

SOMMARIO

| | |
|--|----|
| 1. CLASSIFICATION | 4 |
| TERMINOLOGIA (Spesso utilizzati come sinonimi) | 4 |
| 2. DECISION TREE CLASSIFIER | 5 |
| IMPURITA' DI UN SINGOLO NODO | 5 |
| IDENTIFICARE LA MIGLIORE TEST CONDITION | 6 |
| THRESHOLD | 7 |
| ALGORITMO RICORSIVO di Build dell' albero | 8 |
| ALGORITMO NON RICORSIVO di Build dell' albero | 9 |
| MODEL OVERFITTING | 10 |
| 3. MODEL SELECTION | 11 |
| MODEL EVALUATION | 12 |
| CROSS VALIDATION | 12 |
| HYPER PARAMETER | 13 |
| 4. SUPERVISED LEARNING | 14 |
| K-NEAREST NEIGHBORS | 14 |
| LOGISTIC REGRESSION | 16 |
| SUPPORT VECTOR MACHINE (SVM) | 17 |
| LINEAR SVM | 17 |
| SOFT MARGIN | 18 |
| NON LINEAR SVM | 19 |
| 5. ENSEMBLE METHODS | 20 |
| COSTRUZIONE ENSEMBLE CLASSIFIER | 20 |
| BIAS-VARIANCE DECOMPOSITION | 20 |
| BAGGING | 22 |
| BOOSTING | 23 |
| ALGORITMO DI BOOSTING : ADA BOOST | 23 |
| 6. RANDOM FOREST (BAGGING ALGORITHM) | 25 |
| OUT OF BAG SAMPLES | 26 |
| RANDOM FOREST COME SIMILARITY ESTIMATOR | 27 |
| OUTLIERS | 27 |
| RANDOM FOREST PER L'IMPUTAZIONE DEI MISSING VALUES | 27 |
| 7. FEATURE ENGINEERING | 28 |
| CATEGORICAL FEATURE | 28 |
| TEXT FEATURES | 28 |
| MISSING VALUES | 29 |
| 8. INVESTIGAZIONE APPROFONDITA DECISION TREE PERFORMANCE | 29 |
| CONFUSION MATRIX | 29 |
| RECALL | 31 |
| PRECISION | 31 |
| ACCURACY | 31 |
| F-MEASURE | 31 |
| AUC - ROC CURVE | 32 |

| | |
|---|-----------|
| 9. NAIVE BAYES CLASSIFIER | 34 |
| TIPOLOGIE DI NAIVE BAYES CLASSIFIER | 36 |
| MULTINOMIAL NAIVE BAYES | 36 |
| GAUSSIAN NAIVE BAYES | 36 |
| 10. ARTIFICIAL NEURAL NETWORK | 37 |
| STRUTTURA DI UNA ANN | 37 |
| HYPERPARAMETERS ANN | 37 |
| PERCEPTRON | 38 |
| MULTI-LAYERS | 38 |
| FUNZIONAMENTO | 39 |
| DISCESA DEL GRADIENTE | 39 |
| AGGIORNAMENTO DEI PESI CON DISCESA GRADIENTE | 40 |
| PROBLEMI | 40 |
| VANISHING GRADIENT | 40 |
| EXPLODING GRADIENT | 40 |
| ACTIVATION FUNCTION | 41 |
| 11. CLUSTERING | 42 |
| CRITERI DI ANALISI DEL CLUSTER | 42 |
| K-MEANS ALGORITHM - SINGLE LEVEL | 43 |
| K-MEDOID METHODS (PAM ALGORITHM) - SINGLE LEVEL | 45 |
| HIERARCHICAL CLUSTERING | 46 |
| LINKAGE CRITERIA | 47 |
| HIERARCHICAL-AGGLOMERATIVE CLUSTERING ALGORITHM | 48 |
| COMPLESSITA' | 49 |
| DENSITY BASED CLUSTERING | 50 |
| DBSCAN ALGORITHM - DENSITY | 50 |
| 12. CLUSTERING EVALUATION | 53 |
| INTRACLASS SIMILARITY | 54 |
| INTERCLASS SIMILARITY | 54 |
| SILHOUETTE COEFFICIENT | 54 |
| EXTRINSIC EVALUATION | 55 |
| 13. CLUSTER ANALYSIS | 56 |
| FUZZY C-MEANS | 56 |
| SELF ORGANIZING MAP (SOM) | 59 |
| 14. DIMENSIONALITY REDUCTION | 60 |
| PRINCIPAL COMPONENT ANALYSIS | 60 |
| 15. ASSOCIATION ANALYSIS | 63 |
| SUPPORT | 64 |
| CONFIDENCE | 64 |
| RULE MINING | 65 |
| GENERAZIONE DI ITEMSET da una lista di item | 65 |
| APRIORI PRINCIPLE | 65 |
| GENERAZIONE DI TUTTE LE REGOLE PER ITEMSETS FREQUENTI | 66 |
| SUPPORT COUNTING | 70 |
| ENUMERATING | 70 |

| | |
|--|-----------|
| HASH TREE | 71 |
| FP-GROWTH ALGORITHM | 72 |
| FP-TREE | 72 |
| FREQUENT ITEMSETS GENERATION | 73 |
| 16. RECOMMENDER SYSTEM | 76 |
| CONTENT BASED RECOMMENDER SYSTEM | 76 |
| INVERSE DOCUMENT FREQUENCY | 77 |
| PROFILE USER | 77 |
| COSINE SIMILARITY | 77 |
| USER BASED COLLABORATIVE FILTERING METHODS | 78 |
| USER SIMILARITY - PEARSON CORRELATION | 79 |
| ITEM BASED COLLABORATIVE FILTERING METHODS | 80 |
| ITEM SIMILARITY - ADJUSTED COSINE SIMILARITY | 80 |
| BONTÀ DELLA RACCOMANDAZIONE | 81 |
| 17. DOCUMENT SIMILARITY | 82 |
| K-SHINGLES | 82 |
| MIN HASHING | 83 |
| LHS: LOCALITY SENSITIVE HASHING FOR JACCARD DISTANCE | 85 |
| SIM-HASHING for COSINE SIMILARITY | 86 |

DATA AND WEB MINING

1. CLASSIFICATION

Le informazioni utili ad un task di classificazione consistono in collezioni di record. Ogni istanza è caratterizzata da una tupla (x,y) dove:

- x rappresentano i valori che descrivono l'istanza
- y è la classe dell'istanza - è sempre categoriale

Un modello di classificazione è una rappresentazione astratta della relazione tra il set di attributi e le etichette.

Possiamo rappresentarlo come un modello matematico $f(x) = y$ dove f prende in input il set di attributi x e produce in output y la classe dell'istanza predetta dati i relativi set di attributi.

Es.

| ATTRIBUTE SET | CLASS LABEL |
|--|-----------------|
| features estratte da un messaggio email (contenuto, header, mittente) | spam o non spam |

Un modello di classificazione ha principalmente 2 ruoli :

- usato come MODELLO PREDITTIVO → classificare istanze senza etichetta
- usato come MODELLO DESCRITTIVO → identificare caratteristiche che differenziano le istanze di classi diverse

TERMINOLOGIA (Spesso utilizzati come sinonimi)

- CLASSIFICATORE : tecnica di classificazione, core del processo di ml es. SVM, fa riferimento all'algoritmo.
- MODELLO : ciò che otteniamo come risultato dell'applicazione del classificatore su train set

N.B : ogni modello definisce un classificatore, ma non è vero che ogni classificatore viene definito da un singolo modello. ES. KNN non costruisce un modello esplicito, mentre gli ensemble classifiers combinano l'output di vari modelli

Le performance di un modello vengono valutate confrontando le etichette predette con le vere etichette delle istanza.

$$\text{ACCURACY} = \frac{\text{predizioni corrette}}{\text{predizioni totali}}$$

$$\text{ERROR RATE} = \frac{\text{predizioni errate}}{\text{predizioni totali}}$$

Gli algoritmi di learning della maggior parte delle tecniche di classificazione sono mirati ad avere la più alta accuracy o il più basso error ratio possibile

2. DECISION TREE CLASSIFIER

Nel seguente modello di classificazione l'approccio è quello che a seguito di una risposta ad una certa domanda, venga effettuata una nuova domanda fino a che non riusciamo a decidere la classe di appartenenza.

La serie di domande e le possibili risposte possono essere organizzate seguendo una struttura gerarchica chiamata DECISION TREE.

Un decision tree è organizzato come segue:

- root node → test condition
- internal nodes → test condition
- leaf nodes → contengono la classe

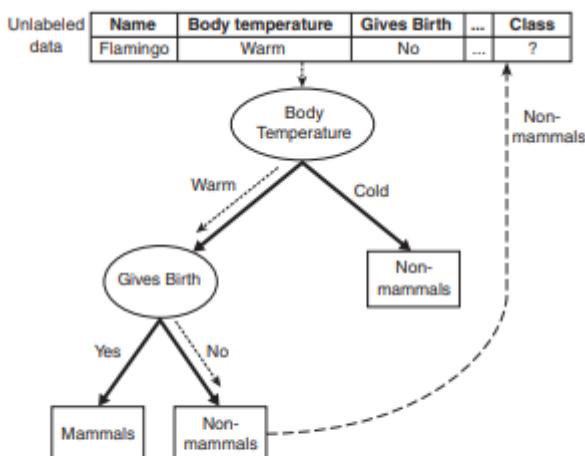


Figure 3.5. Classifying an unlabeled vertebrate. The dashed lines represent the outcomes of applying various attribute test conditions on the unlabeled vertebrate. The vertebrate is eventually assigned to the Non-mammals class.

