

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 Princípio 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
3.11 Come trovo un modello che non overfitta?	20

3.12	Come posso aiutare il modello contro l'overfitting?	20
3.13	Valutazione del modello	20
3.13.1	F1 score	21
3.13.2	Multilabel Confusion Matrix	21
3.14	Ensemble learning	21
3.14.1	Bagging	21
3.14.2	Boosting	21
4	Discovering frequent itemsets	23
4.1	Market-Basket model	23
4.1.1	Principio di monotonicità	23
4.1.2	Confidence	24
4.2	Representation of the Market-Basket data	24
4.2.1	Triangular Matrix e Triplets	24
4.2.2	A-Priori Algorithm	25
4.2.3	Park, Chen, Yu Algorithm	25
4.2.4	Limited Pass Algorithm	26
4.2.5	Toivonen's Algorithm	26
4.2.6	SON Algorithm (Savasere, Omiecinski and Navathe)	27
4.3	Finding Similar Items	28
4.3.1	Shingling	28
4.3.2	Minhashing	29
4.3.3	Minhashing optimization	31
4.3.4	Locality-sensitive hashing	31
5	Text Analysis	35
5.1	Text Data Access	35
5.2	Text Retrieval	35
5.2.1	Document Selection vs Document Ranking	35
5.3	Retrieval Models	36
5.3.1	Vector Space Retrieval Model	36
5.4	Probabilistic Retrieval Models	38
5.4.1	Text statistics: Zipf's Law	38
5.4.2	Language Model	38
5.5	Probabilistic Retrieval Models II	41
5.5.1	The Query Likelihood Retrieval Model	41
5.5.2	Smoothing della probabilità	42
5.6	Search Engine Implementation	43
5.6.1	Tokenizer e Indexer	43
5.6.2	Scorer e Learner	44
5.6.3	Rocchio Feedback	44
5.7	Text classification	45
5.7.1	Computational Lexical Semantics	45
5.7.2	Vector semantics and embeddings	46
5.7.3	Sparse vector representations: Term-document matrix	47
5.7.4	Sparse vector representations: word-word or term-context matrix	47
5.7.5	Positive Pointwise Mutual Information	47
5.7.6	Singular Value Decomposition (SVD) e Latent Semantic Analysis (LSA)	48

5.7.7	Word2vec	48
5.8	Neural Networks Basics	49
5.8.1	Feed Forward Neural Networks	49
5.9	Recurrent Neural Networks	50
5.9.1	Sequence classification	50
5.10	Contextualized embeddings	50
5.10.1	Transformers	50
5.10.2	Multihead attention mechanism	51
6	Explainable AI	53
6.1	Interpretability	53
6.1.1	Interpretable Models	53
6.1.2	Partial Dependence Plot	54
6.1.3	Permutation Feature Importance	54
6.1.4	Leave-One-Covariate-Out	54
6.1.5	Local Surrogate (LIME)	55
6.1.6	Counterfactual Explanation	55
6.1.7	Shapley Values	55
6.1.8	KernelSHAP	56
7	MLOps	57
7.1	Project Life-Cycle	57
7.2	Perchè i progetti di ML falliscono?	57
7.2.1	Mettere un modello in produzione: mlops	58
7.2.2	Problemi con i dati	58
7.2.3	Training and evaluation	58
7.3	Fairness in ML	59
7.3.1	No fairness through unawareness	59
7.3.2	Pitfalls of action	59
7.3.3	Statistical non-discrimination criteria	59
7.3.4	Separation	60
7.3.5	Sufficiency	60
7.4	Come rimuoviamo gli stereotipi?	60

Generalities and Map Reduce

1 Generalities and Map Reduce

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

Motivation: Google Example

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

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

1.1 Introduction to Distributed File Systems

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

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

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

1.2 Map Reduce

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

1. Map
2. Group by key
3. Reduce

map

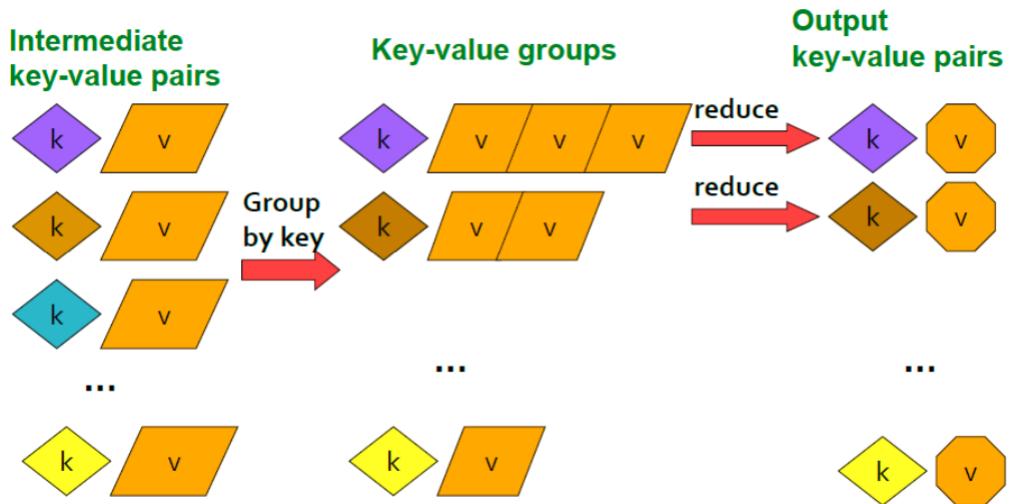
Il "map" è il primo step dove essenzialmente vado alla ricerca delle informazioni utili. Corrisponde di fatto ad una query dove isolo elementi secondo una certa caratteristica. *Extracting something of interest*

group by key

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

reduce

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



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

1.3 Distributed File System Architecture w/ Map Reduce

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

Distributed File System

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

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

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

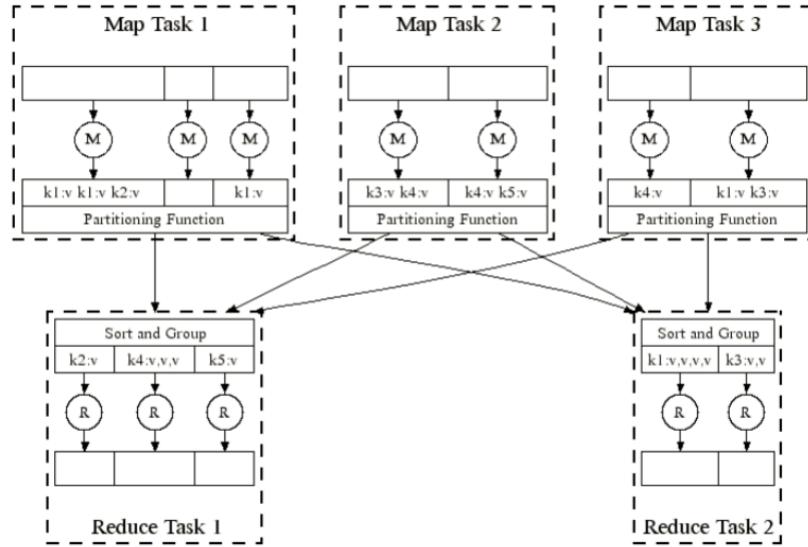
- Map worker: le task del worker sono resettate, i reduce workers vengono notificati quando la task viene compiuta da un altro worker
- Reduce worker: solo le task in-process del worker vengono resettate, la task di reduce viene restartata
- Master failure: MapReduce task abortita e contattato il client

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

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

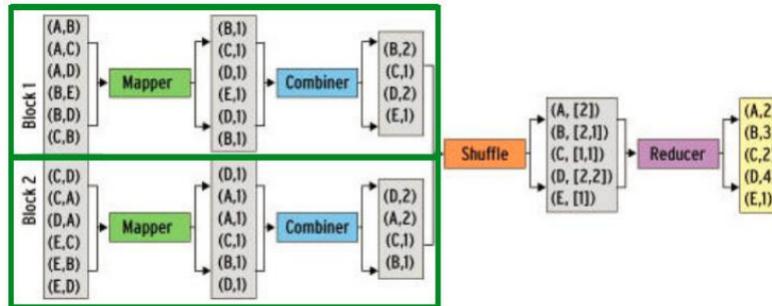
1.4 MapReduce in parallelo

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



1.5 Combiners

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



1.6 Remarks

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

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

1.7 Algoritmi che usano Map Reduce

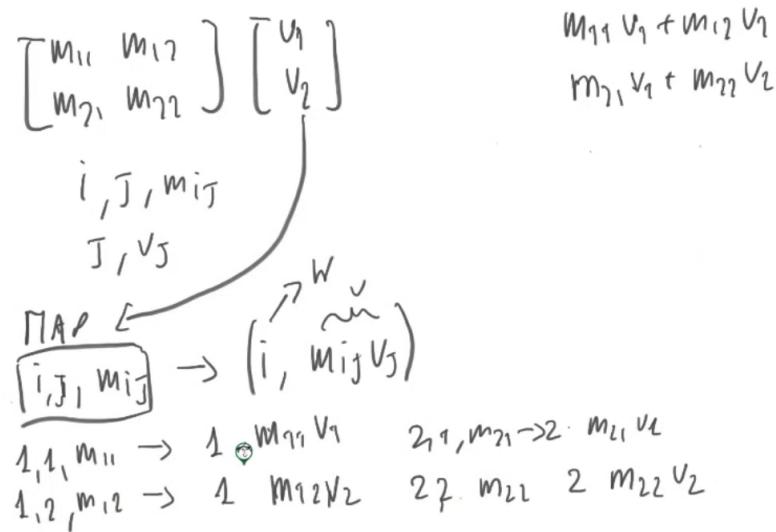
Elevata probabilità di essere chiesto. Esistono diversi algoritmi che sfruttano il map reduce, come la moltiplicazione vettoriale oppure le operazioni di algebra relazionale.

1.7.1 Matrix-Vector multiplication - Vettore che sta in memoria

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

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

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



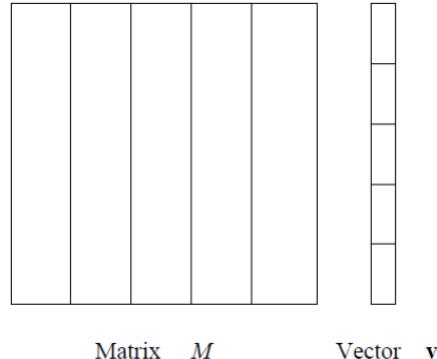
Ora abbiamo il Group By Key Task, che funziona esattamente come spiegato: mette insieme gli elementi che hanno stessa chiave. La chiave, in questo caso, è la **riga della matrice**. Quindi, dopo la Reduce Task che fa la somma e osservando sempre l'immagine, il risultato che otterremo sarà:

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

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

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

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



1.7.3 Relational algebra operations - Selection

$\sigma_C(R)$

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

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

1.7.4 Relational algebra operations - Projection

$\pi_S(R)$

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

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

1.7.5 Relational algebra operations - Union

$UNION(R, S)$

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

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

1.7.6 Relational algebra operations - Intersection

$INTERSECTION(R, S)$

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

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

1.7.7 Relational algebra operations - Difference

Difference($R - S$)

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

1.7.8 Relational algebra operations - Natural Join

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

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

1.7.9 Relational algebra operations - Grouping and Aggregation

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

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

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

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

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

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

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

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

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

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

1.8 Remarks

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

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

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

1.8.1 Reducer size q

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

1.8.2 Replication rate r

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

Relazione tra r e q

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

1.8.3 Similarity Join

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

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

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

Datamining and Machine Learning

2 Data Mining

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

2.1 Principio di Bonferroni

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

2.1.1 Esempio dei malfattori

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

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

Otteniamo, facendo delle stime di probabilità:

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

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

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

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

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

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

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

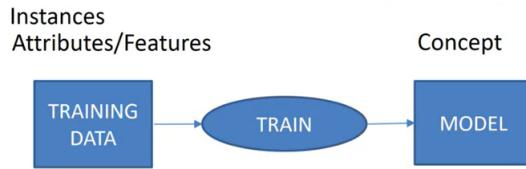
$$5 \times 10^{17} \times 5 \times 10^5 \times 10^{-17} = 250'000$$

