |
|---|---|
| Percentage of retrieved documents that are relevant: precision=TP/(TP+FP) | Quante tra le persone che io predico mi abbandonino, veramente mi abbandonano. |
| Percentage of relevant documents that are returned: recall =TP/(TP+FN) | Quante persone che io predico, mi abbandonano rispetto al numero complessivo che mi abbandoneranno. |

**Ensemble Learning**: mettono insieme tecniche di machine learning.
- **Bagging**: combina diverse predizioni sullo stesso dataset, creo tanti algoritmi sullo stesso dataset e valuto il risultato di ogni algoritmo e scelgo per maggioranza la classe maggiormente votata. Utilizza la Randomization che associa pesi diversi ad ogni istanza. I modelli possono essere di diversi tipi, come alberi decisionali e linear regression.
- **Boosting**: eseguo tanti algoritmi ma ognuno su un determinato insieme, sulle istanze su cui non sono sicuro andrò ad applicare un altro algoritmo.

**Frequent itemsets**: insieme di elementi che appaiono frequentemente insieme, come nel caso degli acquisti (*Market-basket model*), con l'obiettivo di creare regole utilizzate nella disposizione degli items (oggetti) nei baskets (contenitore). Viene creata una "*soglia di supporto*" per determinare il significato di frequenza. Questo tipo di analisi può essere applicata in diversi casi come nella spesa o in un documento di testo. Utilizzato per *descrivere relazioni molti a molti*. Questo processo può essere diviso in due parti, una dove trovo l'insieme di item set (parte più difficile) e una dove trovo genero le regole di associazione.

**Confidenza**: utilizzata per misurare le *regole di accoppiamento*, dove il *Supporto* (Supp) indica il numero di volte in cui i valori di quegli insiemi appaiono insieme. Solitamente il suo valore è **<= 1** perché la probabilità che più elementi appaiano insieme è minore di un numero minore di elementi. Più è elevata e più la regola ha un'importanza elevata. Esistono però certe regole che anche se hanno confidenza massima (=1) non hanno alcun valore perché potrebbe apparire in tutti gli accoppiamenti, per ovviare a questo problema viene introdotto l'interesse.

$$conf(I \to J) = \frac{supp(I \cup J)}{supp(I)}$$

**Interesse**: utilizzato nei casi di oggetti sempre presenti in qualsiasi insieme, quindi l'interesse in questi casi viene portato a 0, come nel caso delle sporte per la spesa, perché la probabilità di j è 1.

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

**27/10/2020**

! Il problema principale di dover analizzare un'insieme per calcolare gli insiemi frequenti è che ogni volta dovrei leggere un numero molto alto di dati (per confrontare i vari accoppiamenti), e ogni volta che aggiungo un dato l'insieme aumenterà in maniera esponenziale.

**Triangular Matrix Method**: metodo che permette di trasformare una matrice in un vettore, la posizione viene calcolata nel seguente modo, mantenendo una notazione ordinata in maniera lessico-grafica.

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

**Triples Method**: permette di rappresentare un dato come una tripla [i, j, c] dove "i" e "j" indicano il prodotto1 e prodotto2 e "c" indica il numero di occorrenze della coppia, i < j così non ripeto le stesse coppie. Ha senso utilizzare questo metodo nel caso ci siano coppie che non compaiono, perché vengono creati solamente gli elementi con c > 0.

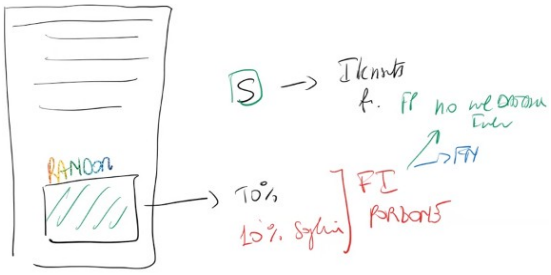
**Monotonicità di un Itemsets**: se I è un set di items frequenti, all'ora ogni sottoinsieme di I è frequente.

**Itemset Massimale**: insieme di elementi che è molto frequente ma è l'insieme massimo, perché aggiungendo un qualsiasi elemento questo set non è più frequente.

! Ci limitiamo al conteggio delle coppie perché sono sicuramente superiori al numero di qualsiasi gruppo di dimensioni maggiori.

**Algoritmo A-Priori**: algoritmo per contare gli insiemi, per contare gli insiemi di N elementi deve eseguire N passate del mio dataset.
- Nella prima passata viene eseguito un mapping degli elementi, creando un codice per ognuno, quindi eseguo un conteggio dei singoletti, avendo come risultato della prima passata la frequenza di ogni elemento.
- Stabilisco una soglia per mantenere una percentuale di elementi più frequenti. Creo le possibili coppie tra questi elementi che andrò ad analizzare per sapere se sono frequenti. Per il principio di monotonicità inverso posso tralasciare quindi gli elementi meno frequenti senza problemi.
- Eseguo la seconda passata del mio dataset per verificare la frequenza dei miei insiemi.

**Algoritmo Randomized**: tecnica approssimata che viene eseguita in memoria e quindi risulta molto più veloce. Ha il problema dei Falsi Positivi, in quanto potrei trovare elementi molto frequenti nella mia porzione ma che in realtà sono poco frequenti nell'interno dataset, questi possono essere tolti con diverse tecniche. Sono anche presenti i Falsi Negativi per lo stesso ragionamento ma che non possono essere trovati e quindi non eliminati.
- Scelgo un soglia di supporto S.
- Prendo una porzione random piccola del mio dataset, in modo che stia in memoria.



**02/11/2020**

**Algoritmo di Park, Chen e Yu**: variazione dell'algoritmo a-priori, utilizza una funzione di hash per eseguire la classificazione (dato un dominio di target prefissato, dal numero preso in input devo associare il bucket).
- Prima passata dove conto i singoletti frequenti e per ogni scontrino genero le possibili coppie. Applico una funzione di hash ad ogni coppia, associandoli a bucket, trovando i bucket frequenti (il numero di bucket deve essere per definizione inferiore al numero di coppie).
- Dai singoletti frequenti posso generare le coppie frequenti, ma solamente quando la coppia candidata finisce, con l'applicazione della funzione di hash, in un bucket frequente.

**Algoritmo di Savasere, Omecinski e Navathe (SON)**: tecnica esatta che non genera falsi positivi.
- Stabilisco un numero di chunk in cui suddividere il mio dataset.
- Data la soglia S iniziale, la divido per il numero di chunk scelti e trovo la nuova soglia da associare ad ogni chunk.
- Nella prima passata analizzo gli elementi in ogni chunk e trovo gli elementi frequenti per ogni chunk, superiori alla soglia, trovando un'insieme di elementi frequenti $F_i$ ma non so se sono frequenti nell'interno dataset.
- Calcolo la frequenza di $F_i$ per ogni chunk, trovando un valore $V_i$ per ogni chunk.
- Sommo i valori della frequenza di ogni $F_i$ e se supero la frequenza di soglia allora sarà frequente sull'intero dataset.
( Il map si occupa di scansionare il dataset e trovare i singoletti frequenti e il reduce si occupa di eliminare i duplicati )

[RIGUARDARE IMPLEMENTAZIONE MAP REDUCE]

**Algoritmo Toivonen**: non genera falsi positivi e falsi negativi, ma in certi casi produce risultati non esatti che non possono essere interpretati.
- Scelgo parte del dataset e calcolo gli itemsets frequenti nel sottoinsieme.
- Costruisco il "**negative border**" degli insiemi frequenti. Il negative border rappresenta l'insieme di elementi che non sono frequenti nel sample, ma tutti gli immediati sottoinsiemi sono frequenti.