Più alberi di decisione possono essere costruiti da un particolare dataset. Alcuni sono meglio di altri, trovare il migliore è computazionalmente molto costoso.

Esistono comunque algoritmi che forniscono una buona accuracy e tempo di costruzione ragionevole.

Utilizzano strategie greedy con un approccio top-down per la costruzione prendendo localmente una serie di decisioni ottimali su quale attributo usare durante il partizionamento

PROBLEMI:

- QUAL è IL CRITERIO DI SPLIT da usare ? (test condition)
- QUAL è IL CRITERIO DI STOP ?

IMPURITA' DI UN SINGOLO NODO

L'impurità di un nodo misura quanto ottimale è uno split in un decision tree, ci aiuta a determinare quale sia lo split ottimale preso in considerazione un non terminal node. Alcune misure utili a valutare l'impurità di un nodo sono:

- $Entropy = - \sum_{i=0}^{c-1} p_i(t) \log_2 p_i(t)$
- $Gini Index = 1 - \sum_{i=0}^{c-1} p_i(t)^2$
- $Classification Error = 1 - \max_i [p_i(t)]$

dove:

- $p_i(t)$ è la frequenza delle istanze del train che appartengono alla classe i nel nodo t
- c è il numero totale di classi
- $e 0 * \log_2 0 = 0$ nei calcoli dell'entropia

| Node N_1 | Count |
|------------|-------|
| Class=0 | 0 |
| Class=1 | 6 |

$$\begin{aligned} \text{Gini} &= 1 - (0/6)^2 - (6/6)^2 = 0 \\ \text{Entropy} &= -(0/6) \log_2(0/6) - (6/6) \log_2(6/6) = 0 \\ \text{Error} &= 1 - \max[0/6, 6/6] = 0 \end{aligned}$$

| Node N_2 | Count |
|------------|-------|
| Class=0 | 1 |
| Class=1 | 5 |

$$\begin{aligned} \text{Gini} &= 1 - (1/6)^2 - (5/6)^2 = 0.278 \\ \text{Entropy} &= -(1/6) \log_2(1/6) - (5/6) \log_2(5/6) = 0.650 \\ \text{Error} &= 1 - \max[1/6, 5/6] = 0.167 \end{aligned}$$

| Node N_3 | Count |
|------------|-------|
| Class=0 | 3 |
| Class=1 | 3 |

$$\begin{aligned} \text{Gini} &= 1 - (3/6)^2 - (3/6)^2 = 0.5 \\ \text{Entropy} &= -(3/6) \log_2(3/6) - (3/6) \log_2(3/6) = 1 \\ \text{Error} &= 1 - \max[3/6, 3/6] = 0.5 \end{aligned}$$

In generale, assumendo che l'errore sia una misura media, indichiamo il GAIN di uno split, sottraendo all'errore medio dell'albero, l'errore del sottoalbero dx + sottoalbero sx

$$Gain(f, t|D) = class. Error(D) - \left(\frac{|D_L|}{|D|} class. Error(D_L) + \frac{|D_R|}{|D|} class. Error(D_R) \right)$$

Quello che vogliamo è che il Gain sia > 0 , ricordiamo che ad ogni split, il gain nella peggior situazione rimane invariato e non può diminuire.

IDENTIFICARE LA MIGLIORE TEST CONDITION

Per determinare quanto è buona una test condition dobbiamo comparare il grado di impurità del nodo padre (prima dello split) con il grado di impurità pesato dei nodi figli (dopo lo split). Maggiore è la differenza e migliore è la test condition

La differenza Δ detta anche GAIN è definita come segue:

$$\Delta = I(parent) - I(children)$$

dove :

$$I(children) = \sum_{j=1}^k \frac{N(v_j)}{N} I(v_j)$$

che corrisponde alla media tra l'impurità dei nodi creati a seguito dello split e $I(parent)$ il grado di impurità del padre e $I(children)$ è l'impurità pesata dopo lo split.

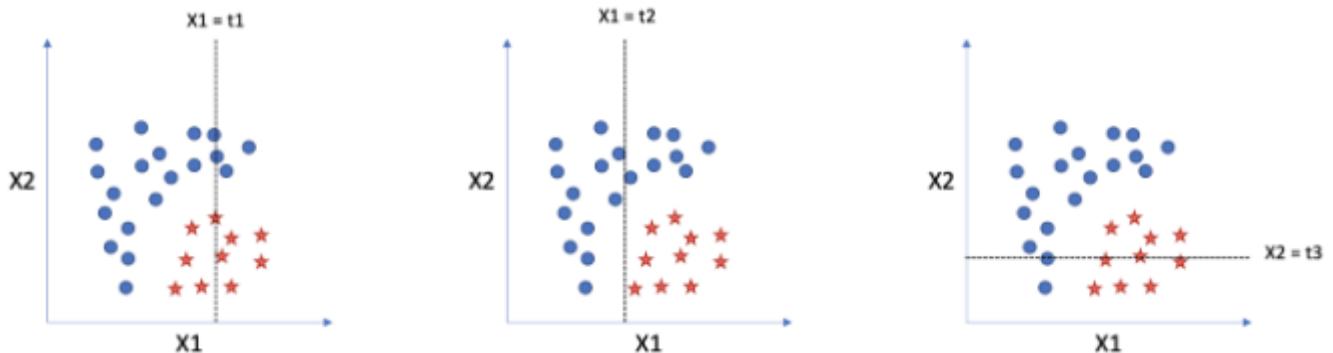
THRESHOLD

il threshold è la soglia che viene utilizzata per effettuare lo split, l'obiettivo è quello di scegliere un threshold che mi dia il miglior split possibile.

Il miglior split è quello che riesce a categorizzare il maggior numero di istanze di un tipo da una parte dello split.

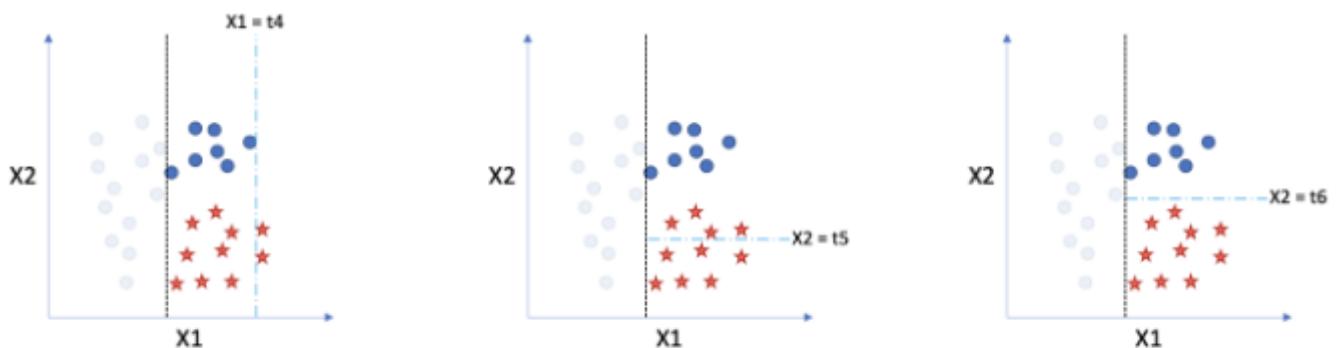
ES.

Passo 1



il miglior threshold è $X_1=t_2$ poiché con un solo split so che avrò sempre una pallina BLU nella parte sx, quindi scegliendo a caso un punto dove $X_1 < t_2$ avrò sempre una pallina blu.

Passo 2



3. Calculate splits for remaining data

Scegliendo lo split $X_2=t_6$ noto come con due split sono riuscito a ottenere dei nodi puri, ovvero dei nodi che contengono solamente istanze di una sola classe, questi nodi diventano i cosiddetti LEAF NODES.

[Esempio con calcoli](#)

ALGORITMO RICORSIVO di Build dell' albero

BuildTree(\mathcal{D}):

1. $BestSplit, BestGain = None$
2. **For each** feature f
3. **For each** threshold t
4. $Gain \leftarrow$ goodness of the split ($f \leq t$)
5. **If** $Gain >= BestGain$:
6. $BestGain \leftarrow Gain$
7. $BestSplit^* \leftarrow (f \leq t)$
8. **If** $BestGain = 0$ or other stopping criterion is met:
9. $\mu \leftarrow$ the best prediction for \mathcal{D}
10. **Return** $Leaf(\mu)$
11. Let f and t be those of $BestSplit = (f \leq t)$
12. $\mathcal{D}_L \leftarrow \{x \in \mathcal{D} \mid x_f \leq t\}$ (Left Partition)
13. $L \leftarrow \text{BuildTree}(\mathcal{D}_L)$ (Left Child)
14. $\mathcal{D}_R \leftarrow \{x \in \mathcal{D} \mid x_f > t\}$ (Right Partition)
15. $R \leftarrow \text{BuildTree}(\mathcal{D}_R)$ (Right Child)
16. **Return** $Node(L, R)$

It is a greedy algorithm (without backtracking, i.e., decisions are not changed) that maximizes the Gain at every step.