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

3 Machine Learning

3.1 Perchè utilizzare il Machine Learning?

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



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

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

3.2 Tipologie di apprendimento

Tra le più famose elenchiamo (in ordine sparso):

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

3.3 Risultato

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

In sostanza, il Machine Learning è utile per:

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

3.4 Tipologie di modelli

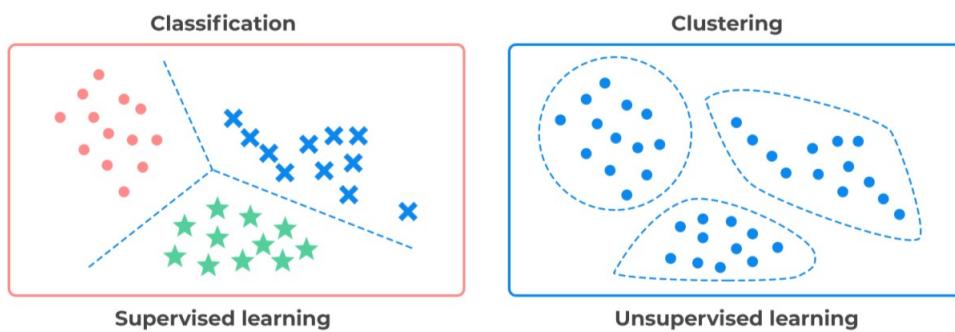
Un modello di ML quando si distingue per vari criteri:

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

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

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

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



3.5 Regressione

3.5.1 Regressione lineare

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

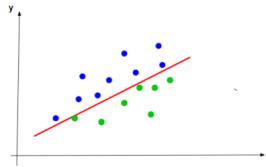
$$\text{life_satisfaction} = \theta_0 + \theta_1 \times \text{PIL}$$

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

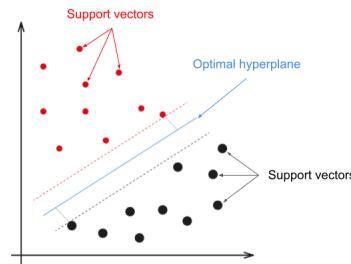
- θ_0 è il bias term, o intercept term
- θ_1 è il peso
- PIL (se vogliamo esprimere un generico valore, metteremo x) indica la nostra input feature
- life_satisfaction (y) sarebbe il valore predetto

E ricadiamo nella categoria di **regressione**.

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



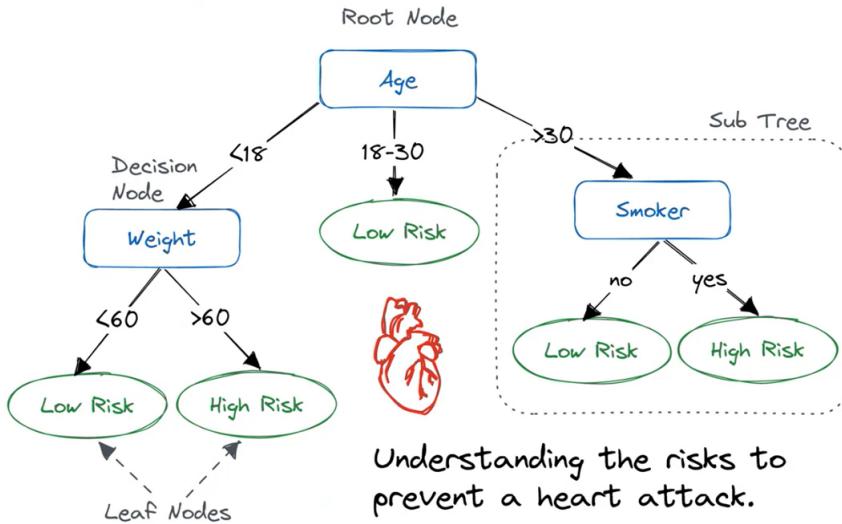
Un esempio pratico è il modello SVM, che sfrutta il concetto di *support vector* per separare linearmente i dati. Entrando nello specifico (ma non troppo) si tratta di *large margin classification*: immagina il classificatore SVM come un modello che cerca di creare la più grande zona di separazione tra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



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

3.5.2 Decision Tree

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



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

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

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

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

L'albero parte da una feature, Age:

In quale gap di età si trova il paziente?

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

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

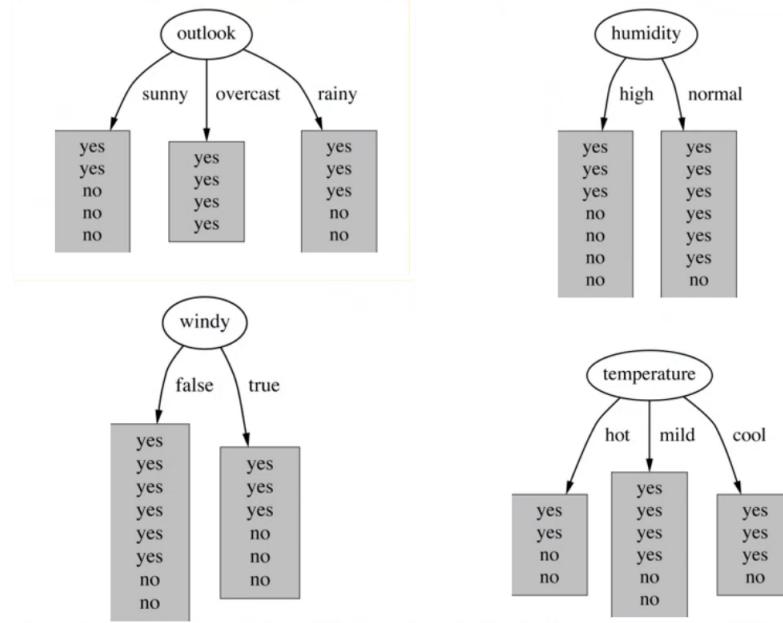
Analizzando in età minore di 18, nota che c'è una feature che fa pendere l'ago della bilancia: Weight (peso): crea quindi due leaf nodes con rischio alto e basso sulla base di tale valore.

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

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

3.5.3 Costruzione di un Decision Tree

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



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

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

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

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

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

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

- L'informazione attesa è:

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

- Mentre l'info gain effettiva di Outlook è:

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

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

3.5.4 Simple Algorithm

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

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

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

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

3.6 Overfitting

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

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

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

3.6.1 i.i.d. assumption per Statistical Modeling

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

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

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

3.7 Instance based learning

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

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

3.8 One rule algorithm

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

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

* Random choice between two equally likely outcomes

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

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

3.9 Naive Bayes Classifier

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

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

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

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

Likelihood of the two classes

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

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

Conversion into a probability by normalization:

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

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

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

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

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

Le normalizzo a uno:

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

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

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

3.9.1 Regola di Bayes

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

Questa regola viene descritta nel modo seguente:

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

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

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

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

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

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

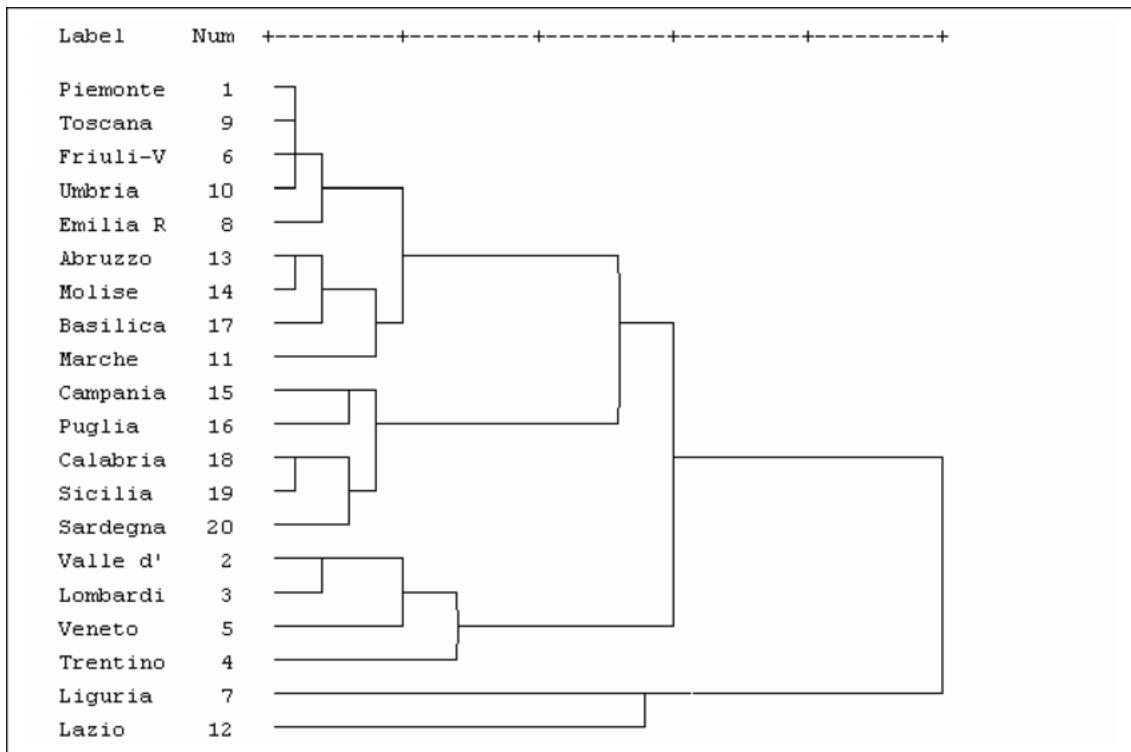
Qual è il problema di questo approccio? Che se la probabilità di un'evidenza dato l'evento è 0, mi porta tutto a 0.

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

3.10 Unsupervised Learning

3.10.1 Clustering

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



3.11 Come trovo un modello che non overfitta?

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

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

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

3.12 Come posso aiutare il modello contro l'overfitting?

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

3.13 Valutazione del modello

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

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

Sulla base di questa tabella, possiamo definire due parametri:

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

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

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

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

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

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

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

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

3.13.1 F1 score

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

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

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

3.13.2 Multilabel Confusion Matrix

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

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

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

3.14 Ensemble learning

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

3.14.1 Bagging

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

3.14.2 Boosting

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

Frequent Itemsets

4 Discovering frequent itemsets

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

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

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

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

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

4.1 Market-Basket model

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

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

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

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

4.1.1 Principio di monotonicità

Regola che limita il numero di frequent itemsets:

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

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

4.1.2 Confidence

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

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

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

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

4.2 Representation of the Market-Basket data

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

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

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

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

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

4.2.1 Triangular Matrix e Triplets

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

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

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

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

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

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

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

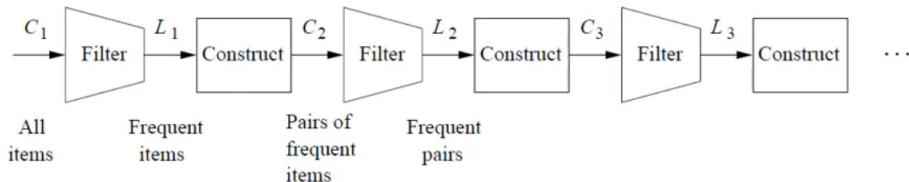
- **Triplets:** la tecnica delle triplettte consiste nella creazione di una tripletta composta da i due valori numerici e il conteggio: $\{1, 2, 3\}$ ma è svantaggioso rispetto alla matrice poichè per una coppia salva 3 interi invece di uno solo. Ha però un vantaggio enorme sulla matrice: tratta le coppie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

4.2.2 A-Priori Algorithm

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

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

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



4.2.3 Park, Chen, Yu Algorithm

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

Esempio. Immaginiamo di avere questa serie di elementi:

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

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

4.2.4 Limited Pass Algorithm

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

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

4.2.5 Toivonen's Algorithm

Trova i risultati in queste due condizioni:

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

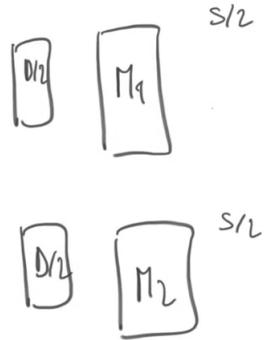
Come funziona? Sfrutta il concetto di *negative border*. Gli itemset non frequenti formati da elementi frequenti (più corretto sarebbe dire itemset non frequenti, i cui sottoinsiemi sono frequenti). Prendiamo solo una porzione, aggiorniamo la soglia, e calcoliamo gli elementi frequenti come visto nel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

4.2.6 SON Algorithm (Savasere, Omiecinski and Navathe)

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

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

Esempio. Immaginiamo di avere due chunks:



Con le relative soglie. M1 mi estrae 4 elementi frequenti F1, F4, F7 e F9 mentre M2 mi estrae FA, F1, F4. Ora prendo l'insieme degli elementi e faccio controllare se sono frequenti in entrambe le macchine. Praticamente, passo ad M1 e M2 l'insieme di valori complessivo e conto le occorrenze in uno e nell'altro caso, se superano la soglia li posso tenere.