**03/11/2020**

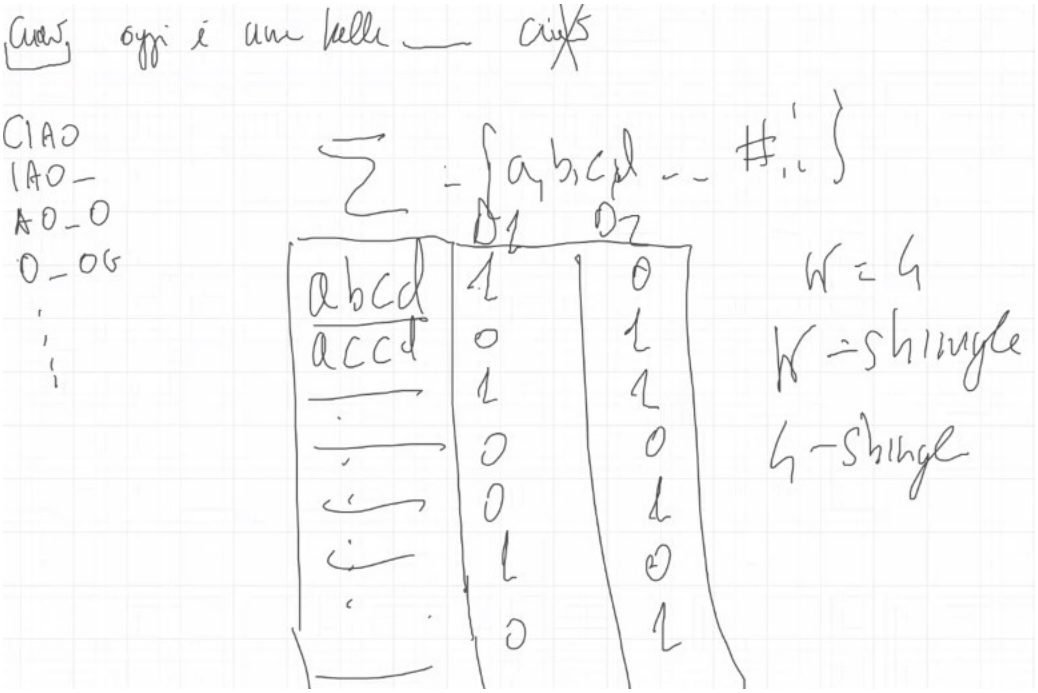
**Misura di Jaccard**: prendendo due oggetti con tante caratteristiche per ognuno di essi, questa misura valuta gli elementi comuni diviso il numero di caratteristiche totali. Quindi valuta le caratteristiche comuni rispetto le caratteristiche che vengono valutate nel complesso. L'obiettivo è trovare elementi che sono vicini tra loro in un grande insieme.

**Collaborative Filtering**: si utilizza la similarità per trovare utenti che sono simili, si utilizza in sistemi di raccomandazione per effettuare suggerimenti adatti all'utente.

**Similar Documents**: pipeline utilizzata per trovare documenti che sono simili, per poterli valutare devo valutarli a coppie.
- **Shingling**: prendo il documento e lo converto in un insieme, convertendolo attraverso un vettore enorme, contenente il documento sotto forma di caratteri.
- **Minhashing**: converto la rappresentazione globale in un insieme più ristretto, in *signatures* (piccole firme) che rappresentano il documento.
- **Locality-sensitive hashing**: confronto le signatures per trovare i documenti più simili tra loro.

! Non posso prendere K=1 perché gli elementi del mio dizionario, in un documento molto grande, comparirebbero almeno una volta trattandosi di singole lettere. Non posso nemmeno prendere un K troppo grande perché la matrice sarebbe di dimensioni troppo alte, infatti supponendo 40 caratteri

possibili e supponendo K=7, nella mia matrice dovrei inserire 40^47 elementi, quindi un numero enorme.




[RIGUARDARE MATRICE DI MIN-HASHING]

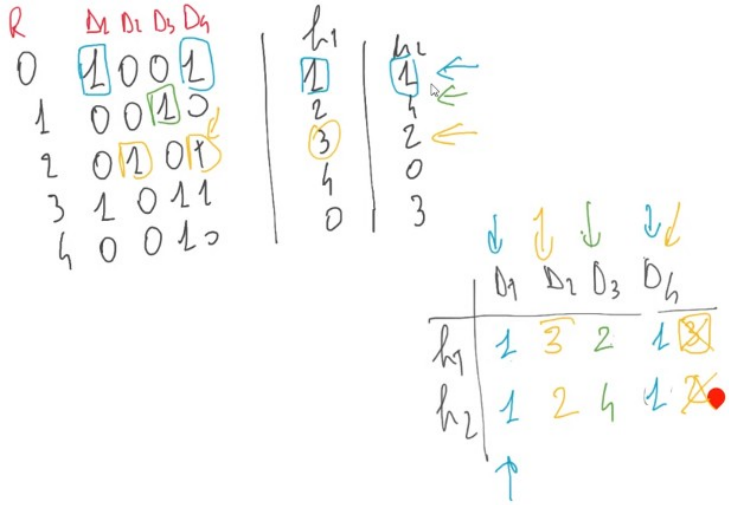
- La tabella *centrale* indica i diversi documenti, uno per ogni colonna.
- La tabella di *sinistra* colorata indica le permutazioni da eseguire, il numero contenuto al suo interno rappresenta il numero di riga a cui corrisponde la permutazione.
- La tabella a *destra*, contiene gli 1 in base alla prima occorrenza di 1 nel documento in base alle permutazioni, questa è considerata la matrice risultato.

$$p[h(D_1) = h(D_2)] = sim(D_1, D_2)$$

$$sim = \frac{D_1 \cap D_2}{D_1 \cup D_2} = \frac{D_1 \text{ AND } D_2}{D_1 \text{ OR } D_2}$$

! La probabilità che due valori di min-hashing coincidano è uguale alla loro similarità di Jaccard.