La potenza dei decision tree sta nel fatto di poter far mutare la costruzione dell'albero a seconda di molte caratteristiche pre-settabili come:

- profondità massima
- numero minimo di istanze per dover splittare un nodo
- numero minimo di istanze richieste ad un nodo per essere considerato foglia
- numero massimo di foglie
- il gain minimo per confermare uno split
- errore minimo per permettere uno split

SVANTAGGIO : possibilità di costruzione di un albero molto sbilanciato

ALGORITMO NON RICORSIVO di Build dell' albero

Ora ipotizziamo di impostare un vincolo sul massimo numero di foglie consentito.

L'algoritmo ricorsivo potrebbe diventare problematico nel momento in cui applicando la ricorsione, potrà andare a creare un albero totalmente sbilanciato dove troviamo, da una parte il `max_leaf_nodes` - 1 e dall'altra parte una sola foglia, ignorando di per sé metà dell'albero.

il problema viene risolto utilizzando un algoritmo non ricorsivo con una QUEUE

Non-recursive, best split first

BuildTree(\mathcal{D}):

1. $Tree \leftarrow \emptyset$
2. $(f \leq t) \leftarrow \text{best split of } \mathcal{D}$
3. $Gain \leftarrow \text{goodness of the split } (f \leq t)$
4. $Queue \leftarrow \langle gain, (f \leq t), \mathcal{D} \rangle$
5. **While** $Queue \neq \emptyset$ and no other stopping criterion is met:
6. $\langle gain, (f \leq t), \mathcal{D}^* \rangle \leftarrow Queue.pop_max()$
7. Add node $(f \leq t)$ to $Tree$ at the leaf corresponding to \mathcal{D}^*
8. (Left Partition)
9. $\mathcal{D}_L \leftarrow \{x \in \mathcal{D} \mid x_f \leq t\}$
10. $(f \leq t) \leftarrow \text{best split of } \mathcal{D}_L$
11. $Gain \leftarrow \text{goodness of the split } (f \leq t)$
12. $Queue.push(\langle gain, (f \leq t), \mathcal{D}_L \rangle)$
13. (Right Partition)
14. $\mathcal{D}_R \leftarrow \{x \in \mathcal{D} \mid x_f > t\}$
15. $(f \leq t) \leftarrow \text{best split of } \mathcal{D}_R$
16. $Gain \leftarrow \text{goodness of the split } (f \leq t)$
17. $Queue.push(\langle gain, (f \leq t), \mathcal{D}_R \rangle)$
18. **Return** $Tree$

MODEL OVERFITTING

Con i metodi di classificazione cerchiamo di ottenere l' errore più piccolo possibile nel nostro training set.

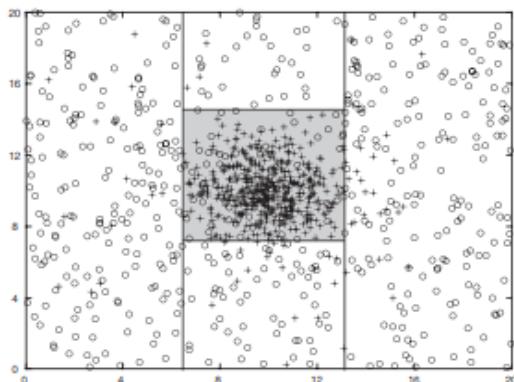
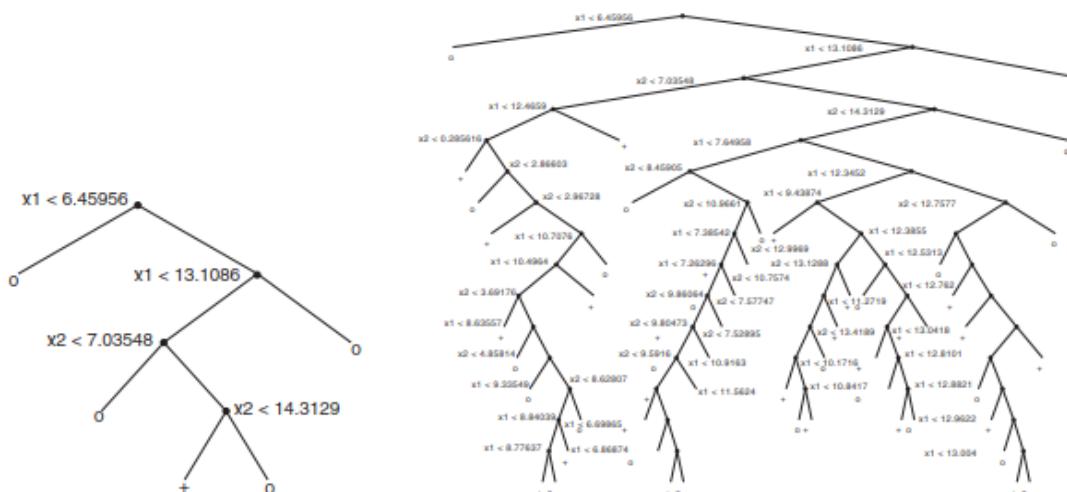
C'è comunque la possibilità che se il modello fitta perfettamente con il training set, abbia scarse performance di generalizzazione e quindi scarse performance nel testing set. Questo problema è detto **overfitting**.

Al contrario se il nostro modello è troppo semplicistico nella classificazione delle istanze si verificherà il cosiddetto **underfitting**.

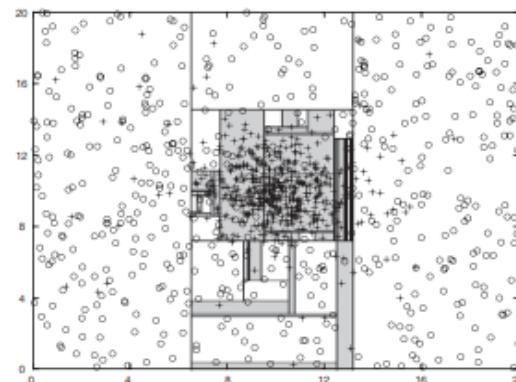
Nel caso di un decision tree quando il modello overfittà i dati offre una scarsa generalizzazione, mentre quando li underfitta offre una classificazione troppo semplicistica e non realistica della vera relazione tra attributi e etichette.

Quindi :

- **OVERTFITTING** : scarsa generalizzazione, modella il rumore e non la relazione, scarse performance sul testing set
- **UNDERFITTING** : classificazione semplicistica, non rappresenta la reale relazione tra attributi ed etichetta.



(c) Decision boundary for tree with 5 leaf nodes.



(d) Decision boundary for tree with 50 leaf nodes.

3. MODEL SELECTION

Ci sono molti possibili modelli di classificazione con vari livelli di complessità da poter usare per identificare i pattern all'interno di un dataset.

Tra tutte queste vogliamo il modello che ci dia il minor error_ratio di generalizzazione. Sappiamo inoltre che il training error ratio non è così affidabile se usato come unico criterio di selezione del modello.

STRATEGIE:

- **Occam's razor** : Consapevoli che l'overfitting aumenta all'aumentare della complessità del modello, la strategia di selezione non deve considerare solo l'error_rate del training set ma anche la complessità stessa del modello. A modelli con il medesimo errore preferiamo sempre il più semplice.

Introduciamo un elemento *alpha*: α di complessità all'errore del modello

$$\text{gen. error}(m) = \text{train. error}(m, D.\text{train}) + \alpha * \text{complexity}(M)$$

α è un iper parametro che bilancia la minimizzazione dell'errore nel train. set e la riduzione di complessità del modello stesso

Per scegliere α possiamo usare un sottoinsieme del training set detto VALIDATION SET, iterando in un range di valori per α e per ogni valore di α costruiamo un modello, ne computiamo l'errore e scegliamo l' α che minimizza l'errore su questo validation set.

In un Decision tree consideriamo la sua complessità come $\frac{K}{N_{\text{train}}}$ dove k è il numero delle foglie e N_train è il numero di istanze nel training set.

Quindi otteniamo che

$$\text{gen. Error}(T) = \text{Error}(T) + \alpha * \frac{K}{N_{\text{train}}}$$

dove Error(T) è il training error dell'albero di decisione

- **Pre pruning (Early Stopping Rule)** : con questo approccio la crescita dell'albero è fermata prima che venga generato un albero che fitti perfettamente con il training set. Per fare ciò è necessario introdurre una forte condizione di stop
Es. una foglia non viene spartita quando il gain nella stima del gen.Error va sotto una determinata soglia.

VANTAGGIO : evita computazioni associate alla generazione di sottoalberi complessi che andrebbero ad overfittare sui dati di train

SVANTAGGIO : se non si ottiene un guadagno significativo con uno dei criteri di suddivisione, ciò non toglie che un gain maggiore possa essere ottenuto allo splitting successivo, ma tali sottoalberi potrebbero non essere costruiti a causa della natura greedy(no backtracking) del pre-pruning

- **Post pruning** : in questo approccio l'albero viene fatto crescere fino alla sua grandezza massima. Poi viene fatto un pruning sull'albero andando a trimmare dei sottoalberi sostituendoli con una foglia della classe rappresentata maggiormente dalle istanze del sottoalbero trimmato(subtree replacement) oppure dal ramo del sottoalbero più utilizzato (subtree raising). Termina quando non possono essere fatti ulteriori miglioramenti al gen.Error osservato con un certo threshold.