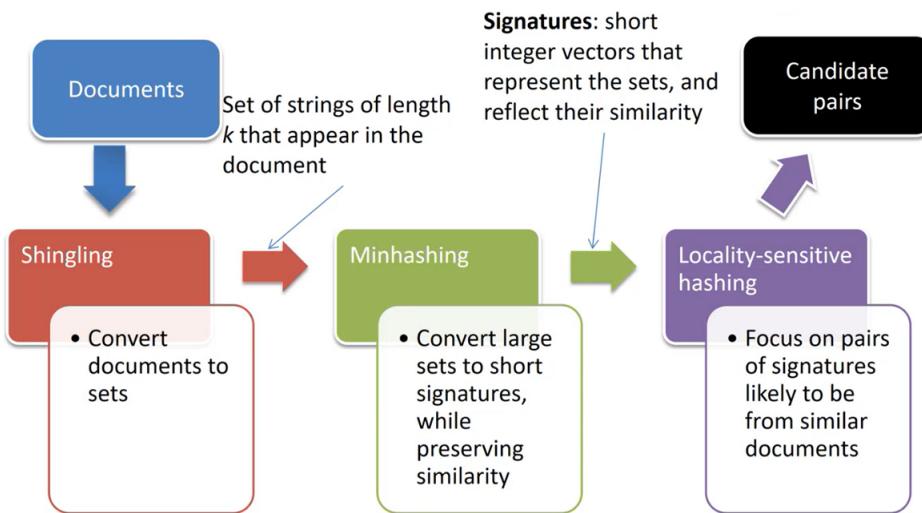
Approccio con MapReduce. Il primo map mi analizza un pezzo e mi trova quelli frequenti. Mi emette una coppia chiave valore con il valore frequente e una chiave associata: per esempio. potrebbe venire una cosa di questo tipo (F1, 1), (F2, 1). Il reducer non fa assolutamente niente, butta fuori le coppie esattamente in questo modo per fare prima. Conviene piuttosto fare due cicli piuttosto che fare tutto in una volta.

Secondo step: altro mapreduce, mettiamo M1 e M2 ma questa volta con gli elementi frequenti: con il combiner possiamo contare quante occorrenze abbiamo in ogni macchina.

4.3 Finding Similar Items

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

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



Tutti i passaggi verranno spiegati nello specifico qui di seguito.

4.3.1 Shingling

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

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

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

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

Se voglio confrontare due documenti, andrò a scrivere:

	D_1	D_2
1	0	1
0	1	1
1	1	1
1	1	1
1	1	0
0	0	0
1	1	0

Dove tutti quei 0 e 1 rappresentano gli shingle per ogni documento, calcolerò la similarità di Jaccard facendo numero elementi in comune diviso numero elementi totali in esame. Stabilisco una soglia, se la superano sono simili. Ma come posso fare se devo confrontare molti documenti?

4.3.2 Minhashing

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

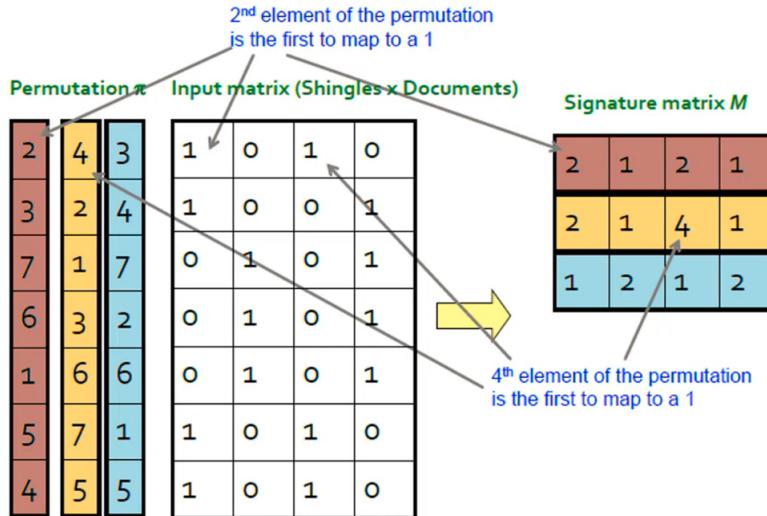
Un primo approccio, tornando all'esempio precedente della tabella in alto: con valori booleani, posso trattarli da tali, quindi calcolarne il valore in decimale per ogni riga (ad esempio, 101 sarebbe 5) e riduco tanto il numero di righe (attenzione! la compattazione avviene lungo la colonna, non lungo le righe quindi in verticale!).

Ma c'è un problema. Compattare tra di loro degli shingle, potrebbe crearmi combinazioni di lettere improbabili e trovare due documenti con tale sequenza simile diventa impossibile.

Altra soluzione. Torniamo ad utilizzare la funzione di hash. Numeriamo le righe: voglio convertire quella matrice in una matrice più corta. Facciamo un esempio con una matrice da un milione di righe, voglio trasformarla in una matrice da mille righe. Dopo aver dato un indice ad ogni riga, lo divido per mille e calcolo il resto; se in quel punto della matrice poi c'è un uno, nella matrice ridotta ci sarà un uno. Guardando sempre l'immagine in alto: l'elemento 1 diviso mille fa 0 con resto di 1. Quindi nella nuova matrice andrò a mettermi nella posizione 1, e ora devo guardare se è uno 0 o un 1 nella prima matrice: è 1. Quindi nella posizione 1 della nuova matrice andrò a mettere 1.

La stessa identica cosa accadrà per il valore 1001, ad esempio, finirà anche lui nella posizione 1. Quindi sto mappando all'interno della stessa cella, più shingle; ciò genera errore. Vuole dire che se viene mappato all'interno della cella uno shingle frequent come può essere 'to' in lingua italiana, insieme allo shingle 'zy' stiamo dando valore 1 anche ad uno shingle non-frequent, ma non è un problema dato che stiamo accorciando il numero di righe.

Minhashing. Sfruttiamo quindi il concetto di hashing per trovare una matrice ridotta, che mantenga la similarità di Jaccard. Creiamo le signature:



Costruiamo una permutazione delle righe, le righe vengono cioè rimescolate (nell'immagine, l'ordine permutato è dato dalla colonna rossa, gialla e azzurra). Fatta la permutazione, vado a costruire una matrice di signature in cui il numero che ho è la prima riga in cui compare un 1 in tale permutazione. Ad esempio, nell'immagine, la prima cella della rossa è 2, perchè compare un 1 nella sua riga 2 (che è la riga 1 della matrice originale). Ovviamente le colonne della matrice di signature corrispondono ai documenti, come nella matrice iniziale. Quindi quello che voglio è che la similarità di Jaccard calcolata per i due documenti sia uguale alla probabilità che le signature nella matrice ridotta siano simili:

$$SIM_j(D1, D2) = P(S(D1) = S(D2))$$

Dove S è la signature. Mi aspetterò che due documenti simili, abbiano signature simili.

Dimostrazione. Vogliamo dimostrare la similarità tra due documenti aventi simili signature.

	<u>C₁</u>	<u>C₂</u>
X	1	1
Y	1	0
Y	0	1
Z	0	0

Distinguo tra righe X (entrambi 1) righe Y (diversi) e righe Z (entrambi 0). Sappiamo che un calcolo possibile è ridurre il numero di righe eliminando quelle di tipo Z inutili nel calcolo della similarità. Al numeratore della similarità di Jaccard avremo X dove sono simili e al denominatore le righe X + Y.

Consideriamo una permutazione, e non calcoliamo le righe Z. Qual è la probabilità togliendo gli Z di avere una X prima di una riga Y? (ricordati come viene costruita la matrice di signature) Essa è:

$$\frac{X}{X+Y}$$

Ma se abbiamo una X prima delle altre, cosa succede nel minhashing? Che i due documenti avevano lo stesso valore. Quindi la probabilità che i documenti avessero lo stesso numero, è pari alla similarità tra i due documenti.

4.3.3 Minhashing optimization

A lezione sono stati spiegati un paio di metodi per automatizzare e migliorare il processo di Minhashing, qui di seguito ne vediamo uno.

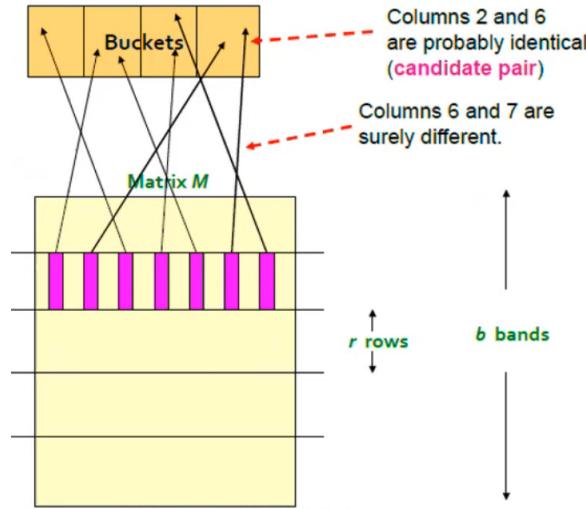
Dopo la permutazione, noi abbiamo un numero di righe ridotto, con un valore di similarità uguale. Non è necessario permutare tutte le righe, posso anche semplicemente fare una permutazione di k righe dove k è minore del numero di righe n. L'unico caso in cui questa operazione può risultare deleteria, sarebbe nel caso in cui alcuni valori non dovessero trovare una corrispondenza; ma per la costruzione della matrice di hashing, in corrispondenza di quei valori troveremmo allora un infinito, che è comunque diverso da qualunque altro valore nella matrice:

2	1	2	1
2	1	4	1
1	2	1	2

Se considerassi k = 3, quel 4 non ci sarebbe, e rimarrebbe comunque un elemento che non porta informazione essendo diverso da tutti gli altri. Potrebbe valere 4 o 1000 che non mi interesserebbe. C'è però un problema. Se scelgo un k troppo basso (ad esempio nella matrice qua sopra un valore critico potrebbe essere k=2) la matrice potrebbe perdere troppe informazioni e con troppi valori ad infinito non posso lavorarci perchè potrebbe capitare che se due documenti erano diversi, considerando k molto basso potrebbero perdere molti elementi diversi che verrebbero sostituiti da degli infiniti diventando uguali e trasformando documenti non simili in documenti simili.