**09/11/2020**



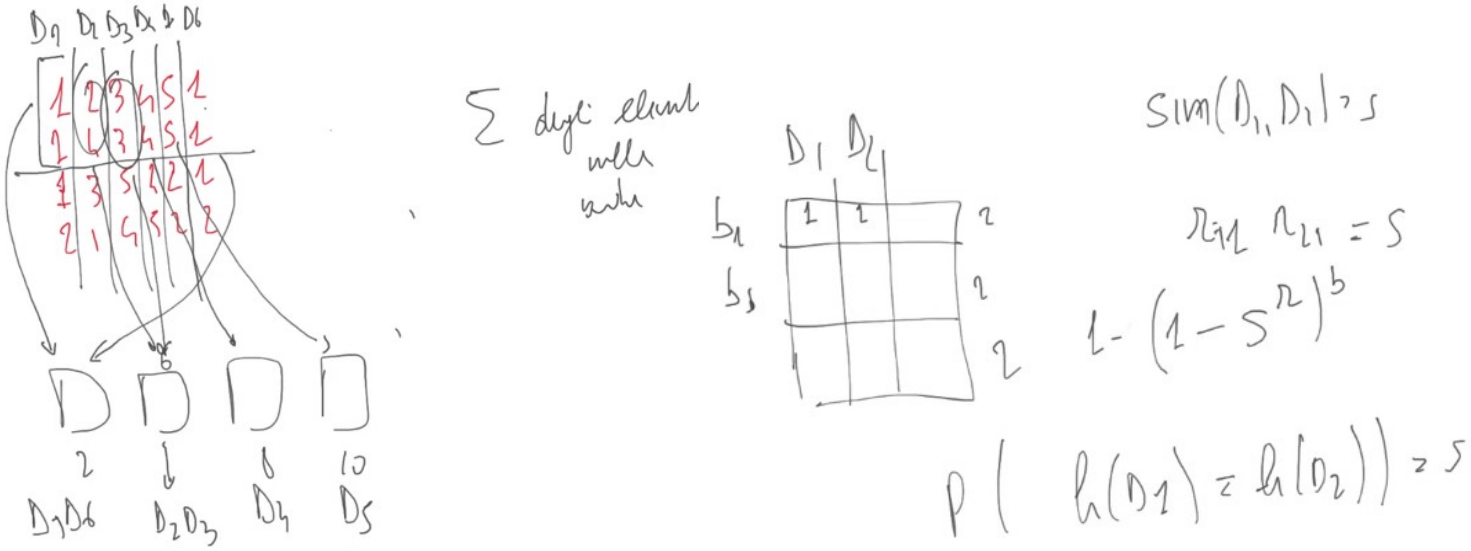

Inizialmente la tabella che contiene i valori di min-hashing è composta da tutti i valori a infinito e devo controllare se i nuovi valori sono *minori* di quelli già presenti, in questo caso devo sostituirli nelle colonne indicate dagli 1 presenti nella tabelle iniziale, in caso contrario, quindi se *maggiori*, non devo effettuare la sostituzione.

! Il caso sfortunato in cui non so se i documenti confrontati sono simili è quando nella tabella ho due valori infiniti.

Per ovviare al caso in cui devo confrontare colonne con valore infinito, posso escluderle/ridurle, ma diventa un problema quando questo numero diventa molto grande.

! Attraverso la funzione di hash divido la matrice contenente tutti i documenti da confrontare, in tanti sottoinsiemi (bucket) di documenti candidati ad essere simili.

**LSH**: tecnica utilizzata per dividere la matrice di hashing in "*bande*" contenente un numero di documenti ognuna. Successivamente applicherò la funzione di hash una volta per ogni banda, assegnando ogni banda ad un bucket (è possibile inserire più di una banda nello stesso bucket) trovando le bande simili all'interno dello stesso bucket. I valori delle bande vengono messe all'interno dello stesso bucket se hanno valori di banda simili, successivamente all'interno del bucket andrò ad eseguire i controlli, rendendo il processo molto più veloce avendo meno elementi da confrontare.



! Se due documenti, complessivamente, hanno similarità s allora la probabilità che due righe siano uguali è s.

**10/11/2020**

| | |
|---|---|
| $S^r$ | probabilità che tutte le righe siano uguali in una banda |
| $1 – S^r$ | probabilità complementare della precedente |
| $(1 – S^r)^b$ | probabilità complessiva sull'intero documento |
| $1 - (1 – S^r)^b$ | probabilità che ci sia almeno una banda in cui tutte le righe coincidono |

*SLIDE 89 Riassunto della tecnica completa LSH

Riguardare seconda parte lezione 10/11 PageRank – Link Analysis

**16/11/2020**

*Stocastico*: la somma delle colonne deve fare 1.
*Irriducibile*: tutte le colonne della matrice devono essere raggiunte dalla transizione (ogni pagina non è mai isolata, ogni pagina deve essere raggiungibile da almeno un nodo).
*Aperiodica*: non ci devono essere pagine che si collegano a pagine che sono collegate alla pagina iniziale, quindi non possono essere presenti cicli, senza via di uscita. Questo perché tutto il valore

che viene diretto verso la pagina sarebbe restituito solo a se stessa e quindi non continuerebbe nella rete, prendendo il nome di Spider Traps.

! Per evitare i link senza via di uscita non li conteggio nel calcolo del page rank e utilizzo una nuova funzione che tiene conto di altri parametri.

**17/11/2020**

**Matrice di transizione**:
- $\mathbf{M_{ij} = 1 / |d_j|}$ ha un valore diverso da 0 se il nodo j punta a un altro nodo i e il valore di dj corrisponde alla cardinalità in uscita da quel nodo.

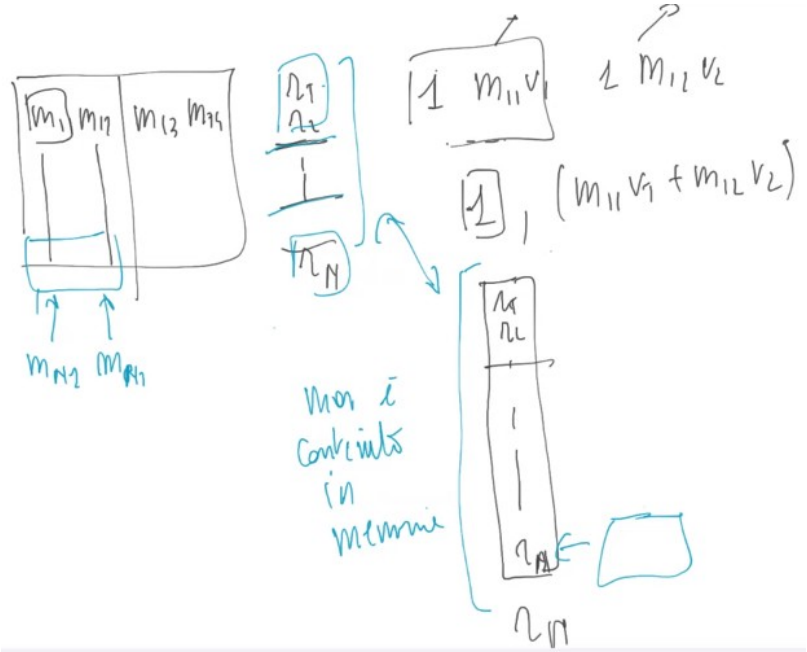
**Termine di teleport**: termine **β** inserito all'interno della funzione di Page Rank per non permettere alla funzione di convergere (nel caso ci siano nodi che puntano a se stesso).

**Equazione di PageRank**:

$$r = \beta M \cdot r + \left[\frac{1-\beta}{N}\right]_N$$

L'operazione M*r è molto complessa a livello computazionale perché si tratta di una moltiplicazione tra matrice e vettore, quindi occupa molto spazio in memoria. Per risolvere questo problema posso applicare il metodo del Map-Reduce ma la matrice M è dimensionalmente elevata e sparsa e r, di conseguenza, ha molti valori. Posso dividere il vettore r in bande sulle righe e effettuare lo stesso procedimento sulla matrice M. Se il Reduce è commutativo e associativo, l'operazione viene copiato anche nel Map.

Il Combiner mi permette di ottenere un valore unico per ogni operazione sulla stessa chiave, senza il quale otterrei un valore per ogni operazione, però questo implica che devo aver letto tutta la matrice per essere a conoscenza di aver terminato tutte le chiavi uguali, che però non può essere contenuto in memoria in quanto è di grandi dimensioni e quindi l'algoritmo cala di prestazioni.