VANTAGGIO: risultati migliori perché le decisioni vengono fatte dopo aver creato l'albero completo.

SVANTAGGIO: computazione aggiuntiva per far crescere dei sottoalberi che poi potranno essere trimmati.

MODEL EVALUATION

Finora la valutazione delle performance di generalizzazione del modello, abbiamo visto essere fatte su quello che è chiamato Training Set, ma queste stime dell'errore sono in indicatori distorti sulle istanze non ancora considerate dal modello, fino a che non vengono incluse per la selezione del modello di classificazione.

Il corretto approccio per la valutazione delle prestazioni di un modello sarebbe quello di valutarle su un test set non utilizzato in nessuna fase di selezione del modello.

Come fare:

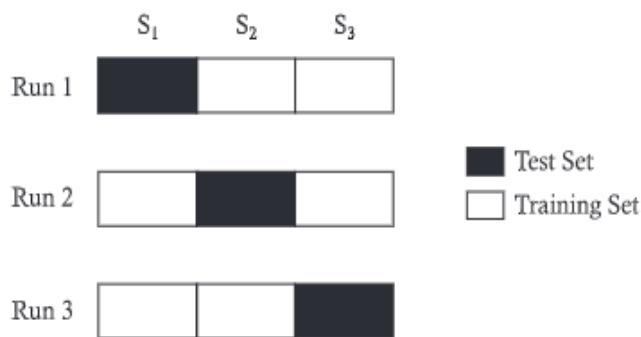
- suddividere il dataset in :
 - Training set → vedremo verrà suddiviso in validation set (per selezione iperparametri) e train, su cui computare l'errore
 - Test set → non deve essere coinvolto nella selezione e valutazione del modello (Pitfalls of model)

2 TECNICHE di PARTIZIONE:

- **HOLDOUT** : tecnica base, il dataset è partizionato randomicamente in train e test. Questa operazione può essere ripetuta più volte per ottenere una distribuzione dell'error_rate, detto random subSampling.
- **CROSS VALIDATION** : tecnica più utilizzata, ha l'obiettivo di riuscire a sfruttare in modo efficace le istanze del dataset sia nella parte di train che di test.

CROSS VALIDATION

Per spiegare questo metodo supponiamo di suddividere le istanze già etichettate, che abbiamo a disposizione, in 3 parti $S_1 S_2 S_3$ di dimensione uguale.



Run 1: alleniamo il modello usando S_2 e S_3 e poi lo testiamo su S_1 . L'errore del modello su $S_1 \rightarrow err(S_1)$.

Run 2: alleniamo il modello usando S_1 e S_3 e poi lo testiamo su S_2 . L'errore del modello su $S_2 \rightarrow err(S_2)$

Run 3: alleniamo il modello usando S_1 e S_2 e poi lo testiamo su S_3 . L'errore del modello su $S_3 \rightarrow err(S_3)$

Figure 3.33. Example demonstrating the technique of 3-fold cross-validation.

L'errore generale si ottiene sommando gli error_rate ottenuti e dividendoli per il numero totale di istanze. Questo approccio è detto 3 cross validation.

Possiamo generalizzare questo metodo in k-fold-cross-validation suddividendo il dataset in k parti uguali per ogni run e poi otteniamo l'error test facendo:

$$err_{test} = \frac{\sum_{i=1}^k err_{sum}(i)}{N}$$

Il corretto k per il nostro modello dipende dalla tipologia di problema.

- piccolo valore di k → allena su un piccolo set ad ogni run ottenendo un error_rate maggiore di quello che ci si aspetta testando sull'intero dataset
- grande valore di k → porta ad un training set sempre più ampio ad ogni run, che va a ridurre il bias dell'error_rate.

N.B

Il metodo di k-cross-validation non garantisce che la percentuale di istanze appartenenti ad un determinata classe vengano equamente distribuite nei sottoinsiemi di test e set. La soluzione sarebbe quella di distribuire le istanze in modo equo tra i subset con uno Stratified Sampling cross-validation.

HYPER PARAMETER

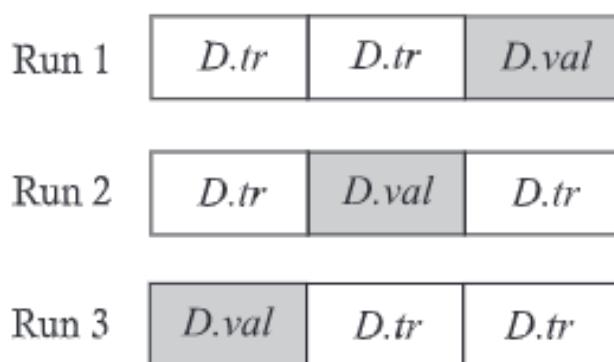
Gli hyper parameter sono parametri dei learning algorithms che devono essere decisi prima di imparare un modello.

Il processo per determinare questi parametri nella fase di model selection è detto Hyper-parameter selection.

Questa operazione viene effettuata attraverso quello che è detto VALIDATION SET ovvero il dataset di train viene suddiviso in un'effettiva parte di train e una di validation.

Questa suddivisione può essere applicata nel k-cross-validation dove ogni subset di train della procedura di k-cross viene a sua volta suddiviso in 2 parti di train e una di validation

Le 2 parti di train vengono utilizzate per allenare un modello per ogni valore di una ipotetica lista di valori di p* (hyper parameter).



Il modello viene poi applicato al validation set (Line 7) e calcolato il validation_error per ogni modello della i-esima run con un determinato parametro p* (Line 8).

Infine viene fatta la media di ogni validation_error con parametro p^* per ogni run i, ottenendo così il validation_error di p^* (Line 11).

Successivamente verrà preso il p^* che ha minimizzato il valore di validation_error (Line 12).

Algorithm 3.2 Procedure `model-select($k, \mathcal{P}, D.\text{train}$)`

```

1:  $N_{\text{train}} = |D.\text{train}|$  {Size of  $D.\text{train}$ .}
2: Divide  $D.\text{train}$  into  $k$  partitions,  $D.\text{train}_1$  to  $D.\text{train}_k$ .
3: for each run  $i = 1$  to  $k$  do
4:    $D.\text{val}(i) = D.\text{train}_i$ . {Partition used for validation.}
5:    $D.\text{tr}(i) = D.\text{train} \setminus D.\text{train}_i$ . {Partitions used for training.}
6:   for each parameter  $p \in \mathcal{P}$  do
7:      $m = \text{model-train}(p, D.\text{tr}(i))$ . {Train model}
8:      $\text{err}_{\text{sum}}(p, i) = \text{model-test}(m, D.\text{val}(i))$ . {Sum of validation errors.}
9:   end for
10:  end for
11:   $\text{err}_{\text{val}}(p) = \sum_i^k \text{err}_{\text{sum}}(p, i)/N_{\text{train}}$ . {Compute validation error rate.}
12:   $p^* = \operatorname{argmin}_p \text{err}_{\text{val}}(p)$ . {Select best hyper-parameter value.}
13:   $m^* = \text{model-train}(p^*, D.\text{train})$ . {Learn final model on  $D.\text{train}$ }
14:  return  $(p^*, m^*)$ .

```

4. SUPERVISED LEARNING

K-NEAREST NEIGHBORS

Gli alberi di decisione e i classificatori basati su regole sono detti EAGER LEARNERS poiché:

- quando ricevono un dataset iniziano a classificare le istanze (learning)
- non attendono il dataset di test per imparare
- hanno bisogno di molto tempo per imparare e meno tempo per classificare le istanze

Una strategia opposta sarebbe quella di ritardare il processo di modellazione dei dati di train fino a che non è necessario classificare le istanze del dataset di test.

Queste sono chiamate strategie di LAZY LEARNERS :

- non richiedono model building, computazionalmente costoso,
- non costruiscono un modello globale, sono suscettibili al rumore,
- misura di prossimità suscettibile a features non rilevanti e/o ridondanti/correlate.

L'algoritmo NEAREST NEIGHBORS può essere utilizzato per classificare le istanze del test set sfruttando l'approccio del ROTE CLASSIFIER, ovvero, performare la classificazione guardando a quale istanza del train set assomiglia di più a quella che deve essere classificata. Le decisioni vengono prese in locale sulla base delle istanze di train

"If it walks like a duck, quacks like a duck, and looks like a duck, then it's probably a duck."

L'algoritmo KNN rappresenta ogni istanza in uno spazio di d dimensione, dove d è il numero di attributi. Data un'istanza di test viene calcolata la vicinanza alle istanze presenti nel training set.