4.3.4 Locality-sensitive hashing

Terzo step: ora che ho ridotto le righe, voglio poter fare il numero più ridotto possibile di confronti tra le colonne. Con LSH lavoriamo sulla matrice di minhashing, per poi andare ad utilizzare una funzione che candidi delle coppie. Creo dei gruppi di documenti che possibilmente sono coppie di elementi simili. Come li trovo? Ad esempio potrei calcolare la somma nella matrice di minhashing dei valori di ogni documento, e quelli con valori di somma simile sono documenti simili. Quale può essere il problema di questa funzione? Che basta che cambi uno dei valori e categorizza diverso (ad esempio documento 2,2,1 come nell'immagine e 2,4,1). Quindi posso avere dei falsi negativi. Mentre i falsi positivi non sono un grosso problema: i valori di somma possono venire uguali, ma poi vanno osservati sul documento. LSH compensa quindi il problema dei falsi negativi suddividendo in bande la matrice e applicando la funzione di hash ad ogni banda. Basta infatti che anche solo una banda abbia gli stessi valori che vado a finire nello stesso bucket.

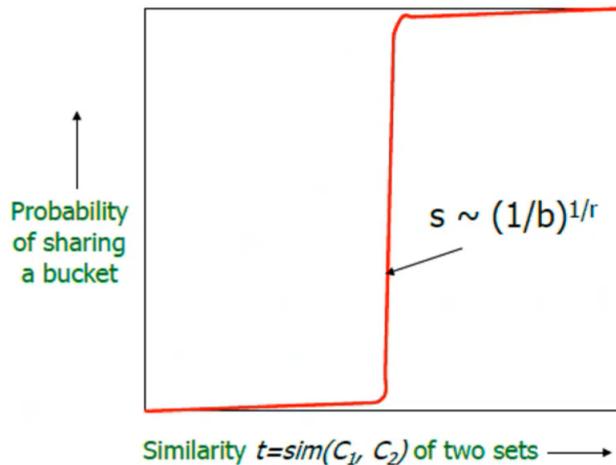


Questo procedimento diminuisce il costo dei confronti significativamente tanto.

Importante. Due colonne vanno nello stesso bucket quando hanno gli stessi valori ovvero quando i due documenti hanno un'alta similarità di Jaccard. La probabilità che le signatures abbiano lo stesso valore in una banda (e diventino quindi coppie candidate) è:

$$1 - (1 - s^r)^b$$

Dove s è il valore di similarità, r il numero di righe e b il numero di bande. Quindi possiamo manipolare questo valore:



Se uso b e r uguali a 1, la probabilità di essere nello stesso bucket è uguale alla similarità, il che mi va bene ma per ridurre il numero di falsi negativi, devo cambiare funzione.

Esempi. Troviamo coppie con $s = 0.8$, mettiamo $b=20$ e $r=5$. Facendo con dei documenti C_1 e C_2 la cui $\text{sim}(C_1, C_2) = 0.3$ vedremo:

- probabilità che C_1 e C_2 siano identici in una banda = $0.3^5 = 0.00243$
- probabilità che C_1 e C_2 siano uguali in almeno 1 delle 20 bande = $1 - (1 - 0.00243)^{20} = 0.0474$

Di conseguenza, la probabilità di avere delle coppie candidate con 0.3 similarità, sarà intorno al 4%, valore accettabile. Il caso opposto sarebbe pensare alle coppie con 0.8 similarità:

- probabilità che C1 e C2 siano identici in una banda = $0.8^5 = 0.328$
- probabilità che C1 e C2 siano uguali in almeno 1 delle 20 bande = $1 - (1 - 0.328)^{20} = 0.00035$

Questo significa che circa 1 coppia su 3000 che esaminiamo (con 0.8 o più di similarità) viene considerata falso negativo. Anche questo errore è più che accettabile (troviamo il 99.965% di coppie nei documenti simili).

Text Analysis

5 Text Analysis

5.1 Text Data Access

L'obiettivo principale del text data access è di connettere user con le giuste informazioni al momento giusto. Questa operazione può essere svolta in due modi:

- pull, dove l'user prende l'iniziativa di cercare le info ed estrarrele dal sistema
- push, dove il sistema prende l'iniziativa e offre informazioni all'user.

Ci sono due modi per interrogare una collezione di dati, il *search engine* e *sistema di raccomandazione*. Nel primo caso siamo noi a dare la query. Nell'altro caso, è tipo su netflix che consiglia delle serie. Quindi pull è il search engine, push è raccomandazione. Prende il nome di *Text retrieval* quello che noi abbiamo appena descritto come search engine, ovvero trovare un'informazione tramite query. Problema solitamente complesso: non è una semplice query in un database che fornisce un risultato esatto. La risposta desiderata è anche soggettiva, dipende dalla correttezza della query scritta dall'utente (basta pensare alla ricerca su Google e come cambiano i risultati sulla base di ciò che si cerca).

5.2 Text Retrieval

Data una collezione di documenti, il text retrieval può essere definito come: utilizzando una query dell'user, (per esempio, una *descrizione* dell'informazione ricercata) identificare un subset di documenti che soddisfano la richiesta.

- Abbiamo V vocabolario di tutte le parole di un natural language.
- q , query dell'user, dove $q_i \in V$
- d_i , documento tale che $d_{ij} \in V$
- la query è più corta del documento
- una collezione C è l'insieme di documenti d
- assumiamo che esista un sottoinsieme R di documenti, che sono rilevanti per la query q

Il sottoinsieme perfetto $R(q)$ non è sempre il risultato. Per correttezza, considereremo un $R'(q)$, come un'approssimazione del risultato esatto $R(q)$ comunque accettabile.

Come possiamo trovare quindi $R'(q)$? Esistono due metodi: *document selection* o *document ranking*.

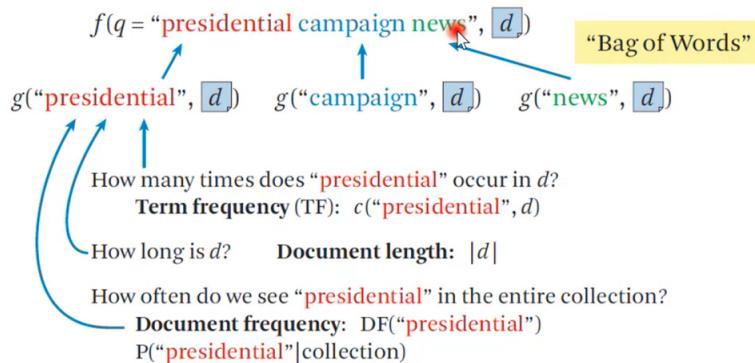
5.2.1 Document Selection vs Document Ranking

La document selection utilizza un classificatore binario per classificare se un documento è relevant o no rispettando una particolare query. Usando questa strategia, il sistema capisce l'effettiva rilevanza di un documento.

Nel document ranking, abbiamo una funzione di ranking e l'user sceglie un limite. In questo caso, il sistema impara una rilevanza relativa: quali documenti potrebbero essere rilevanti. Per il principio del probability ranking, fare un ranking decrescente di rilevanza fornisce una informazione utile se l'utilità di un documento per l'user è indipendente dall'utilità degli altri documenti e se l'user potrà guardare la lista dei risultati sequenzialmente.

5.3 Retrieval Models

Esistono diversi modelli per fare retrieval, e si distinguono per tipologia di approccio al problema. Alcuni sono basati sempre sul concetto di similarità, altri lavorano sul piano probabilistico. Molti modelli si basano anche sulla consapevolezza che i documenti di testo vengono rappresentati come bag of words, non shingles. Si sceglie un vocabolario (lingua) di riferimento e si usa quello.

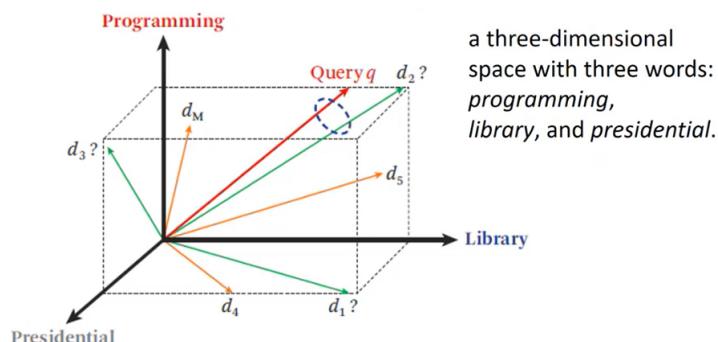


Ma allora come faccio a decidere quale documento è più importante? Per esempio posso contare le occorrenze della parola ricercata, oppure tenere conto quanto è lungo il documento, poiché più parole ci sono, più è probabile che almeno una delle parole della mia query compaia nella ricerca; un ultimo valore che è necessario conoscere sono le stop words: parole "inutili". Quando, quanto ecc... sono parole che non ci interessano quindi le eliminiamo. Poi ci sono parole importanti, che dipendono dal dominio della ricerca. In un documento di calcio, "Goal" è una parola molto frequente, meno discriminatoria nella ricerca. Ma una parola come "Messi", è molto più specifica. Questo effetto prende il nome di document frequency (spiegato meglio in seguito).

Le metriche più importanti sono sicuramente **term frequency** e **document frequency** quando si parla della funzione di ranking (che chiameremo TF e DF).

5.3.1 Vector Space Retrieval Model

Modello molto semplice basato sulla similarità che trasforma i documenti della collezione in vettori e anche la query dell'utente.



Vado ad orientarli nello spazio e guardo quelli che sono più vicini al vettore della query. Per esempio nell'immagine parliamo di d2.

Come trasformiamo una query in un vettore? Stabilisco un vocabolario, di parole, ogni parola è una componente e ha una misura che dipende dalle occorrenze della parola. Dopo è necessario calcolare la distanza: coseno dell'angolo sotteso tra i due vettori. Più i vettori sono allineati, più l'angolo tende a 0, più il coseno tende a 1, *più i vettori sono simili*.

Vediamo ora le tecniche per la ricerca di similarità utilizzando questo modello:

- Vector Space Model Booleano: mi rappresenta le parole con 1 se è presente, 0 se non è presente. Utilizziamo poi il dot product per trovare la similarità: $\text{Sim}(q,d) = q \cdot d = \sum_{i=1}^N x_i y_i$ dove le x sono in q , e le y sono in d . Non funziona quasi mai purtroppo, perché è troppo semplice non considera il TF.
- Improved VSM Instantiation: miglioriamo la tecnica precedente aggiungendo TF. Non consideriamo più l' i -esimo elemento di q e d come 1 o 0 in base alla presenza della parola ma lo sostituiamo con il numero di volte che essa compare.
- IDF Approach: non basta nemmeno aggiungere TF, allora mettiamo anche IDF che è l'inverso della DF. Utilizziamo la IDF perché la DF sarebbe il numero di elementi in cui una certa parola appare, ma vogliamo penalizzare le parole che appaiono in troppi documenti, quindi usiamo il suo inverso.

$$IDF(w) = \frac{M+1}{DF(w)}$$

E usiamo il suo logaritmo per smorzare alcuni valori troppo elevati. Ottenuto tale valore comunque, sfruttiamo TF come prima ma ci moltiplichiamo IDF che funge da peso.

$$f(q, d) = \sum_{i=1}^N x_i y_i = \sum c(w, q) c(w, d) \log\left(\frac{M+1}{DF(w)}\right)$$

- TF Transformation: l'approccio descritto nel punto precedente ha comunque un problema, la probabilità varia linearmente all'aumentare di TF; per esempio basta che la parola "cipolla" sia presente 2 volte che quel documento ottiene una probabilità doppia rispetto ad un documento dove appare una sola volta. Per compensare andiamo a smorzare TF ad esempio mettendo $1 + \log(1 + \text{TF})$.
- State-of-Art VSM Ranking Functions: pivoted lenght e BM25 si basano su altri parametri aggiuntivi che modificano il TF. BM25 sfrutta questa formula:

$$y = \frac{(k+1)TF}{TF+k}$$

Dove $k+1$ è il valore massimo che il TF può raggiungere. Se non è presente la parola, vale 0, altrimenti tenderà a $k+1$. Pivoted lenght si basa invece sulla lunghezza del documento.