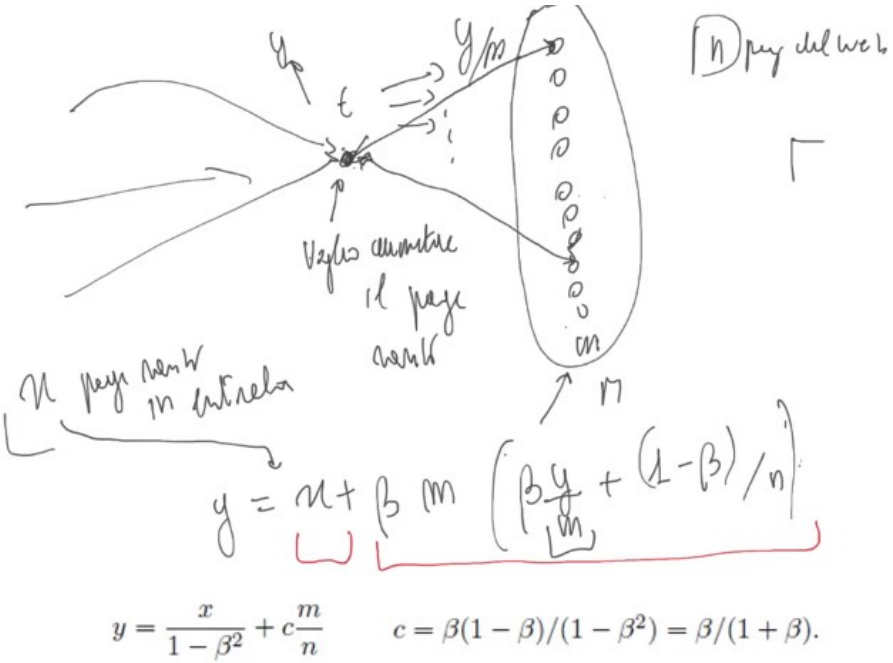

! Per spezzare il risultato posso dividere la matrice in bande orizzontali oltre a quelle verticali che ho già diviso in precedenza. Infine troverò $r_{new}$ diviso in tanti blocchi.

**Topic-sensitive PageRank**: utilizzati per indirizzare la ricerca verso un'area specifica, che consente di effettuare ricerche più specifiche nel caso ad esempio si cerchi un termine che può avere più significati.

$$\mathbf{v}' = \beta M\mathbf{v} + (1-\beta)\mathbf{e}_S/|S|$$

Simile all'equazione precedente con la differenza che la matrice $\mathbf{e}_s$ contiene valori diversi in base all'argomento, vincolando i percorsi verso gli interessi dell'utente.

**Link Spam**: tecnica per imbrogliare il Page Rank, utilizzando siti vuoti che linkano la pagina a cui voglio aumentare il Page Rank, e di conseguenza quando una pagina esterna linka la mia pagina, tutto il Ranking verrà suddiviso alle mie pagine fantasma che successivamente lo restituiranno alla mia pagina (questo avveniva attraverso i commenti inseriti in pagine importanti).




$$y = \frac{x}{1-\beta^2} + c\frac{m}{n} \qquad c = \beta(1-\beta)/(1-\beta^2) = \beta/(1+\beta).$$

**Trust Rank**: è il Page Rank calcolato su pagine affidabili, utilizzato per risolvere il problema delle pagine non affidabili che utilizzano il metodo precedente, riconoscendo strutture di pagine che si linkano a vicenda.

**Algoritmo HITS**: attribuisce ad ogni pagina due valori/score.
- *hub*, identifica quanto la pagina è esperta.
- *authority*, indica l'autorità della pagina in base alla rilevanza del contenuto.

All'inizio assegno ad ogni hub valore 1 e assegno ad ogni authority la somma dei valori degli hub che puntano a questa. Successivamente porto ad ogni hub il valore ottenuto in authority. Ad ogni passaggio *normalizzo* ad 1 i valori per poter convergere.

**<u>23/11/2020</u>**

**Semantic Text Analysis**: tecnica utilizzata per analizzare e stabilire il contenuto di una pagina web o documento, per poter mettere a disposizione all'utente le informazioni inerenti alla richiesta.
- **pull mode**, quando è l'utente che stabilisce quando ha bisogno di un'informazione come avviene attraverso un search engine (applicazioni che implementano il Text Retrival). Questo processo può avvenire in due modi, Querying se l'utente effettua un query di ricerca oppure Broswing se l'utente naviga nel web attraverso collegamenti contenuti nelle pagine collegate tra di loro.

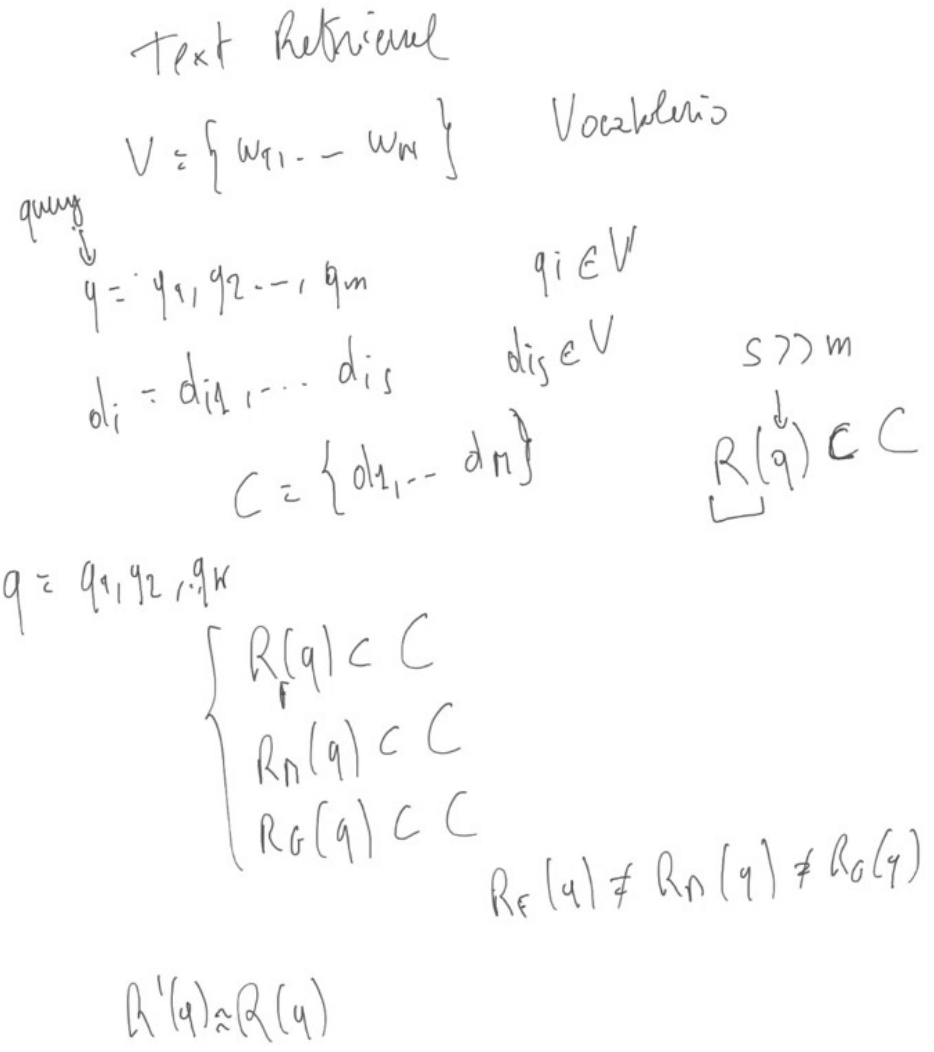
- **push mode**, il sistema prende iniziative sulla visualizzazione nel web, come avviene ad esempio con i suggerimenti dei film o notizie scelte in base alla propria navigazione.