K in K-NN indica il numero k di istanze di train più vicine all'istanza z di test :

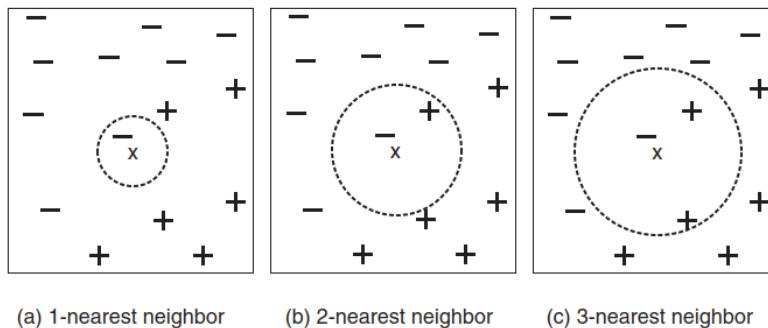


Figure 6.6. The 1-, 2-, and 3-nearest neighbors of an instance.

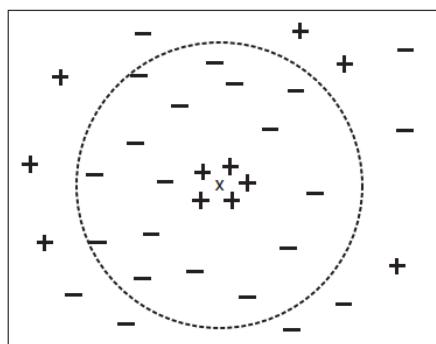


Figure 6.7. k -nearest neighbor classification with large k .

- Se k è troppo piccolo KNN diventa suscettibile ad overfitting perché troppo sensibile al rumore,
- Se k è troppo grande KNN potrebbe sbagliare la classificazione perché l'intorno potrebbe contenere istanze di train che sono effettivamente molto distanti dai suoi veri vicini. (fig 6.7) e inoltre computazionalmente costoso.

KNN da pesi uguali a tutte le features, talvolta potrebbe rivelarsi problematico nel momento in cui potrei avere distanze molto grandi su una sola feature e le altre tutte uguali, il risultato sarà una distanza totale molto ampia perché la distanza euclidea da la stessa importanza a tutte le features.

Soluzioni:

- cerco di rendere i dati confrontabili dandogli uno stesso range a tutte le features,
- utilizzo un approccio di tipo Distance-Weighted Voting (il peso è inversamente proporzionale alla distanza) invece che Majority Voting,

Algorithm 6.2 The k -nearest neighbor classifier.

- 1: Let k be the number of nearest neighbors and D be the set of training examples.
 - 2: **for** each test instance $z = (\mathbf{x}', y')$ **do**
 - 3: Compute $d(\mathbf{x}', \mathbf{x})$, the distance between z and every example, $(\mathbf{x}, y) \in D$.
 - 4: Select $D_z \subseteq D$, the set of k closest training examples to z .
 - 5: $y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i)$
 - 6: **end for**
-

LOGISTIC REGRESSION

La regressione logistica è un algoritmo di classificazione con apprendimento supervisionato utilizzato per prevedere la probabilità di una variabile target. La natura della variabile target o dipendente è dicotomica, il che significa che ci sono solo due classi possibili.

In parole poche, la variabile dipendente è binaria e i dati sono codificati come 1 (successo/sì) o 0 (fallimento/no).

Matematicamente, un modello di regressione logistica predice $P(Y=1)$ in funzione di X . Si tratta di uno dei più semplici algoritmi di ML che può essere utilizzato per vari problemi di classificazione, come il rilevamento di spam, la previsione di diabete, il rilevamento del cancro, ecc.

Assunzione per l'utilizzo della regressione logistica :

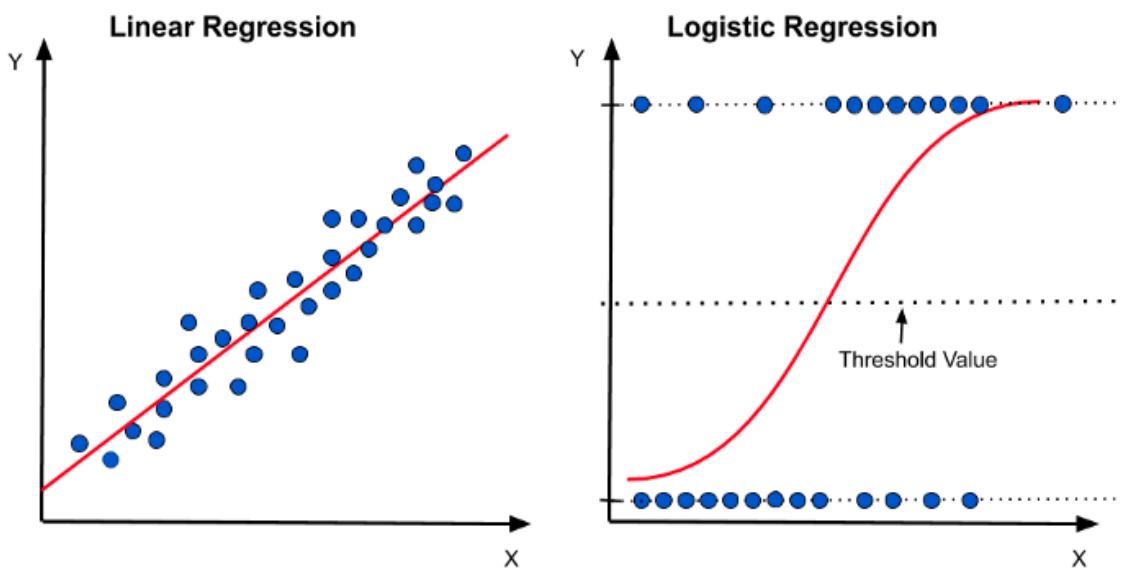
- la variabile target deve essere necessariamente sempre binaria
- non deve esserci multicollinearità tra le features, quindi le variabili devono essere indipendenti una dall'altra
- le features incluse devono essere significative
- è necessario un campione ampio per utilizzare la regressione logistica

L'obiettivo della regressione logistica è quello di riuscire a schiacciare l'output di una regressione lineare tra 0 e 1, in questo modo poter avere in output delle probabilità e risolvere un problema di classificazione.

$$Y = \text{Sigmoid } (b_0 + b_1 X_1 + b_2 X_2 + \dots + b_n X_n)$$

where, Sigmoid = $f(x) = \frac{1}{1 + e^{-x}}$

Therefore, $Y = \frac{1}{1 + e^{-(b_0 + b_1 X_1 + b_2 X_2 + \dots + b_n X_n)}}$



SUPPORT VECTOR MACHINE (SVM)

SVM è un modello di classificazione che apprende i confini di decisione lineari o non lineari nello spazio degli attributi per separare le classi.

- SVM possiede forti capacità di regolarizzazione, infatti è in grado di regolare la complessità del modello per offrire buone performance di generalizzazione,
- è in grado di imparare modelli molto espressivi senza soffrire di overfitting,
- rappresenta i confini di decisione utilizzando solamente nel set di train le istanze più complicate da classificare, chiamate SUPPORT VECTOR.

L'algoritmo è interessato a trovare un iperpiano che posizioni le istanze delle classi in posizioni opposte nell'iperpiano stesso.

- Chiamiamo **“margin”** la distanza tra le istanze più vicine di classi opposte lungo la direzione perpendicolare al confine decisionale selezionato.
- Più piccolo è il margine, maggiore è il rischio di **“misclassificazione”**.
- Le istanze che determinano il margine sono denominate **“vettori di supporto”**.

Se questo iperpiano esiste si dice che il dataset è LINEAR SEPARABLE.

LINEAR SVM

L'equazione generica di un iperpiano di separazione può essere scritta come:

$$w^T x + b = 0$$

dove:

- x rappresenta gli attributi
- (w, b) rappresenta i parametri dell'iperpiano
- L'istanza x_i appartiene ad una parte dell'iperpiano in base al segno di $w^T x + b = 0$

$$w^T x_i + b > 0 \quad \text{if } y_i = 1 \qquad \qquad w^T x_i + b > 0 \quad \text{if } y_i = -1$$

dove la distanza tra un qualsiasi punto dal iperpiano è data da

$$D(x) = \frac{|w^T x + b|}{\|w\|}$$

$|.|$ indica il valore assoluto e $\|.\|$ indica la lunghezza del vettore

Indichiamo la distanza del più vicino punto di classe $y=1$ dall'iperpiano come $k_+ > 0$ e
 $k_- < 0$ la distanza dall'iperpiano del più vicino punto di classe $y=-1$, possiamo
indicarlo così:

$$\frac{|w^T x_i + b|}{\|w\|} \geq k_+ \text{ se } y_i = 1 ; \frac{|w^T x_i + b|}{\|w\|} \leq k_- \text{ se } y_i = -1$$

possiamo riscrivere l'equazione come il prodotto di y_i e $(w^T x_i + b)$ come

$$y_i (w^T x_i + b) \geq M \|w\|$$

dove M è il parametro relativo al margine dell'iperpiano, cioè, se $k_+ = k_- = M$ allora il
margine $k_+ - k_- = 2M$.