5.4 Probabilistic Retrieval Models

Determinare, dato il documento e data la query, quanto esso sia rilevante. Si vede bene in questo esempio:

Query <i>q</i>	Document <i>d</i>	Relevant? <i>R</i>	
q1	d1	1	
q1	d2	1	
q1	d3	0	$f(q, d) = p(R = 1 d, q) = \frac{\text{count}(q, d, R = 1)}{\text{count}(q, d)}$
d1	d4	0	
q1	d5	1	$P(R = 1 q1, d1) = 1/2$
:			$P(R = 1 q1, d2) = 2/2$
q1	d1	0	$P(R = 1 q1, d3) = 0/2$
d1	d2	1	
q1	d3	0	
q2	d3	1	
q3	d1	1	
q4	d2	1	
q4	d3	0	

Confronta più risposte sulla domanda di rilevanza, come se confrontassi la risposta di più utenti. Approssimiamo la rilevanza alla probabilità:

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

Della query, data una rilevanza e un documento. Quindi la domanda diventa:

Qual è la query che mi dà come risultato tale documento?

Perchè è vantaggioso? Posso stimare le query ipotetiche, senza averne necessariamente una.

5.4.1 Text statistics: Zipf's Law

Il rank di una parola moltiplicato per la frequenza di tale parola è una costante.

$$r \times f = k$$

Quello che dice questa legge, nello specifico, è che ci saranno poche parole che appaiono tantissimo (le parole più frequenti) ma si decresce in maniera non lineare, quindi ci saranno tantissime parole che appaiono pochissimo.

5.4.2 Language Model

Il language model è una probabilità che io assegno ad una frase. Si suddivide in:

- machine translation: traduzione linguistica corretta
- correzione di pronuncia
- speech recognition: $p(\text{I saw a van}) >> p(\text{eyes awe of an})$

E tanti altri. Nel probabilistic language modeling l'obiettivo è quello di calcolare la probabilità di una frase o sequenza di parole. In questo modo, un'altra task correlata è sicuramente la prediction della upcoming word.

A livello matematico, l'obiettivo è quello di calcolare la probabilità di una parola w data una history $h = p(w|h)$; per farlo ci possiamo affidare alla *chain rule of probability*, cerchiamo di suddividere il problema, dove la probabilità di una frase diventa:

$$P(A, B, C, D) = P(A)P(B|A)P(C|A, B)P(D|A, B, C)$$

Ma non arriviamo ad un grande risultato perchè con frasi troppo lunghe sono troppo complesse. Dobbiamo usare dei bigrammi! Usiamo solo l'elemento precedente. Quindi in un documento invece di andare a visualizzare intere frasi, guardiamo solo la parola che precede. Il calcolo della probabilità, in quel caso, diventa:

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}w_i)}{\sum \text{count}(w_{i-1}w)}$$

Ovvero la probabilità della coppia che sto esaminando, diviso il numero di correlazioni tra la parola precedente e un'altra qualsiasi parola che non sia quella giusta. Il discorso è comunque analogo nel caso di trigrammi, o nel caso considerassimo k parole alla volta con $k > 3$.

Come è cambiata con il machine learning? Non è capace, un modello machine learning non riesce con troppi dati, ma una rete neurale sì.

Ovviamente vado poi a moltiplicare (come mostrato in alto) la probabilità di ogni bigramma fino ad ottenere la probabilità di una frase.

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

In questa immagine è possibile visualizzare un esempio di calcolo su bigrammi. Una domanda sorge spontanea, come vengono gestiti gli 0? Potrebbero dare problemi di generalizzazione! Allora per evitare che ci sia uno 0, possiamo utilizzare la tecnica di Laplace Smoothing, per cui aggiungiamo un 1 al conteggio di tutto.

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

I risultati della tabella cambiano di tanto! E non solo gli zeri.

Quello che avviene infatti nel Laplace Smoothing non è solo un aumento di 1 del conteggio, ma quando vado a calcolare le probabilità:

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}w_i) + 1}{\sum \text{count}(w_{i-1}w) + V}$$

Al denominatore vado anche a mettere il vocabolario V . Quindi il conteggio nelle tabelle, prima e dopo Laplace, cambia:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Ed è dovuto al fattore di discount d :

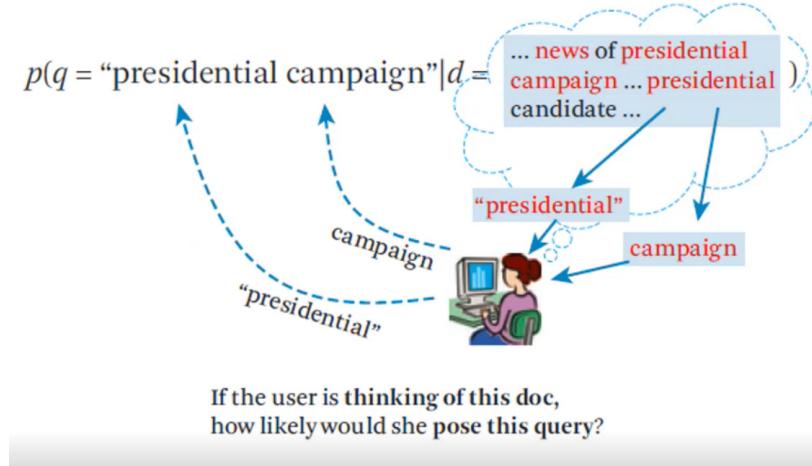
$$d = \frac{c^*}{c}$$

Dove:

$$c^* = (c_i + 1) \frac{N}{N+V}$$

5.5 Probabilistic Retrieval Models II

Per fare un search engine che si basa sulla probabilità, noi possiamo usare il concetto di language model. Possiamo andare a guardare in un documento quante volte appare una determinata coppia.



Esempio. Andiamo a guardare nel documento la probabilità che "presidential campaign" sia una domanda per questo specifico documento. Posso usare il language model, e guardare la probabilità delle parole singole all'interno del documento. Se moltiplico tra di loro le due probabilità ottengo la probabilità che la persona stia cercando queste keyword ma che loro non siano necessariamente correlate. **Assumiamo che la query sia generata campionando parole dal documento.** Ma nella pratica noi abbiamo delle parole correlate!

5.5.1 The Query Likelihood Retrieval Model

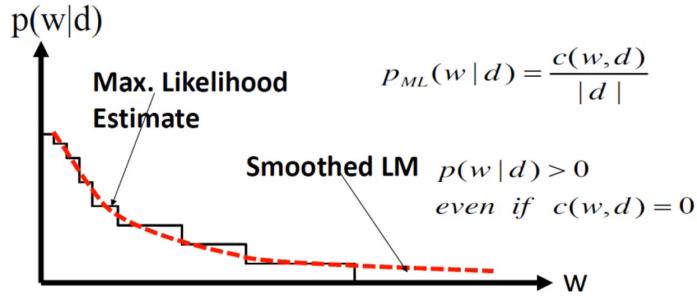
In un sistema probabilistico, la funzione $f(q,d)$ query-document è data dalla probabilità della query dato il documento $\log p(q|d)$ che posso suddividere sulle parole:

$$\sum \log p(w_i|d) = \sum c(w, q) \log p(w|d)$$

Dove il termine c è la frequenza della parola nella query. Quanto è importante quella query q per il documento d ? Lo andiamo a calcolare dalla rilevanza, vista per i vector space model. Si calcola come la frequenza della parola nella query (c) per la probabilità della parola nel documento. Questo modello ha una debolezza: spieghiamo bene con un esempio. La formula funziona bene per ciò che appare sia nel documento che nella query, oppure per quando non appare nella query (che fa 0). Tuttavia, se nella query la parola è presente, mentre nel documento è presente in parte, fa tutto 0, mentre noi vorremmo comunque far finire quel documento nel ranking.

Esempio. Parola: Francesco Guerra. Francesco non compare mai, nel documento attuale, ma Guerra sì! Ottiene comunque uno 0.

Dobbiamo quindi trovare una funzione di probabilità che non sia mai 0 quando il valore non è presente.



5.5.2 Smoothing della probabilità

Quale probabilità assegno a quelle parole che attualmente avrebbero 0? Potremmo mettere la probabilità della parola in una generica collection language model, moltiplicata per un α costante.

$$p(w | d) = \begin{cases} p_{\text{seen}}(w | d) & \text{if } w \text{ is seen in } d \\ \alpha_d p(w | C) & \text{otherwise} \end{cases}$$

Collection language model

Ora riscrivendo la funzione di ranking otteniamo:

$$\sum c(w, q) \log \frac{p_{\text{seen}}(w | d)}{\alpha_d p(w | C)} + |q| \log \alpha_d + \sum c(w, q) \log p(w | C)$$

Dove mettiamo in mostra tutte le possibili parole, seen e unseen. Quelle che vedi nella formula, sono queste parti, nello specifico:

$$\begin{aligned} \log p(q | d) &= \sum_{w \in V} c(w, q) \log p(w | d) \\ &= \sum_{w \in V, c(w,d) > 0} c(w, q) \log p_{\text{seen}}(w | d) + \sum_{w \in V, c(w,d) = 0} c(w, q) \log \alpha_d p(w | C) \\ &\quad \text{Query words matched in } d \quad \text{Query words not matched in } d \\ &\quad \sum_{w \in V} c(w, q) \log \alpha_d p(w | C) \quad \sum_{w \in V, c(w,d) > 0} c(w, q) \log \alpha_d p(w | C) \\ &\quad \text{All query words} \quad \text{Query words matched in } d \\ &= \sum_{w \in V, c(w,d) > 0} c(w, q) \log \frac{p_{\text{seen}}(w | d)}{\alpha_d p(w | C)} + |q| \log \alpha_d + \sum_{w \in V} c(w, q) \log p(w | C) \end{aligned}$$

Questa è una formula dove abbiamo tutti i componenti della vector space. Al numeratore del primo logaritmo abbiamo la Term Frequency e al denominatore la Document Frequency. L'ultimo termine dell'equazione è ignorabile per il ranking.

5.6 Search Engine Implementation

Un sistema IR è composto da 4 componenti:

- **Tokenizer:** genera l'input del sistema
- **Indexer:** indexing di molti documenti in uno spazio ridotto
- **Scorer/Ranker:** fare lo score dei documenti
- **Learner:** utile quando è presente un feedback

5.6.1 Tokenizer e Indexer

Il Tokenizer ha come obiettivo quello di estrarre delle features dal documento; l'Indexer invece mi trova la feature nei documenti sfruttando il concetto di inverted index: un indice che data una parola, mi trova i documenti che contengono quella parola. Normalmente gli inverted index si compongono di due file:

- lexicon, che contiene una lista di parole, quelle contenute nel documento
- posting, che contiene i dettagli delle parole

Come funziona un inverted index? Si va nei documenti e col tokenizer, si estraggono le parole o i token che riteniamo opportuni.



Term	Doc #
now	1
is	1
the	1
time	1
for	1
all	1
good	1
men	1
to	1
come	1
to	1
the	1
aid	1
of	1
their	1
country	1
it	2
was	2
a	2
dark	2
and	2
stormy	2
night	2
in	2
the	2
country	2
manor	2
the	2
time	2
was	2
past	2
midnight	2

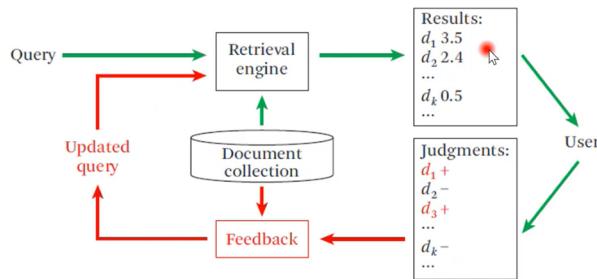
Term	Doc #
a	2
aid	1
all	1
and	2
come	1
country	1
country	2
dark	2
for	1
good	1
in	2
is	1
it	2
manor	2
men	1
midnight	2
night	2
now	1
of	1
past	2
stormy	2
the	1
the	1
the	2
the	2
their	1
time	1
time	2
to	1
to	1
was	2
was	2

Un file così è poco utile e confuso. L'indexer fa un orientamento alfabetico e questo è già un esempio di come lavora l'inverted index. L'inverted index può essere infatti diviso in un dictionary che fitta in memoria e contiene termini unici e un file posting che contiene le posizioni dei token, dove sono nel documento (a volte), e viene salvato in memoria.