**Text Retrival**: branca della ricerca che consente di sviluppare strumenti che supportano l'accesso ai documenti e permette una ricerca più mirata a ciò che l'utente vuole davvero, rendendo la ricerca molto più veloce all'interno del web che sarebbe infinita visto il grande numero di pagine.

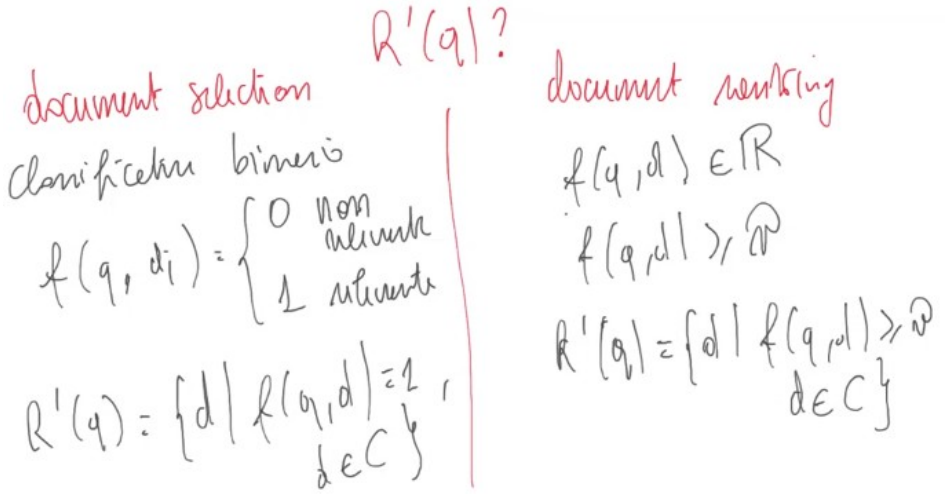
! Ogni utente potrebbe inserire diverse parole chiavi per query ma volere le stesse informazioni, quindi esistono diversi modi per riferirsi alla stessa informazioni, quindi una query descrive un'informazione in maniera incompleta.

**Text Retrival vs Database Retrival**:
- Per effettuare una query su un Database Retrival si devono avere conoscenze sulla struttura e quindi su cosa eseguire la query, e il risultato interessato è una tupla, di conseguenza il risultato di una query sarà sempre lo stesso.
- Invece sul Text Retrival non conosco la struttura del dato che sto cercando in quanto i dati non hanno una struttura propria e quindi non otterrò sempre lo stesso risultato, ma cambierà in base alle ricerche dell'utente.

! Nel Search Engine una componente fondamentale non è tanto la velocità della ricerca ma capire cosa l'utente vuole che sia ritornato.

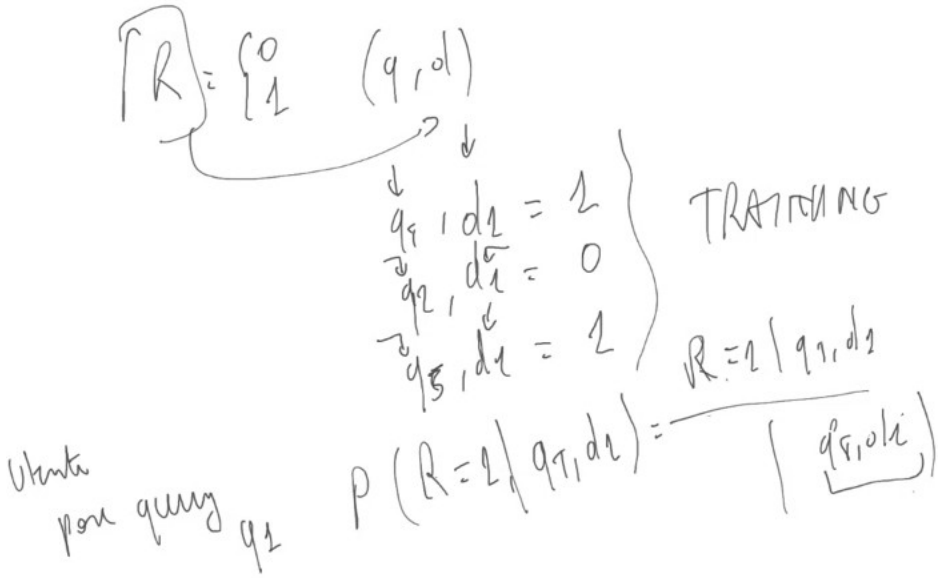


**R’(q)** rappresenta l’insieme di documenti rilevanti per la ricerca effettuata dall’utente, per comporre questo insieme esistono due metodi:
- **Document Selection**: utilizza un classificatore binario a cui viene passata una query e un insieme di documenti e assegna un valore binario in base alla rilevanza del risultato.
- **Document Ranking**: utilizza una funzione che applicata ad una query e un documento, restituisce un valore reale, potendo così scegliere di mostrare agli utenti solo i documenti con un valore maggiore di una determinata soglia, oppure senza soglia utilizzando un metodo decrescente in base alla rilevanza delle parole ricercate.



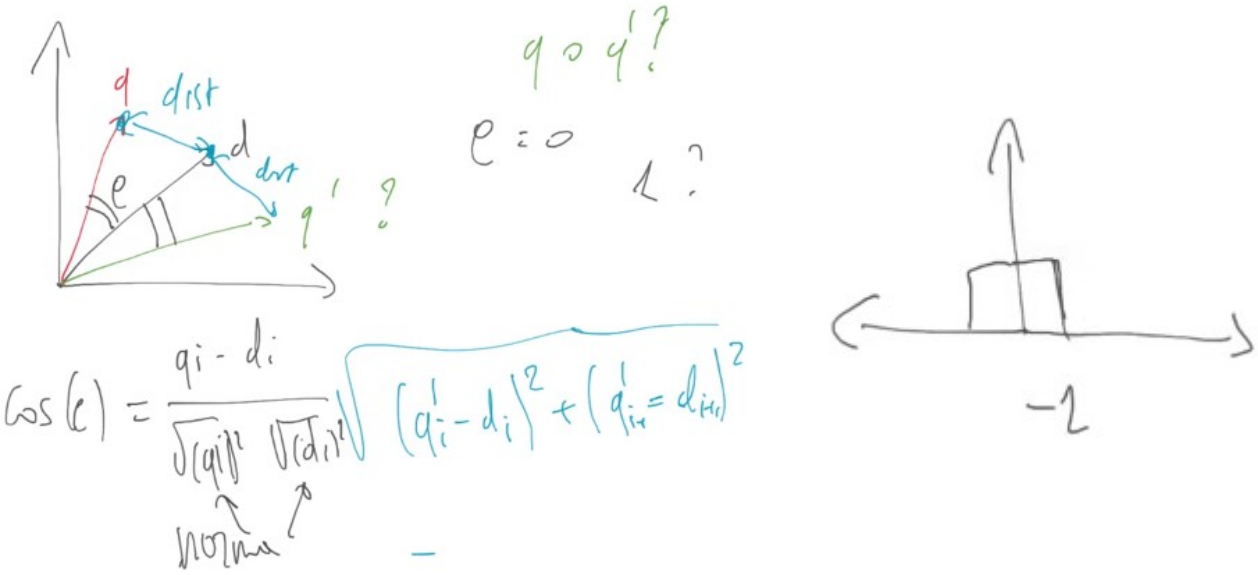

**Teorema di Probability Ranking Principle**: strategia ottimale che permette di ritornare una lista ordinata di documenti in ordine decrescente in base alla rilevanza di predizione, secondo le seguenti assunzioni:
- ogni documento è indipendente da ogni altro.
- la navigazione viene effettuata in maniera decrescente per rilevanza e non casuale.