Per trovare il massimo margine dell'iperpiano che soddisfa il precedente vincolo, bisogna considerare il seguente problema di ottimizzazione:

$$\max_{b,w} M \quad \text{applicato a} \quad y_i(w^T x_i + b) \geq M\|w\|$$

questo problema viene ridotto al seguente :

$$\min_{w,b} \frac{\|w\|^2}{2} \quad \text{applicato a} \quad y_i(w^T x_i + b)$$

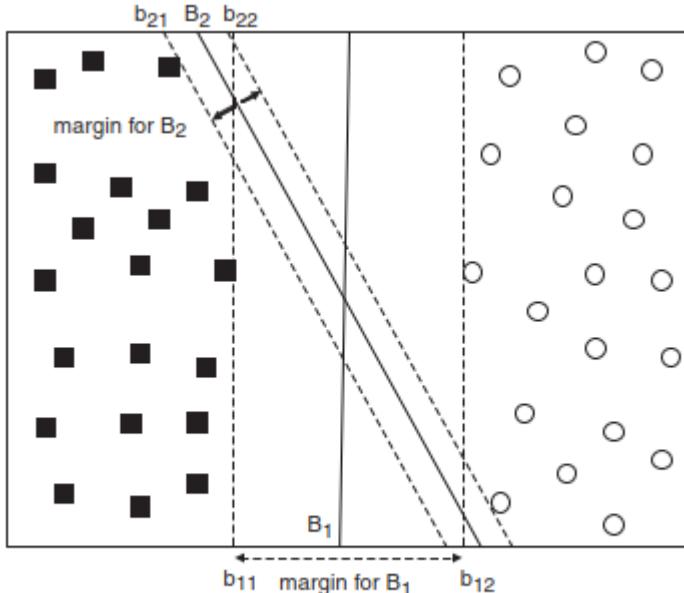


Figure 6.32. Margin of a hyperplane in a two-dimensional data set.

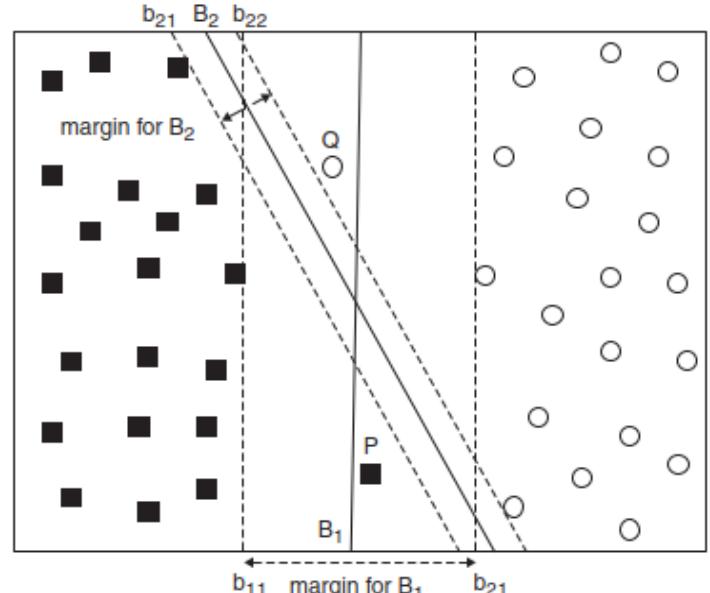


Figure 6.35. Decision boundary of SVM for the non-separable case.

SOFT MARGIN

Come vediamo nella figura sopra notiamo che il confine di decisione B1 classifica in maniera errata le nuove istanze Q e P, mentre il confine B2 le classifica in maniera corretta. Questo non significa che il confine B2 sia migliore di B1, perché Q e P potrebbero rappresentare rumore nel dataset di train.

B1 rimane preferibile a B2 poiché il margine è più ampio, e un margine più ampio ha minor rischio di misclassificazione.

Quando il dataset non è linearmente separabile è possibile sopperire a ciò tramite l'utilizzo dell'approccio noto come SOFT-MARGIN. Per fare ciò l'algoritmo di learning di SVM deve considerare il "trade-off" tra l'ampiezza del margine e il numero di training_error commessi dall'iperpiano.

Per introdurre il concetto di errore dobbiamo rilassare il vincolo di appartenenza per un piccolo gruppo di istanze.

Introduciamo quindi una SLACK VARIABLE $\varepsilon_i \geq 0$ per ogni istanza di train x_i come segue:

$$y_i(w^T x_i + b) \geq 1 - \varepsilon_i$$

ε_i sarà diverso da 0 nel momento in cui l'iperpiano non sarà in grado di posizionare l'istanza x_i nella parte di iperpiano delle istanze appartenenti alla classe y_i .

In presenza di questa variabile SLACK è importante avere un iperpiano che:

- massimizzi il margine e quindi offre una buona generalizzazione
- minimizzi il valore delle variabili SLACK assicurando quindi un basso training_error

Quindi :

$$\min_{w,b,\varepsilon_i} \frac{\|w\|^2}{2} + C \sum_{i=1}^n \varepsilon_i$$

applicato a

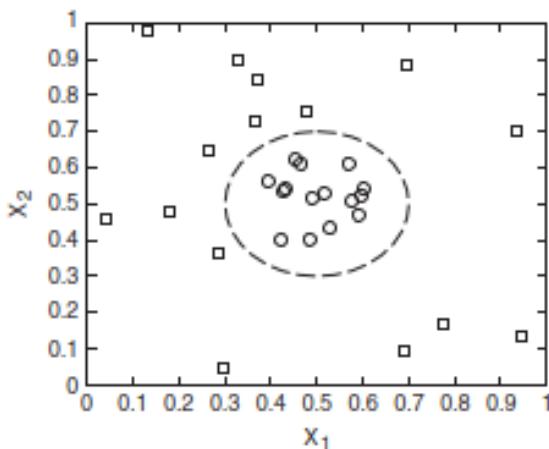
$$y_i(w^T x_i + b) \geq 1 - \varepsilon_i \text{ with } \varepsilon_i \geq 0$$

dove C è un iper parametro che rappresenta il trade-off tra massimizzazione del margine e minimizzazione del training_error.

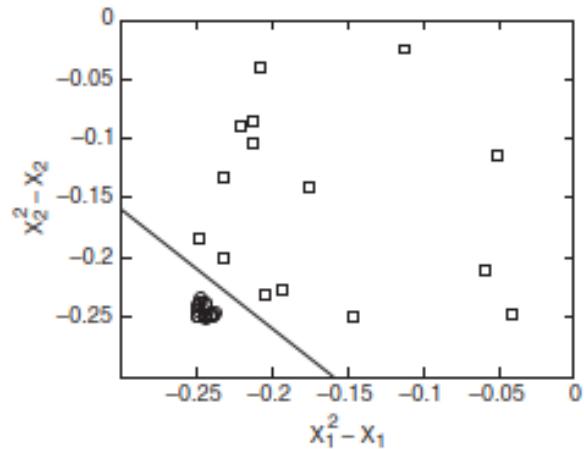
Un valore elevato di C pone più attenzione alla minimizzazione dell'errore di formazione che alla massimizzazione del margine.

NON LINEAR SVM

L'idea per riuscire ad applicare SVM a dataset che non sono linearmente separabili è quella di trasformare i dati dal loro spazio originale x in un nuovo spazio $\varphi(x)$ dove è possibili applicare SVM e trovare l'iperpiano. L'iperpiano generato è poi proiettato sullo spazio originale, avendo così un confine di decisione non lineare.



(a) Decision boundary in the original two-dimensional space.



(b) Decision boundary in the transformed space.

Figure 6.39. Classifying data with a nonlinear decision boundary.

5. ENSEMBLE METHODS

In questa sezione verranno presentate delle tecniche per migliorare l'accuratezza della classificazione unendo le predizioni di molteplici classificatori. Queste tecniche sono conosciute come ENSEMBLE METHODS o CLASSIFIER COMBINATION METHODS.
Un ensemble method crea un set di BASE CLASSIFIER dal dataset di train e esegue la classificazione guardando ai voti delle predizioni fatti da ogni BASE CLASSIFIER.

COSTRUZIONE ENSEMBLE CLASSIFIER

L'idea di base è quindi quella di creare molteplici classificatori dai dati originali, e aggregare le loro predizioni su dati sconosciuti.

Gli ensemble classifier possono essere costruiti in diversi modi:

1. **MANIPOLAZIONE DEL TRAINING SET**: l'idea è quella di ri-campionare i dati originali con distribuzioni differenti e costruire un classificatore per ogni training set:
 - BAGGING
 - BOOSTING
2. **MANIPOLAZIONE DELLE FEATURES DI INPUT**: in questo approccio viene preso un sottoinsieme delle features di input per formare ogni training set. Questo sottoinsieme può essere scelto randomicamente. Questo approccio funziona molto bene quando ci sono molte features ridondanti.
3. **MANIPOLAZIONE DELLE CLASSI**: questo metodo può risultare utile quando c'è un grande numero di classi. Il problema di classificazione viene trasformato in un problema di classificazione binario partizionando randomicamente le classi in due sottoinsiemi disgiunti A0 e A1.
Le istanze la cui classe appartiene al sottoinsieme A0 riceve classe 0, le rimanenti classe 1. Le istanze ri-elaborate vengono utilizzate per allenare un base classifier. Ripetendo questo processo si ottiene un insieme di base classifier.
Quando viene eseguito un test ogni base classifier C_i è utilizzato. Se un'istanza viene classificata come classe 0 tutte le classi che appartengono al sottoinsieme A0 ottengono un voto, stessa cosa per la classe 1 e A1. I voti vengono conteggiati e la classe che ha ricevuto maggior n di voti è assegnata all'istanza di test.
4. **MANIPOLAZIONE DELL'ALGORITMO DI LEARNING**: manipolazione dei parametri degli algoritmi per generare classificatori differenti

BIAS-VARIANCE DECOMPOSITION

La bias-variance decomposition è un metodo formale per analizzare l'errore di generalizzazione di un modello predittivo. Sebbene l'analisi sia leggermente differente per classificazione rispetto a regressione, discutiamo l'intuizione con problema analogo di regressione.