5.6.2 Scorer e Learner

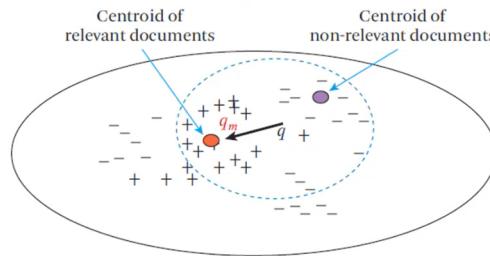
Data una keyword dobbiamo calcolare lo score, procedimento svolto dallo scorer sulla base delle informazioni ottenute da tokenizer e indexer, che salverà gli score, facendo poi il sorting dei documenti per rank.

Il learner usa una cronologia, delle informazioni precedenti per migliorare il risultato.



Il feedback che otteniamo è molto importante. Lo utilizziamo per modificare l'effettiva query dell'utente. Sappiamo che la query q , porta dei determinati documenti per cui noi abbiamo candidato delle query Q , allora modifichiamo la query dell'utente. Faccio un'interrogazione falsa sulla base delle sue richieste passate.

5.6.3 Rocchio Feedback



Avvicina la query ad un insieme di documenti rilevanti. Quello che dice il feedback di Rocchio è che:

$$q_m = \alpha \cdot q + \frac{\beta}{|D_r|} \cdot \sum d_j - \frac{\gamma}{|D_n|} \cdot \sum d_j$$

Dove gli elementi beta e gamma sono dei coefficienti per i documenti rilevanti e quelli useless. q è il vettore della query originale, D_r e D_n sono relevant e non-relevant documents, mentre i parametri aggiuntivi servono per calibrare il movimento del vettore originale.

I coefficienti sono molto delicati: ad esempio, se α fosse 0, ci si baserebbe molto solo su il passato di q , quindi overfitteremmo sulla query dell'utente. Nella pratica, andremo incontro ad un particolare problema: performeremmo un calcolo molto costoso, per aggiornare pesi e centroidi. I non-relevant, per di più non saranno per niente utili e dovremo fare attenzione alla questione overfitting (i parametri devono essere ricavati empiricamente).

5.7 Text classification

Consiste nel:

- Assegnare generi, categorie...
- Spam detection
- Identificazione

E molte altre cose. Quindi quello che si può fare è dato un documento d , e un set di classi Y , predire una classe y_i per d . Si può fare con delle regole, oppure esistono anche delle tecniche che si basano su Machine Learning.

Attenzione! Le tecniche di machine learning funzionano perfettamente, ma vanno in crisi: ad esempio, nella sentiment analysis. Immaginiamo la frase:

Marco è bello

Che ha accezione positiva. Ma basta aggiungere una parola:

Marco non è bello

Per cambiarne completamente il significato. E per un algoritmo di ML, capire questo è complesso.

5.7.1 Computational Lexical Semantics

Lavorare con la semantica delle parole, assegnargli un significato. Le macchine non sanno il significato delle parole. Ad esempio, trovare le relazioni tra parole, trovare le similitudini, anche trovare un modo per disambiguare una parola (ho 20 anni, venti sarebbe anche plurale di vento).

Cosa fai quando non sai il significato di una parola? Guardi sul vocabolario. Ma c'è un problema: sul vocabolario, ci sono i cosiddetti *lemmi*. Un lemma è ad esempio: io mangio, il suo lemma è mangiare sul vocabolario e come puoi vedere sono estremamente diversi. Quando si parla di parole correlate, potremmo dire ad esempio caffè e tazza. Oppure una relazione tassonomica, dal più grande al più piccolo, *villetta* è sottoinsieme di casa (esempio di tassonomia). In particolare si dice **iponimia** se ci si muove verso il basso e **ipernimia** se ci si muove verso l'alto. Ma il vocabolario questo tipo di informazioni non le contiene quindi anche usarlo è qualcosa di riduttivo.

Usare un vocabolario è una cosa complessa per una macchina. *Esempio*. Definizione di rosso:

Colore del sangue, o di un rubino.

La definizione di sangue invece è:

Il liquido rosso che circola nelle vene, arterie e cuore degli animali.

Le due definizioni di facile interpretazione per un umano, meno facile per una macchina che nota il collegamento ma non sa come utilizzarlo. WordNet è stato il primo dizionario lessicale, utile per applicazioni, fatto apposta per codificare un vocabolario utilizzabile da una macchina.

Per la disambiguation, esistono dei dataset appositi che possono essere utilizzati in ML con un approccio *supervised*. Per fare disambiguazione, esiste l'algoritmo di Lesk, che confronta un termine con il suo significato su WordNet e insieme al contesto riesce a disambiguare:

The **bank** can guarantee **deposits** will eventually cover future tuition costs because it invests in adjustable-rate **mortgage** **securities**.

bank ¹	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	"he cashed a check at the bank", "that bank holds the mortgage on my home"
bank ²	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	"they pulled the canoe up on the bank", "he sat on the bank of the river and watched the currents"

Come si può vedere dall'esempio nell'immagine, l'algoritmo di Lesk trova dei match nella definizione di baca come istituzione, e non come banco.

5.7.2 Vector semantics and embeddings

Possiamo pensare di avere parole simili, in **contesti** simili. Non ci serve necessariamente un vocabolario per capire delle parole ma basta il contesto.

*A bottle of tesguino.
Everybody like tesguino.
Tesguino makes you drunk.
We make tesguino out of corn.*

Da queste frasi riusciamo perfettamente a capire cosa sia il tesguino, semplicemente dal contesto! (O almeno, lo immaginiamo)

Iniziamo a vedere un approccio per cui rappresentiamo le parole come un vettore e questo vettore rappresenta il significato che attribuiamo a tale parola. Per costruirli sfruttiamo la **distributional hypothesis** che semplicemente sostiene che le parole vicine siano in un qualche modo correlate. Attraverso gli embeddings poi possiamo raggruppare e fare dei clusters con parole dal significato simile.

Possiamo costruire quattro tipologie di vettori:

- sparse vector representations
- SVD (singular value decomposition) and LSA (Latent Semantic Analysis)
- neural networks models
- contextualized embeddings

5.7.3 Sparse vector representations: Term-document matrix

Ogni riga rappresenta una parola nel vocabolario e ogni colonna rappresenta un documento di una collection.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	5	117	0	0

L'embedding lo costruisco su questa matrice: battle, sarà un vettore 1,1,8,15. Quello che facciamo quindi è creare una matrice di frequenza.

5.7.4 Sparse vector representations: word-word or term-context matrix

Sulle righe e sulle colonne abbiamo delle parole: il numero rappresenta se le parole compaiono insieme. Contiamo le 3 parole prima e le 3 dopo e vediamo il risultato del conteggio. Poi dipende quante parole vogliamo guardare, possiamo arrivare anche alle 10 parole ma ricerchiamo molto il significato.

	aardvark	...	computer	data	pinch	result	sugar	...
apricot	0	...	0	0	1	0	1	
pineapple	0	...	0	0	1	0	1	
digital	0	...	2	1	0	1	0	
information	0	...	1	6	0	4	0	

Però queste matrici sono molto grandi.

5.7.5 Positive Pointwise Mutual Information

$$PPMI(w, c) = \max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0)$$

Molto spesso, viene negativa quando il contesto non c'entra nulla con la parola. Si usa la positive per arrotondare a 0.

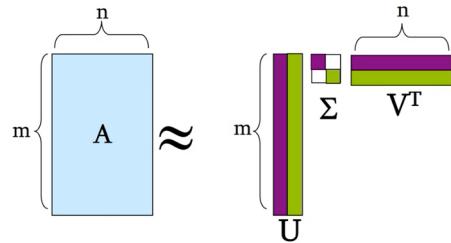
5.7.6 Singular Value Decomposition (SVD) e Latent Semantic Analysis (LSA)

Prende una distribuzione di valori, e ruota lungo gli assi per trovare una dimensionalità inferiore senza perdere informazione.

$$A_{[m \times n]} = U_{[m \times r]} \Sigma_{[r \times r]} (V_{[n \times r]})^T$$

Dove A è la matrice di input, U sono i left singular vector e V sono i right singular vectors, sigma sono i singular values. m sono i documents, n i terms, e r sono i concepts. Sigma è una matrice diagonale di valori ordinati in ordine decrescente.

$$\mathbf{A} \approx \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T = \sum_i \sigma_i \mathbf{u}_i \circ \mathbf{v}_i^T$$



Quello che fa è costruire una matrice densa, partendo da una sparsa. Elimino degli elementi da un certo punto in poi. Cosa succede? Che ho un'approssimazione della matrice originale, ma nel ML non è un male approssimare, generalizzo di più. Come decido però che approssimazione fare? Siamo nel caso di LSA (Latent Semantic Analysis) chiamata anche TruncatedSVD se manteniamo almeno un 80-90% dell'energia.

5.7.7 Word2vec

Attraverso questa tecnica creiamo dei vettori per ogni parola con le parole vicine ad essa in questo momento. Diventa un problema di classificazione binaria. Sistema rivoluzionario poiché self-supervised, coglie da solo le relazioni tra parole senza necessità di label.

Gli embeddings creati con questo metodo sono statici, impara un singolo embedding per ogni parola. Esistono tecniche più avanzate più recenti che guardano bene anche al contesto. La versione skip-gram:

1. Tratta la parola e quella vicina come esempi positivi
2. Prende delle parole casuali dal lexicon come esempi falsi
3. Usa la logistic regression per classificare esempi buoni e negativi
4. Usa i pesi della regressione come embeddings

Skip-gram usa due embeddings per ogni parola, uno per una parola come target e uno per quando diventa contesto per altre parole, e il risultato è la somma dei due.

Le parole positive sono facili da selezionare (sono quelle vicine a quella target) quelle negative vengono selezionate in base alla loro probabilità nel documento e quante sceglierne è un iperparametro. Normalmente si mette la weighted unigram frequency, è una probabilità pesata non quella pura,

gestita tramite un parametro anch'essa. Questo perchè per la probabilità di Zipf, ci sarebbero poche parole tanto frequenti e molte parole poco frequenti. Con il parametro andiamo a schiacciare quelle tanto frequenti, mentre gonfiando quelle poco frequenti.

Importante. Gli embeddings godono di una particolare regola: la distanza calcolata tra due parole di senso diverso, ma semanticamente collegate (ad esempio Re e Regina) e delle parole a loro correlate (ad esempio Uomo e Donna) è uguale. Quindi la distanza che c'è tra re e uomo è uguale a quella che c'è tra regina e donna. Questa proprietà prende il nome di *parallelogramma*.

5.8 Neural Networks Basics

Grazie a questi potenti algoritmi non abbiamo problemi degli altri algoritmi ML. Essi rientrano nella categoria DL (Deep Learning) e sono completamente blackbox. Essi si basano sul concetto di neurone, che è fatto come un neurone umano:

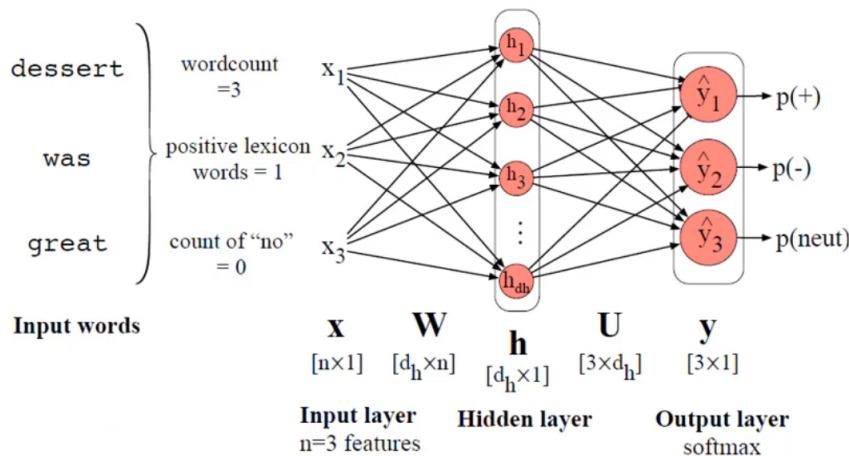
- ha un input, che è l'informazione dal neurone nel layer precedente o l'input di sistema
- ha una funzione di attivazione, che costruisce il valore di output
- ha un output che viene passato al layer successivo, o all'uscita dell'algoritmo

Esistono diverse funzioni di attivazione, la più importante sicuramente è la ReLU (rectified linear unit) che è una funzione = 0 per x negativo, poi restituisce x (quindi è una retta).