**Retrival Models**
- *Similarity idea*: cerchiamo documenti che sono simili alla query effettuata dall’utente, basato sul concetto di vicinanza.
- *Probabilistic retrival models*: metodo basato su modelli probabilistici, le queries e i documenti sono osservazioni di variabili casuali, si ha una variabile **R** che indica la rilevanza di una query.



**Bag-of-words**: sistema utilizzato per ottenere le precedenti assunzioni, serve per riassumere un documento scegliendo determinate parole e contandone la occorrenze o la semplice presenza, ma non tiene conto dell'ordine. Ovviamente la *frequenza* dei termini dipende dalla lunghezza del documento, in quanto in un documento più *lungo* compariranno più spesso, infine è presente *l'importanza*/specificità di una parola che viene rapportata al numero di documenti in cui viene trovata, perché se una parola compare in tutti i documenti analizzati allora sarà di poca rilevanza.

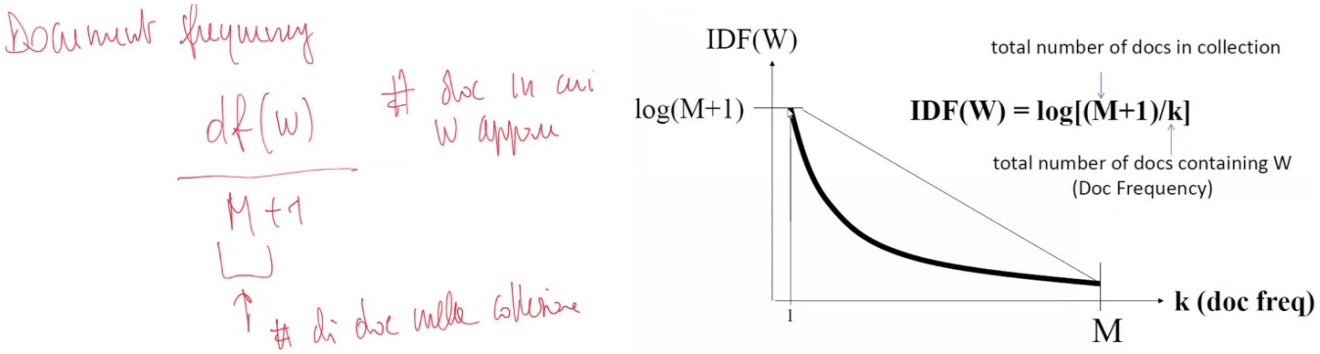
**Vector Space Retrieval Models**: modello che si basa sul concetto si *similarità*, la rilevanza di un documento è correlata alla similarità tra un documento e la query. Creiamo una rappresentazione vettoriale sia del documento che della query, prendendo il Bag-of-words e utilizzandolo come documento per confrontarlo. Per poter calcolare la *distanza* tra i vettori può essere utilizzata la distanza euclidea oppure il *coseno* per calcolare l'angolo racchiuso tra di essi e più questo sarà vicino a 1 questi saranno vicini, viceversa per la vicinanza allo 0, -1 per il valore opposto.



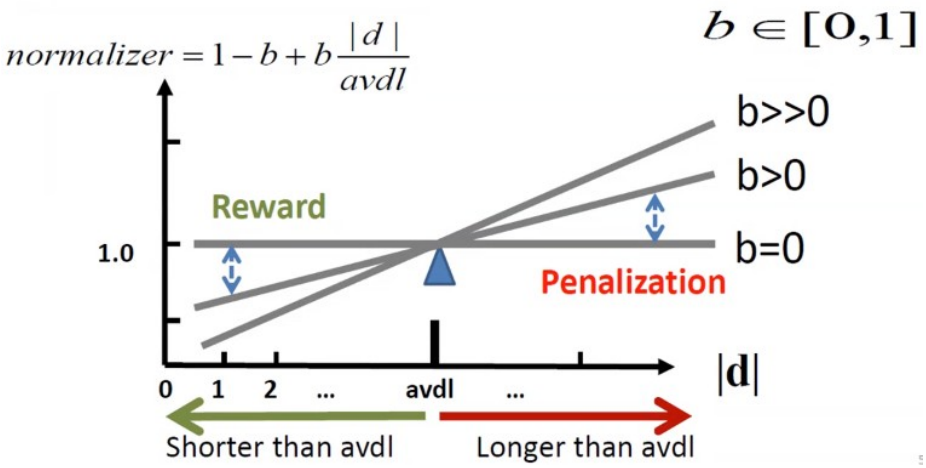

**24/11/2020**

**Improved VSM Instantiation**: per migliorare il modello precedente può essere utilizzata la *term frequency*, ovvero il numero di volte che il termine viene ripetuto.

**Document frequency**: in quanti documenti appare una parola. Una parola molto frequente avrà valore vicino ad 1, viceversa se poco presente sarà vicino a 0. Quindi può essere utilizzato per capire se una parola è molto frequente in ogni documento e di conseguenza ha poca importanza, per calcolarlo può essere diviso il termine della Term frequency per il Document frequency, oppure utilizzando *Inverse Document Frequency*.



**Pivoted Lenght Normalization**: per normalizzare l'importanza delle parole trovate in un documento di grandi dimensioni rispetto ad un documento di dimensioni ridotte. Il valore normalizzatore è contenuto nel termine b.
- b = 1 effettuo la normalizzazione massima.
- b = 0 non effettuo nessuna normalizzazione.




**Probabilistic Retrival Models**: *funzione di ranking* basata sulla probabilità che un documento sia *rilevante* per la query selezionata, la stima può avvenire attraverso un semplice conteggio.

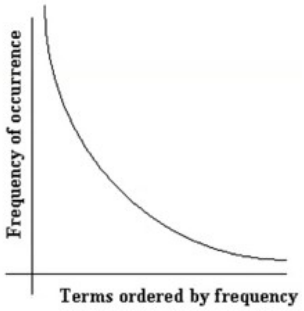
**Query Likelihood**: dato un documento, qual'è la *query ideale* che farebbe l'utente per ottenere uno specifico documento.

$$p(R=1|d,q) \approx p(q|d,R=1)$$

**Legge di Zipf**: esiste una *correlazione tra frequenza e ranking* delle parole, quindi la loro moltiplicazione rimane costante. Infatti ho pochi termini che compaiono molte volte nel documento e viceversa molti termini che compaiono poche volte.