Consideriamo il dover sparare una serie di proiettili da una posizione iniziale x ad un target(che può leggermente muoversi) y . Il target corrisponde all'output che desideriamo data la nostra istanza di test, mentre la posizione iniziale x corrisponde agli attributi osservati. In questo esempio il proiettile rappresenta il modello utilizzato per predirre il target utilizzando gli attributi osservati.

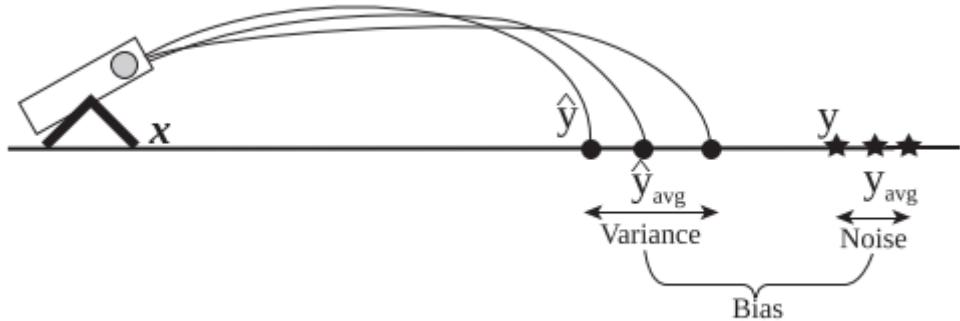


Figure 6.44. Bias-variance decomposition.

\hat{y} → punto in cui il proiettile colpisce il terreno, ovvero la predizione del modello

\widehat{y}_{avg} → predizione media di caduta del proiettile dopo vari tentativi

y_{avg} → media reale del valore del target

Idealmente vorremmo che la nostra predizione fosse il più vicino possibile al vero target. Notiamo come sono possibili differenti traiettorie del proiettile e queste si basano sulle differenze nell'approccio alle informazioni di train usate per scegliere il modello.

Quindi possiamo osservare una VARIANZA nella predizione di \hat{y} in diverse corse del proiettile. Inoltre il target nel nostro esempio non è fermo ma può muoversi, creando una componente di RUMORE(noise) nel vero target. Questo può essere identificato come la natura non-deterministica di un output, dove lo stesso set di attributi può avere output diversi.

La differenza tra \widehat{y}_{avg} e y_{avg} è chiamato BIAS del modello. Il bias rappresenta quanto la predizione del modello si è avvicinata al target medio

In un contesto di classificazione l'errore di generalizzazione di un modello m può essere suddiviso in fattori che riguardano la varianza il bias e il rumore del modello come segue

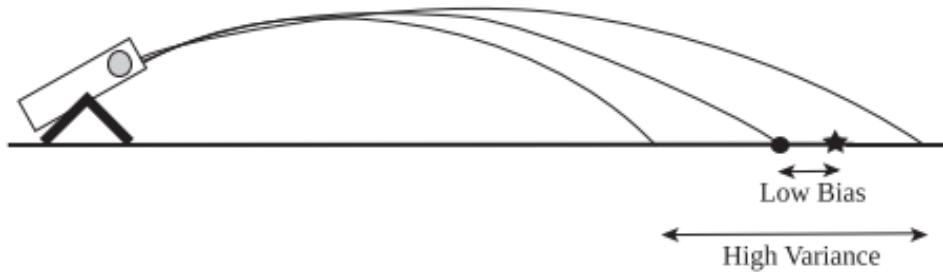
$$gen. error(m) = c_1 * noise + bias(m) + c_2 * variance(m)$$

Dove c1 e c2 sono costanti che dipendono dalle caratteristiche di train e set.

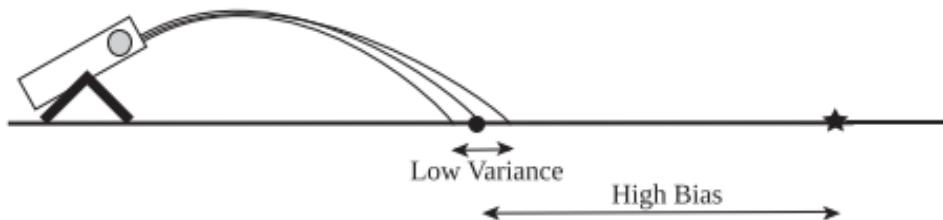
Possiamo dire che un modello mostra buone performance di generalizzazione hanno:

- BIAS basso
- VARIANZA bassa

Però se la complessità del modello è alta ma la dimensione del training set è bassa potremmo avere il problema dell'overfitting avendo basso bias ma varianza alta.



(a) Phenomena of Overfitting.



(b) Phenomena of Underfitting.

Se un base-classifier mostra basso bias ma alta varianza questo è suscettibile a overfitting, un piccolo cambiamento nel training porterà ad una predizione differente.

Però combinando le risposte di molteplici base classifier possiamo aspettarci di ridurre la varianza totale.

Quindi gli ensemble methods mostrano predizioni migliori riducendo in primis il BIAS.

BAGGING

La tecnica del bagging si basa sul multi re-sampling (con reinserimento) da un dataset con distribuzione di probabilità uniforme. Ogni sample ha la stessa dimensione dei dati originali.

Dato il reinserimento ad ogni estrazione alcune istanze potrebbero essere ripetute nel training set mentre altre potrebbero essere omesse. In media un sample D contiene circa il 63% dei dati originali. Se N è sufficientemente grande la probabilità converge a 63.2%.

PROCEDURA:

Dopo aver allenato k classificatori, l'istanza di test è assegnata alla classe che riceve il maggior numero di voti

Algorithm 6.5 Bagging algorithm.

- 1: Let k be the number of bootstrap samples.
 - 2: **for** $i = 1$ to k **do**
 - 3: Create a bootstrap sample of size N , D_i .
 - 4: Train a base classifier C_i on the bootstrap sample D_i .
 - 5: **end for**
 - 6: $C^*(x) = \operatorname{argmax}_y \sum_i \delta(C_i(x) = y).$
 $\{\delta(\cdot) = 1 \text{ if its argument is true and } 0 \text{ otherwise.}\}$
-

BOOSTING

Boosting è una procedura iterativa per cambiare la distribuzione delle istanze di train per allenare dei base classifiers focalizzati sul classificare istanze difficili da classificare. Diversamente dal bagging, boosting assegna un peso ad ogni istanza di train e può modificare il peso in maniera adattiva alla fine di ogni iterazione di boost.

I pesi possono essere utilizzati nei seguenti modi:

- Utilizzati per apprendere un modello orientato verso le istanze con pesi maggiori
- Utilizzati per informare la distribuzione di campionamento utilizzata per estrarre una serie di campioni di dati

IDEA:

- Ad ogni istanza viene assegnato inizialmente un peso identico $1/N \rightarrow$ tutte le istanze hanno le stesse probabilità di essere campionate (con reinserimento)
- Estrazione di un campione con distribuzione D
- Costruzione di un Classificatore dal training set e usato per classificare le istanze originali.
- I pesi delle istanze di train vengono aggiornati alla fine di ogni ciclo di boost. Il peso viene incrementato a tutte le istanze che sono state classificate in maniera errata e quello delle istanze classificate correttamente diminuito \rightarrow il classificatore si focalizza sulle istanze più complicate da classificare
- Le istanze il cui peso è maggiore hanno più probabilità di essere campionate nel sample del successivo ciclo di boost. All'aumentare del numero di cicli di boost la presenza delle istanze più complicate da classificare sarà prevalente nei campioni.
- Il classificatore finale viene ottenuto unendo tutti i classificatori generati ad ogni ciclo di boost.

ALGORITMO DI BOOSTING : ADA BOOST

Nell' algoritmo di adaboost, l'importanza di un classificatore C_i dipende la suo error rate:

$$\varepsilon_i = \frac{1}{N} \left[\sum_{j=1}^N w_j I(C_i(x_j) \neq y_j) \right]$$

Dove $I(p) = 1$ se p è vero, altrimenti = 0. L'importanza di un classificatore C_i è data dai seguenti parametri,

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

Notare che α_i ha un grande valore positivo, più l'error rate è vicino a 0, e un grande valore negativo se l'error rate è vicino a 1. Inoltre si tenga in considerazione che α_i è utilizzato per aggiornare i pesi delle istanze di test.