5.8.1 Feed Forward Neural Networks

Prendono questo nome perchè per scorrerli si va dal primo layer all'ultimo. Esistono altri modi per percorrerli. Si ha un input layer x , una serie di hidden layers h , e un output layer y . Negli hidden layer l'informazione viene elaborata e esce dagli output, dove per una interrogazione vengono fuori più risposte con un relativo score dato dalla funzione softmax che normalizza a 1 in modo da rendere gli score capibili anche da un umano.

Ma come la alleno su una frase? Potrei prendere gli embeddings, e farne la media, o prendere il valore max o min. Possiamo oppure unirli tutti: questa operazione prende il nome di pooling.



5.9 Recurrent Neural Networks

Sono particolari reti neurali che considerano la risposta precedente, e vengono utilizzate in NLP per l'analisi del testo. Tiene conto della posizione delle parole, la sequenza viene rispettata.

Ha degli svantaggi: necessita di tanti dati e perde efficacia con testi lunghi, non rilevando le relazioni tra parole distanti. Come possono due parole distanti avere una correlazione? Ad esempio, i pronomi riferiti al soggetto esplicito della frase.

5.9.1 Sequence classification

Metodo di funzionamento per la text classification con Recurrent NN. Abbiamo come input una frase e come risultato ottengo una prediction e un hidden layer che conosce tutte le parole. Possiamo quindi sfruttare quell'H per fare classificazione su altre frasi. Proprio in questo sta il suo vantaggio: poter allenare H e riutilizzarlo.

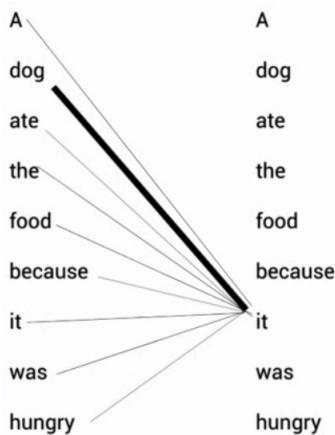
Ogni neurone ha una doppia funzione che è classificare e fornire contesto per la parola successiva. Difficile mantenere buone prestazioni. Esiste anche l'architettura LSTM (Long-Short Term Memory) che viene utilizzata in questo ambito, e aggiunge all'architettura base delle RNN un *forget*: decide gli elementi da salvare e quali possono essere dimenticati, attraverso una sigmoide finale che decide gli elementi da dimenticare.

5.10 Contextualized embeddings

Mentre nel word2vec viene fatto un embedding per ogni parola, in questo caso gli embeddings cambiano in base al contesto: state of art è il meccanismo di attenzione dei transformers.

5.10.1 Transformers

Architettura particolare che si basa sul meccanismo di attenzione: il contesto delle parole è variabile e dipendente dalla frase. Viene realizzato attraverso una rete encoder-decoder e possiamo ottenere un risultato di questo tipo:



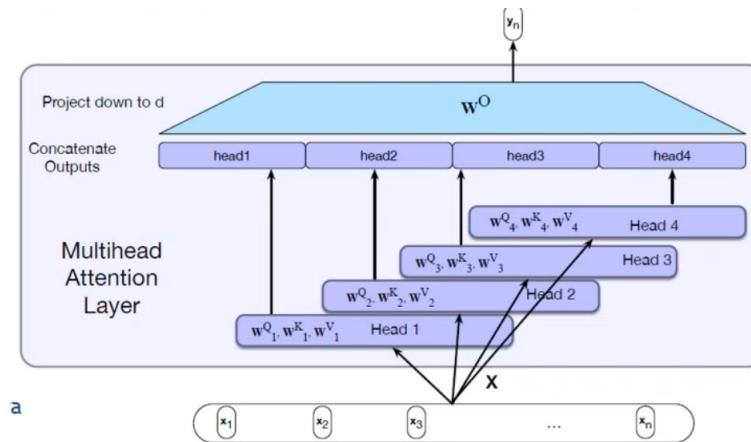
Ovvero la parola it guadagna il significato di dog, vengono correlate. Questo tipo di meccanismo serve principalmente per frasi molto lunghe. Sfrutta la conoscenza delle parole future, che non è scontato sapere.

Ma come funziona il meccanismo di self-attention? Il calcolo dei confronti nella matrice

ha come risultato uno score per ogni valore della query e ogni key, anche quelle che si seguono. Per evitarlo, vengono azzerate, in modo da eliminare ogni tipo di conoscenza sulle loro relazioni. L'attenzione è quadratica quindi va monitorata poiché troppo costosa. L'input viene usato in modo aggiuntivo: sommiamo i valori dell'input per non perdere informazione (*residual block*).

5.10.2 Multihead attention mechanism

Lo stesso input viene processato da n blocchi transformer, ogni head produce un risultato per lo stesso input, ogni blocco è chiamato head. In questa architettura viene sfruttato anche il concetto di positional encoding, per fornire informazioni utili sulla posizione dell'input (sapere la posizione di una parola all'interno della frase).



Explainable AI

6 Explainable AI

6.1 Interpretability

L'obiettivo è sempre quello di capire il perchè una macchina riesce ad imparare dai dati. Vogliamo quindi motivare perchè un algoritmo sceglie in un certo modo.

Ma perchè non lasciare decidere al modello e fregarsene? Uno dei principali motivi è la fairness. Un modello che decide uomo e donna, bianco o nero... il modello deve generalizzare ma soprattutto deve evitare razzismo, omofobia, misoginia. Se capiamo il motivo dietro alle sue scelte, possiamo influenzarle. Si distinguono:

- modelli intrinsic: modelli ultra interpretabili, come gli alberi
- modelli post hoc., modelli aggiuntivi, cioè io ho dei modelli non interpretabili che mi danno l'accuracy e dei modelli semplici che spiegano quelli non interpretabili

Come risultati dell'interpretazione otteniamo:

- feature summary statistic: una statistica per ogni feature
- model internals: per esempio dei weights
- data points: categoria che identifica tutte le tecniche che danno come risultato dei data points per l'interpretabilità
- intrinsically interpretable model: approssimare un modello black box con uno white box

E si possono distinguere anche tra spiegazione globale e locale, dove una ovviamente spiega a livello generale un modello mentre l'altra è sulla singola predizione. Poi ci sono diverse metriche per la valutazione della spiegazione, come l'accuracy e la fedeltà. La fedeltà è molto importante perchè rappresenta quanto la spiegazione è corretta. Ovvero quanto il modello utilizzato riesce a rappresentare il modello più complicato.

Algoritmi diversi, danno pesi diversi. Quindi un altro problema da porci è la consistency insieme alla stability. Ovviamente la spiegazione dev'essere ben diversificata sulla base dell'istanza, altrimenti non è comprensibile; e deve avere un certo grado di importanza per far capire l'importanza delle feature spiegate.

6.1.1 Interpretable Models

- **Linear regression:** molto semplice da capire perchè è un modello lineare la cui rappresentazione è una semplice retta. Tanto più il coefficiente è elevato, tanto più quel valore avrà importanza. La valutazione dell'importanza può essere fatta attraverso l'utilizzo delle Effect Plot, che si calcolano semplicemente come il peso per l'attributo.
- **Logistic regression:** L'interpretazione non è semplice come nella linear regression, soprattutto parlando dei pesi. Il cambiamento di una feature di una unità, cambia il valore esponenzialmente (\exp).
- **Decision tree:** probabilmente il modello white box per eccellenza, una decisione binaria multipla, per capirlo basta "seguire il percorso che fa l'albero". Come faccio a dare più o meno importanza nella decisione di un nodo dell'albero? Guardiamo il calo di entropia (guardiamo l'info gain).

Andiamo a vedere come funzionano i modelli agnostici globali.

6.1.2 Partial Dependence Plot

Mostra l'effetto marginale che hanno una o due features sulla prediction del modello ML. Sono dei diagrammi, che mostrano l'impatto sul risultato finale.

Si calcola con la feature di interesse e dei valori significativi per tale feature; fisso un valore significativo e cambio le altre feature: ad esempio, feature **Temperatura** valore significativo 10, fisso questo e cambio tutte le altre, poi faccio la prediction. Faccio la media dei risultati e costruisco i diagrammi.

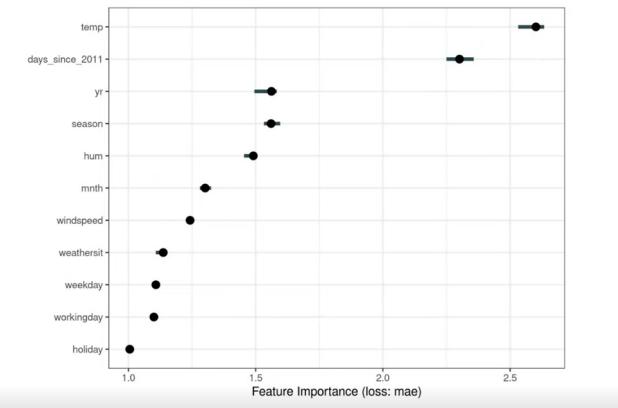
$$f_S(x_S) = \frac{1}{n} \sum_{i=1}^n f(x_S, x_C^{(i)})$$

Ed è una spiegazione globale.

6.1.3 Permutation Feature Importance

Valutiamo se una feature è importante semplicemente guardando quanto cambia il risultato se modifichiamo i valori che aveva in precedenza tale feature.

Torniamo sull'esempio precedente, **Temperatura**, e prendiamo tutti i suoi valori. Facciamo uno shuffle e li scambiamo. Come varia l'accuracy del modello? Se varia molto, allora la feature è molto importante altrimenti non lo è.



Questo è un possibile diagramma delle pfi di un dataset. Ed è sempre un discorso globale, perchè lo dico in funzione delle altre features.

Attenzione! Calcolare la pfi è un'operazione da fare nel test e non nel training, per evitare overfitting (che è molto presente durante il training).

6.1.4 Leave-One-Covariate-Out

Alleno senza una feature e guardo come varia il risultato. Introduco una nuova grandezza: LOCO Feature Importance, come il rapporto tra l'errore sul dataset iniziale e quello senza la feature. Non otteniamo un diagramma, potremmo fare come per il permutation feature importance.

6.1.5 Local Surrogate (LIME)

Modello agnostico locale. Si usa un po' ovunque. Costruisce un modello lineare che rappresenta un modello complesso, sulle singole prediction. Il modello surrogato non è uguale al modello complesso, ma nell'intorno considerato le prediction sono simili!

Costruiamo un nuovo dataset, creando una perturbazione del dataset originale. Individuiamo un valore di una feature, e lo sostituiamo con valori in un suo intorno. Ottengo una label diversa! Un altro modo potrebbe anche essere quello di eliminare delle features (ad esempio, per le feature categoriche). Più la perturbazione mi porta vicino, più è importante quindi sarà una perturbazione pesata. Uso un dataset perturbato con una funzione lineare e ottenere un risultato simile.

$$\text{explanation}(x) = \text{argmin}_L(f, g, \pi_x) + \Omega(g)$$

Questo è l'explanation model per x. Il valore in pi greco è la grandezza dell'intorno di x.

6.1.6 Counterfactual Explanation

Altre spiegazioni a livello locale. Si può spiegare banalmente in questo modo: se mia nonna avesse avuto le ruote sarebbe stata una carriola.