$r * f = k$

$r * P(r) = c$




**30/11/2020**

**Language Models**: modelli utilizzati per rappresentare frasi in NLP, assegnano una probabilità ad una frase, rappresentando quanto una frase è probabile. Quindi data una frase qual'è la probabilità di una parola che segua la frase analizzata.

**Maximum likelihood estimation (MLE)**: stima la probabilità di un termine data la sequenza che lo precede.

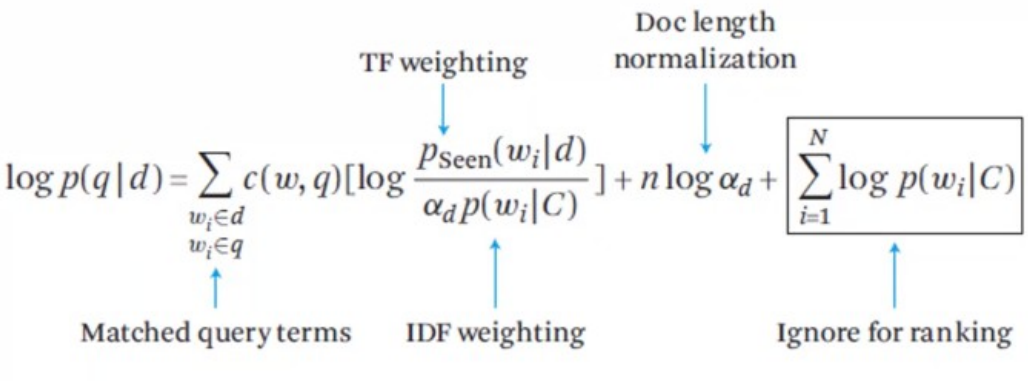
$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

**Stemming**: si eliminano tutte le desinenze, parti finali delle parole che vengono eliminate per normalizzarle, considerando solo la radice.

**Laplace smoothing**: utilizzato per trovare gli elementi che occorrerebbero *zero* volte. Ma facendo ciò andrò a perturbare il conteggio dei valori iniziali. (Sono presenti diversi metodi)

$$P_{Laplace}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)+1}{c(w_{i-1})+V}$$

**Query Likelihood Retrival**




**<u>01/12/2020</u>**

**Query Expansion**: espansione della query iniziale che avviene attraverso l'inserimento di nuove informazioni acquisite dalla selezione dei documenti scelti dall'utente al momento della prima query, così da ottenere maggiore precisione nella ricerca successiva.

**Rocchio feedback:** si allontana dai documenti non rilevanti e viceversa si avvicina a quelli più rilevanti.

$$\vec{q}_m = \alpha \cdot \vec{q} + \frac{\beta}{|D_r|} \cdot \sum_{\vec{d}_j \in D_r} \vec{d}_j - \frac{\gamma}{|D_n|} \cdot \sum_{\vec{d}_j \in D_n} \vec{d}_j,$$

q is the original query vector that is transformed into qm, the modified vector.
Dr is the set of relevant feedback documents and
Dn is the set of non-relevant feedback documents.
The parameters α, β, and γ are weights that control the amount of movement of the original vector.

<u>! Il peso dato dai casi di interesse sono più rilevanti rispetto a quelli meno rilevanti.</u>

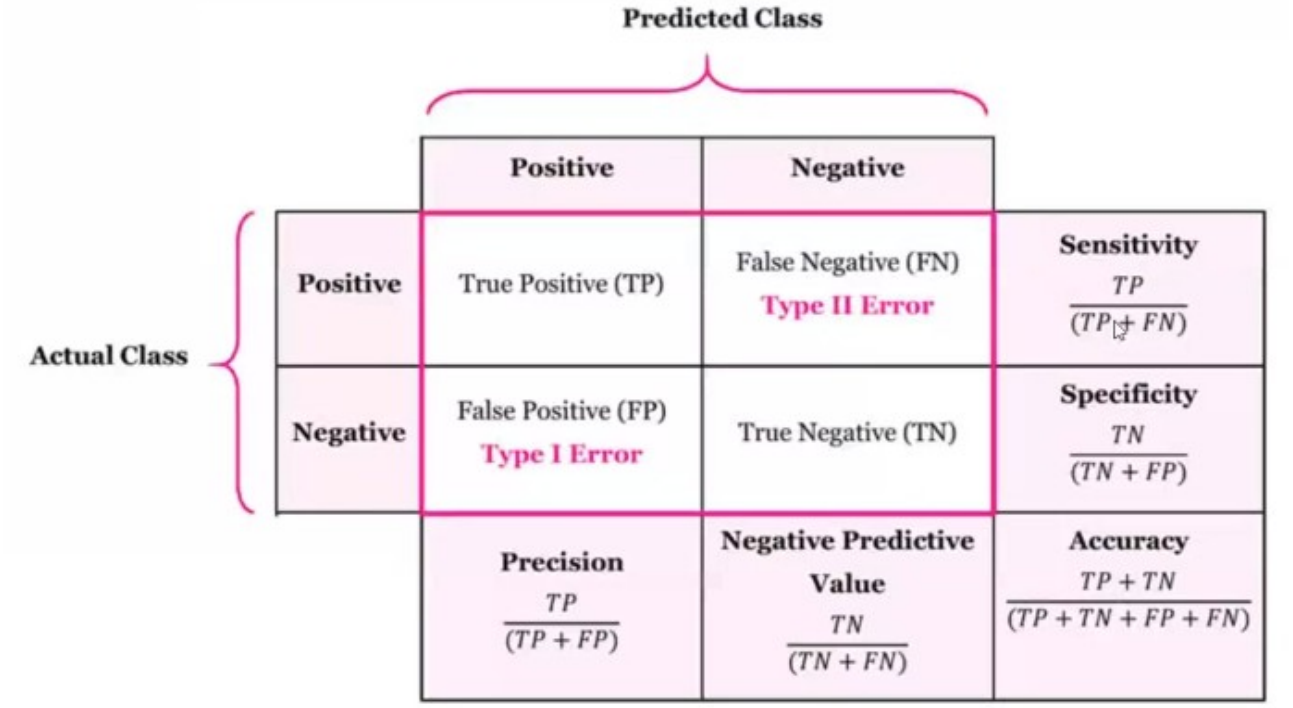
**Search Engine Implementation**
- *Tokenizer*: rappresenta come costruire la matrice (BOW), processo molto complesso dove devo scegliere come costruire le colonne, come dividere i miei dati e quali scelte di rappresentazione utilizzo, come separo le parole.
- *Indexer*: sistema che permette di trovare i documenti che contengono una certa query, ciò che viene tornato dopo la ricerca. Sistema che trasforma un documento in un *indice inverso*, partendo da una parola chiave e trovo la pagina di riferimento. L'indice inverso è composto dal *lexicon*, insieme dei termini indicizzati costruito in modo tale da rimanere in memoria e quindi scorso in modo molto veloce, e dal *posting file*, contiene i dettagli per ogni termine del lexicon.

- *Scorer/Ranker*: applica una delle formule per la similarità dei documenti, rappresentato dal vector space model o sistema probabilistico implementato.
- *Feedback/Learner*: spiegato in precedenza con il Rocchio feedback o più complessi.

**Text Classification**: algoritmi di classificazione applicati al testo.

! Problema del riconoscimento sarcasmo e negazioni di parole positive.