Il ragionamento dietro alle counterfactual è di facile interpretazione per l'utente, perché ipotizzo che se una delle feature fosse diversa, probabilmente sarebbe diverso anche il risultato della prediction. Uno dei principali contro sarebbe il 'Rashomon effect': ci sono più motivi solitamente per cui una prediction cambia, quindi lo stesso caso potrebbe avere tante spiegazioni diverse. Inoltre potrei cambiare feature ipotizzando cose impossibili.

Quindi riassumendo: modifico delle feature fino ad ottenere il risultato di prediction opposto.

6.1.7 Shapley Values

Immaginiamo che ogni valore di feature sia semplicemente un 'giocatore' e la prediction il risultato del gioco.

Esempio. Hai allenato un modello per predire il prezzo di un appartamento. Uno di essi, ha come prezzo previsto 300k. Questo appartamento ha certe features: area 50, secondo piano, parcheggio e gatti proibiti. In generale, il valore previsto per questi appartamenti sta sui 310k, quindi sei in perdita di 10k. Potremmo vedere le singole feature quanto tolgoano dal prezzo, e questo spiegherebbe il prezzo finale. Ad esempio, il problema potrebbe essere la questione gatto, che potrebbe abbassare il prezzo dell'appartamento.

Devo quindi confrontarmi con le altre 'coalizioni' ovvero uso delle feature diverse e vedo la prediction come cambia.

I 4 assiomi del valore di Shapley:

- **Efficienza:** tutti i ricavi della grande coalizione N sono da spartire tra tutte le features
- **Simmetria:** giocatore i e j che hanno lo stesso contributo, ottengono lo stesso ricavo
- **Linearità:** i valori di due 'giochi' distinti si possono sommare linearmente
- **Dummy player:** Quelli che non contribuiscono non ottengono nulla

La **Shapley value equation**:

$$\phi_i(N, v) = \frac{1}{N!} \sum |S|!(|N| - |S| - 1)! [v(S \cup \{i\}) - v(S)]$$

Dove la sommatoria su tutte le possibili coalizioni senza l'elemento che sto considerando. E l'ultimo termine sottraggo la coalizione con e senza l'elemento considerato.

$$\phi_i(N, v) = \frac{1}{N!} \sum_{S \subseteq N \setminus \{i\}} |S|! (|N| - |S| - 1)! [v(S \cup \{i\}) - v(S)]$$

Average
Weight
Marginal contributions

Esistono un paio di modi per approssimare la Shapley, dato che è molto complessa da calcolare, come la SHAP che approssima il calcolo.

6.1.8 KernelSHAP

Il valore nella SHAP fa una maschera (kernel) per quello prende il nome di kernelshap. Vediamo la formula nel dettaglio:

$$g(z') = \phi_0 + \sum_{j=1}^M \phi_j z'_j$$

Dove z è il coalition vector, il vettore delle coalizioni, che è 0 o 1 e fa quindi una maschera. Facendo una maschera di fatto sto anche creando un nuovo dataset!

Dopo aver applicato la maschera, peso ogni punto secondo questa formula:

$$\pi_x(z') = \frac{(M-1)}{\binom{M}{|z'|} |z'| (M-|z'|)}$$

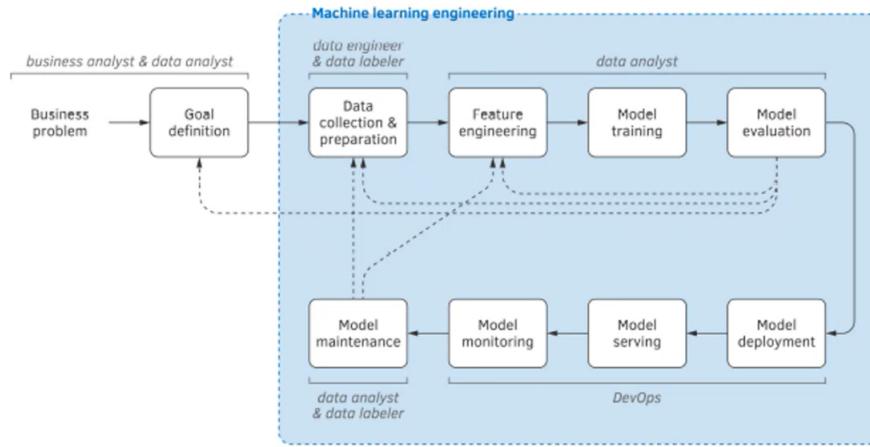
Che vuol dire che se ho solo valori mascherati quella riga vale molto, se ne ho un po' non mascherati vale meno. Ora fitta il modello con i pesi di Shapley.

ML Ops

7 MLOps

7.1 Project Life-Cycle

Parliamo della produzione di modelli. Vediamo la pipeline nello specifico dall'immagine seguente:



Model deployment, ovvero come viene realizzato (linguaggio); model serving, ovvero come il modello viene utilizzato; model monitoring, il modello va controllato periodicamente. Questo perchè potrebbe non avere le stesse performance che aveva in training e non in produzione. Ci sono diverse regole per cui si può scegliere di usare il ML o si sceglie di non usarlo, dipende molto dalla task, non va usato oltre la necessità.

7.2 Perchè i progetti di ML falliscono?

Le motivazioni principali sono:

- mancanza di talento con esperienza
- mancanza di supporto dalla leadership
- mancanza d'infrastruttura dati
- sfida del data labeling: label mancanti vanno aggiunte manualmente

In seguito a tutti questi problemi è nata una branca di ML chiamata MLOps che usa delle tecniche per vedere come possa funzionare una tecnica di ML quando viene messa in produzione.

E ancora:

- quantità di dati: i modelli sono molto sensibili ai dati quindi devono cambiare con loro
- decadimento dei modelli: col tempo degrada
- località: modelli sbilanciati sulla classe da predire

7.2.1 Mettere un modello in produzione: mlops

Ovviamente messa in produzione intendiamo rendere il servizio utile al pubblico. Di conseguenza, il modello dovrà avere delle determinate performances, per essere ritenuto all'altezza della produzione. Un modello potrebbe stare in produzione per anni senza alcun guadagno, l'operazione di Ops è necessaria.

Ci sono 5 aspetti chiave importanti da sapere nei MLOps:

- sviluppo
- deployment
- monitoring
- iterazione
- governance

7.2.2 Problemi con i dati

Molto spesso poi ci sono problemi con i dati:

- Costi elevati
- Scarsa qualità
- Rumore
- Bias

Bias è una polarizzazione, il dato pende verso una decisione. Ci sono diversi tipi di bias, ma il concetto è sempre questo: il bias influenza tanto la decisione del modello perché i dati sono fortemente influenzati. Oppure una classe sbilanciata, che tratto attraverso l'oversampling o l'undersampling: over aggiungo degli elementi in maniera randomica basandosi sugli altri dati (data augmentation). Undersampling fa la stessa cosa ma rimuove delle righe.

Tutte queste operazioni fanno parte della pipeline del data engineer: data ingestion, dove acquisiamo i dati, exploration e validation, dove si analizzano i dati raccolti, data wrangling, che comprendono tutte le operazioni di pulizia dei dati, data labeling, dare etichette, data splitting dove prepariamo il dataset per il modello.

7.2.3 Training and evaluation

Quando si crea il modello ci sono poi delle fasi da seguire:

- Training: tuning degli iperparametri durante la fase di training del modello
- Evaluation: allenato sui dati finali per poter essere utile all'utente
- Testing: test finale di accettazione
- Model packaging: esportazione del modello in un formato definito

7.3 Fairness in ML

La fairness è importante perchè i dati possono essere una rappresentazione distorta di ciò che è giusto. I modelli imparano le disparità involontariamente, ad esempio la disparità geografica (modelli razzisti). Questo prende il nome di bias preesistente. Esistono altri due tipi di bias, che sono quello tecnico e quello emergente.

Il bias tecnico viene fuori nella fase di prediction con dati non corretti o mancanti, che va ad amplificare spesso e volentieri il bias preesistente.

Esempio. Persone che mettono il sesso, altre non lo mettono. Il modello guarda la distribuzione che conosce e ipotizza i dati mancanti, ma non è detto che sia il risultato corretto. Un'altra cosa potrebbe essere lo stemming, le parole leaves (foglie) e lasciare (leave) abbreviate sono leav uguali. Anche il ranking introduce un errore, il primo sarà un risultato buono ma non sempre è nell'ordine corretto. Infine, l'emergent bias si manifesta come 'ricco che diventa più ricco', vado ad influenzare ad esempio nella raccomandazione di un film e modifco troppo la scelta.

7.3.1 No fairness through unawareness

Rimuovere o ignorare attributi sensibili che possono essere inefficaci e pure dannosi, perchè alcune correlazioni del dataset potrebbero risultare poi troppo deboli.

A volte le feature sensibili sono ridondanti se ci sono altre feature però se le rimuoviamo il classificatore troverà ridondanza altrove.

7.3.2 Pitfalls of action

Una volta creato il modello si generano le risposte. Si guardano le azioni e i feedback. Ovvero si guarda come agisce e si capisce il feedback del modello. Il feedback può essere interpretato per migliorare il modello come nei search engine.

Esistono 3 tipologie di feedback:

- **Self fulfilling prediction:** se il modello prevede correttamente, il retraining rafforzerà la predizione fatta
- **Predizioni che influenzano il training set:** il modello predittivo causerà degli arresti, che verranno aggiunti al training set e continueranno ad essere considerate come zone ad alto rischio
- **Predizioni che influenzano fenomeni o la società in larga scala:** il comportamento delle forze dell'ordine, colmo di pregiudizi, influenzera la povertà e i crimini delle zone a rischio

7.3.3 Statistical non-discrimination criteria

Questo criterio mira a definire l'assenza di discriminazione attraverso espressioni statistiche, coinvolgendo variabili aleatorie descrivendo lo scenario di classificazione.

Variabili random (A,R) devono soddisfare l'indipendenza se la predizione che facciamo è indipendente dall'attributo sensibile, quindi A è irrilevante, e tale indipendenza viene chiamata demographic parity.

Il demographic parity basta? Questo valore è scorrelato dall'accuracy sul dataset perchè è una misura del modello che prevede qualcosa, ma non della sua accuracy. Questa prende anche il nome di Independence, ovvero la prediction non è influenzata da feature inutili (es: uomo, donna).

7.3.4 Separation

Lo score di una predizione e l'attributo devono essere indipendenti dalla predizione. Fondamentalmente, tutti i modelli devono fare gli stessi errori in tutti i gruppi della variabile sensibile.

7.3.5 Sufficiency

Le variabili random (R,A,Y) soddisfano la sufficiency se $Y \perp A | R$ che è il simbolo della separation. Se il classificatore prevede 1, quando la probabilità è 0.5, questa deve rimanere 0.5 in tutti i gruppi sociali.

Metric	Definition	Fairness notion	Range	Interpretation
Disparate Impact (DI) [29]	$\frac{Pr(\hat{Y}=1 S=0)}{Pr(\hat{Y}=1 S=1)}$	demographic parity	$[0, \infty)$	$DI = 1 \rightarrow$ completely fair $DI = 0 \rightarrow$ completely unfair $DI = \infty \rightarrow$ completely unfair
True Positive Rate Balance ($TPRB$) [41]	$Pr(\hat{Y}=1 Y=1, S=1) - Pr(\hat{Y}=1 Y=1, S=0)$	equalized odds	$[-1, 1]$	$ TPRB = 0 \rightarrow$ completely fair $ TPRB = 1 \rightarrow$ completely unfair
True Negative Rate Balance ($TNRB$) [41]	$Pr(\hat{Y}=0 Y=0, S=1) - Pr(\hat{Y}=0 Y=0, S=0)$	equalized odds	$[-1, 1]$	$ TNRB = 0 \rightarrow$ completely fair $ TNRB = 1 \rightarrow$ completely unfair

7.4 Come rimuoviamo gli stereotipi?

Esistono 3 tecniche per rimuoverli: pre-processing, in-training, post-processing. Differiscono ovviamente per dove sono inseriti. Tra i due quello più sicuro è il preprocessing, ovvero che viene fatto prima della produzione del modello. Quello in-training è sul discorso dei pesi durante il training.