[RIGUARDARE MULTI-LABEL CLASSIFICATION]




**15/12/2020**

**Approcci Collocazionali**: a differenza del BoW utilizzano le parole nelle immediate vicinanze del termine che si vuole analizzare, rendendo più veloce il processo perché altrimenti si dovrebbe scorrere l'intero dataset rendendo il processo molto più lento. Si possono utilizzare *parts-of-speech* che definiscono il ruolo di determinate parole all'interno della frase.

**Algoritmo di Lesk**: approccio che utilizza il dizionario, data una frase contenente parole disambigue (con diversi significati), l'algoritmo conta quante volte le parole nel contesto target sono contenute nelle definizioni fornite da *WordNet*. Tanto più una parola è contenuta e tanto più apparterrà a quel contesto.

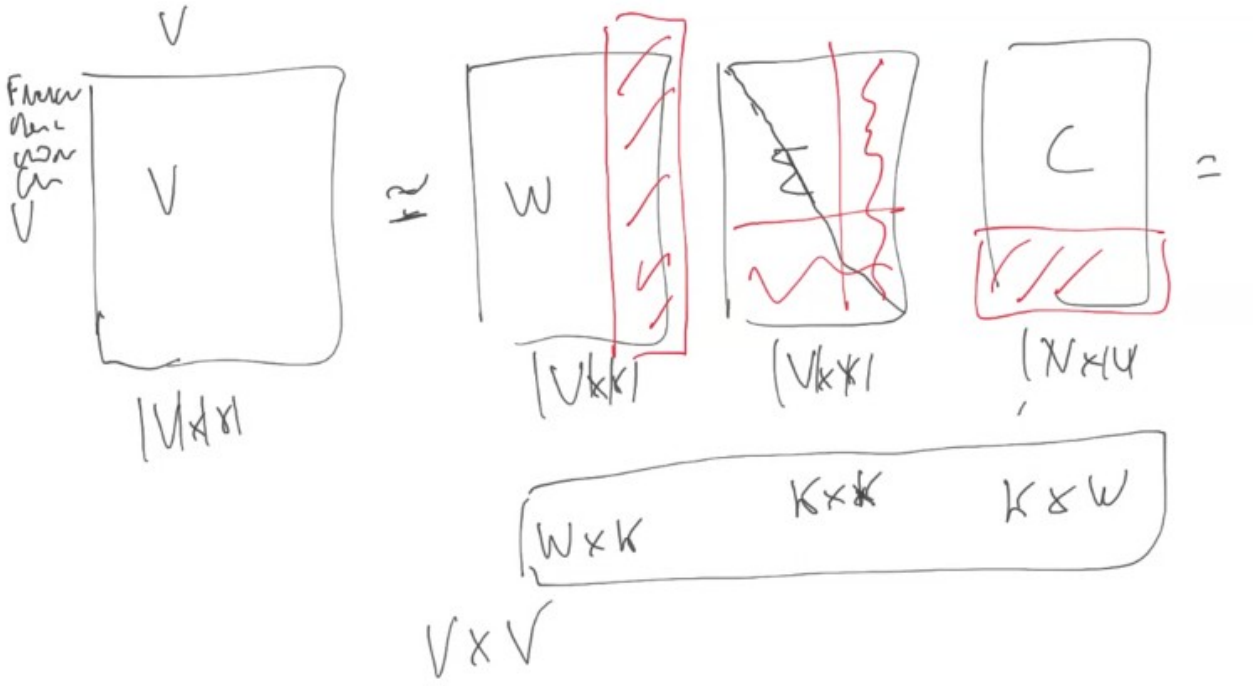
**Graph-based method**: si utilizza un grafo per collegare le parole della frase per trovare il termine centrale, così da trovare il centro del discorso.

**Word Similarity**: tecniche per trovare la similarità delle parole, attraverso un albero di associazione e calcolo la distanza di una parola da un'altra contando i passaggi che devo fare, sempre utilizzando WordNet.

**Embeddings**: vettori che attraverso la vicinanza delle frasi, riescono a rappresentare una determina parola, capendone il significato. Questo concetto si basa sulla correlazione e utilizza l'encoding, quindi parole simili avranno un valore simile.

**21/12/2020**

**SVD method**: ottengo una matrice più confusa e generalizzata della precedente, approssimazione utilizzata per ottenere una matrice più densa e con gli stessi valori della iniziale, ottengo una matrice con lo stesso numero di righe ma con numero di colonne minore.




! Problemi degli algoritmi di machine learning nella produzione, riguardano il problema dei dati che devono essere controllati se no si può avere un decadimento molto rapito (**Tecnical Debt**). Bisogna inoltre prestare attenzione su quale porzione di dati mantenere allenato l'algoritmo.

**22/12/2020**

**Sistemi di Raccomandazione**: si basano sulla matrice di utilità, composta sulle righe dagli utenti, sulle colonne dagli elementi/items e nelle intersezione il rating, si tratta quindi di una matrice sparsa perché nessun utente potrà mai aver avuto contatto con tutti gli elementi contenuti. L'obiettivo è predire alcuni di questi valori non conosciuti, quelli più significativi. Dalla matrice possono essere estratte implicitamente informazioni.
- **Content-based systems**: utilizzati estraendo caratteristiche, riguardanti il contenuto, dal prodotto analizzato per effettuare conclusioni su prodotti simili.
- **Collaborative filtering**: utilizzati per trovare utenti simili e quindi trarre conclusioni sui dati mancanti. Serve per suggerire prodotti che sono piaciuti ad un utente con caratteristiche simili.

| | Pros 👍 | Cons 👎 |
|---|---|---|
| Collaborative | No knowledge-engineering effort, serendipity of results, learns market segments | Requires some form of rating feedback, cold start for new users and new items |
| Content-based | No community required, comparison between items possible | Content descriptions necessary, cold start for new users, no surprises |
| Knowledge-based | Deterministic recommendations, assured quality, no cold-start, can resemble sales dialogue | Knowledge engineering effort to bootstrap, basically static, does not react to short-term trends |

**Duality of Similarity**: al posto di trovare la similarità tra gli utenti trovo la similarità tra gli item, calcolo questa similarità nei tempi "morti" e così sarò molto più veloce nel momento di registrazione di un nuovo utente a cui devo dare suggerimenti.

**User-Based reccomandation**: raccomandazioni basate sulla similarità tra utenti, attraverso diverse formule e successivamente posso calcolare con altre formule la relativa predizione.

$$sim(a,b) = \frac{\sum_{p \in P}(r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P}(r_{a,p} - \bar{r}_a)^2}\sqrt{\sum_{p \in P}(r_{b,p} - \bar{r}_b)^2}}$$

| | Item1 | Item2 | Item3 | Item4 | Item5 |
|---|---|---|---|---|---|
| Alice | 5 | 3 | 4 | 4 | ? |
| User1 | 3 | 1 | 2 | 3 | 3 |
| User2 | 4 | 3 | 4 | 3 | 5 |
| User3 | 3 | 3 | 1 | 5 | 4 |
| User4 | 1 | 5 | 5 | 2 | 1 |

sim = 0,85
sim = 0,70
sim = -0,79

$$pred(a,p) = \overline{r_a} + \frac{\sum_{b \in N} sim(a,b) * (r_{b,p} - \overline{r_b})}{\sum_{b \in N} sim(a,b)}$$