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \times \begin{cases} e^{-\alpha_j} & \text{if } C_j(\mathbf{x}_i) = y_i \\ e^{\alpha_j} & \text{if } C_j(\mathbf{x}_i) \neq y_i \end{cases},$$

Invece di usare uno schema di voto di maggioranza, la predizione viene fatta da ogni classificatore C_i pesato in base a α_i . L'approccio di AdaBoost penalizza modelli che hanno un'accuracy bassa, ovvero quelli generati nei primi cicli di boost. Se un ciclo di boost intermedio produce un error rate maggiore del 50% i pesi vengono ripristinati al valore originale $1/N$ e la procedura di ricampionamento viene ri-eseguita.

Algorithm 6.6 AdaBoost algorithm.

- 1: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \dots, N\}$. {Initialize the weights for all N examples.}
- 2: Let k be the number of boosting rounds.
- 3: **for** $i = 1$ to k **do**
- 4: Create training set D_i by sampling (with replacement) from D according to \mathbf{w} .
- 5: Train a base classifier C_i on D_i .
- 6: Apply C_i to all examples in the original training set, D .
- 7: $\epsilon_i = \frac{1}{N} [\sum_j w_j \delta(C_i(x_j) \neq y_j)]$ {Calculate the weighted error.}
- 8: **if** $\epsilon_i > 0.5$ **then**
- 9: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \dots, N\}$. {Reset the weights for all N examples.}
- 10: Go back to Step 4.
- 11: **end if**
- 12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
- 13: Update the weight of each example according to Equation 6.103.
- 14: **end for**
- 15: $C^*(\mathbf{x}) = \operatorname{argmax}_y \sum_{j=1}^T \alpha_j \delta(C_j(\mathbf{x}) = y)$.

ES.

Boosting Round 1:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| x | 0.1 | 0.4 | 0.5 | 0.6 | 0.6 | 0.7 | 0.7 | 0.7 | 0.8 | 1 |
| y | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 |

Boosting Round 2:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| x | 0.1 | 0.1 | 0.2 | 0.2 | 0.2 | 0.2 | 0.3 | 0.3 | 0.3 | 0.3 |
| y | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Boosting Round 3:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| x | 0.2 | 0.2 | 0.4 | 0.4 | 0.4 | 0.4 | 0.5 | 0.6 | 0.6 | 0.7 |
| y | 1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

(a) Training records chosen during boosting.

| Round | x=0.1 | x=0.2 | x=0.3 | x=0.4 | x=0.5 | x=0.6 | x=0.7 | x=0.8 | x=0.9 | x=1.0 |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 |
| 2 | 0.311 | 0.311 | 0.311 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| 3 | 0.029 | 0.029 | 0.029 | 0.228 | 0.228 | 0.228 | 0.228 | 0.009 | 0.009 | 0.009 |

(b) Weights of training records.

Figure 6.49. Example of boosting.

| Round | Split Point | Left Class | Right Class | α |
|-------|-------------|------------|-------------|----------|
| 1 | 0.75 | -1 | 1 | 1.738 |
| 2 | 0.05 | 1 | 1 | 2.7784 |
| 3 | 0.3 | 1 | -1 | 4.1195 |

(a)

| Round | x=0.1 | x=0.2 | x=0.3 | x=0.4 | x=0.5 | x=0.6 | x=0.7 | x=0.8 | x=0.9 | x=1.0 |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| Sum | 5.16 | 5.16 | 5.16 | -3.08 | -3.08 | -3.08 | -3.08 | 0.397 | 0.397 | 0.397 |
| Sign | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

(b)

Figure 6.50. Example of combining classifiers constructed using the AdaBoost approach.

6. RANDOM FOREST (BAGGING ALGORITHM)

L'algoritmo di random forest cerca di migliorare le performance di generalizzazione costruendo un insieme di alberi di decisione non correlati. Random forest è costruito sull'idea di bagging (usare un campione di train differente per allenare il modello). Tuttavia l'elemento chiave che distingue random forests dal bagging è che ad ogni nodo interno dell'albero, il miglior criterio di split è scelto randomicamente tra un set di attributi selezionati.

In questo modo le random forest costruiscono un insieme di alberi di decisione non solo manipolando le istanze di train ma usando anche un set di attributi che talvolta differiscono per ogni albero.

Dato un training set D di n istanze e d attributi, la procedura base di un random forest classifier può essere riassunto così:

1. Costruire un campione D_i dal training set campionando randomicamente con inserimento dal training set da D .
2. Usa D_i per allenare un decision tree T_i . Ad ogni nodo interno di T_i , viene scelto randomicamente un set p di attributi e scelto l'attributo da questo set che porta all'impurità più bassa su quello split. Questo procedimento viene ripetuto per ogni nodo fino ad avere delle foglie pure (con solo elementi della stessa classe).

Random Forest Algorithm

1. **for** $i = 1$ to k :
2. get a bootstrap sample D_i from dataset D
3. train a full tree M_i on D_i
4. at each split use only $F \ll d$ random features
 k
5. $RF = \bigcup_{i=1}^k M_i$
6. **return** RF

Per effettuare la predizione su un'istanza x :

- CLASSIFICAZIONE → voto di maggioranza tra i vari decision tree
- REGRESSIONE → $\frac{1}{K} \sum_{k=1}^K M_k(x)$, media delle predizioni dei decision tree

Una volta che l'insieme di decision tree è stato costruito, la predizione media (voto di maggioranza) sull'istanza di train è utilizzata come predizione finale della random forest.

Notiamo inoltre che tutti gli alberi della random forest sono alberi completi quindi alberi che possono essere molto grandi. Quindi il Random Forest classifier può essere

considerato unstable poiché ha basso bias ma varianza alta, a causa della loro dimensione.

Un'altra caratteristica è quella di una bassa correlazione tra i parametri dei modelli e le predizioni dei modelli, ciò è attribuito all'uso di un training set indipendente D_i per ogni albero T_i .

Tuttavia random forest hanno il vantaggio di scegliere il miglior attributo per lo splitting tra il set casuale di attributi a disposizione, questo aiuta a rompere una struttura di correlazione se esiste tra gli alberi T_i . Poiché considerando un insieme di attributi che sono ottimi predittori, i decision tree userebbero i medesimi attribuiti poiché saranno quelli che avranno il miglior criterio di split. E questo causerebbe alta correlazione tra gli alberi.

Utilizzando un subset limitato randomico di attributi per ogni nodo interno, garantisce una selezione per lo split sia di forti predittori che deboli predittori, creando diversità negli alberi.

Unendo le predizioni dei modelli costruiti su un insieme di alberi non correlati, random forest è in grado di ridurre la varianza senza impattare sul basso bias, rendendo la random forest robusta all'overfitting.

Il numero di attributi selezionati per ogni nodo è scelto da p , un hyper parametro del classificatore random forest. Un basso valore p porta ad una riduzione della correlazione ma riducendo la forza, un valore alto migliora la forza ma porta ad una maggior correlazione simili al bagging.

Una convenzione per task:

- di classificazione → è di scegliere un p per ogni nodo pari a \sqrt{d} dove t è il numero totale di attributi, con size minima del nodo = 1
- di regressione → è di scegliere un valore di p pari a $p/3$, con size minima del nodo = 5

OUT OF BAG SAMPLES

Un importante feature delle random forest è quella di utilizzare out-of-bag(OOB) samples:

For each observation $z_i = (x_i, y_i)$, construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which z_i did not appear.

La stima dell'error-rate fatta da OOB è circa identica a quella ottenuta utilizzando N-fold-cross-validation, diversamente da altri stimatori non lineari, possono essere allenate in un'unica sequenza con cross-validation lungo il processo. Una volta che OOB errore si è stabilizzato è possibile terminare il train.

RANDOM FOREST COME SIMILARITY ESTIMATOR

In definitiva, un modello raggruppa istanze simili e fornisce per esse la stessa previsione. Un decision tree utilizza i predicati dei nodi per identificare un sottoinsieme di istanze per le quali viene fornita la stessa previsione. In questo modo, questo raggruppamento di similarità si basa sull'etichetta di destinazione piuttosto che sulle caratteristiche. In un certo senso, questo supera il problema di pesare correttamente le caratteristiche per calcolare la somiglianza: quali caratteristiche sono rilevanti e come determinare se due istanze sono simili è implicitamente determinato dall'algoritmo di addestramento.

Come sfruttare una Random Forest per calcolare la somiglianza tra istanze diverse?

Possiamo costruire una misura di similarità sulla base della seguente ipotesi:

- due istanze sono simili se attraversano la foresta casuale lungo percorsi simili.

Ciò significa che due istanze sono simili se cadono nelle stesse foglie. Misuriamo quindi la somiglianza tra due istanze come il rapporto tra il numero di foglie in comune raggiunte durante l'attraversamento della foresta.

In linea di principio, questo metodo può essere applicato a qualsiasi foresta di decision tree, compresi bagging e boosting. Random forest è preferibile per i seguenti motivi:

- RF è migliore di Boosting in quanto RF dà lo stesso peso agli alberi
- RF è migliore di Bagging perché i suoi alberi sono più diversificati.

OUTLIERS

Seguendo una linea di ragionamento simile, possiamo utilizzare una Foresta casuale per identificare gli outlier nel nostro set di dati prima/senza eseguire un algoritmo di clustering.

Possiamo definire informalmente come outlier le istanze che sono dissimili dagli altri punti del dataset. Definiamo quindi il punteggio di outlier di un'istanza come l'inverso della somiglianza quadratica cumulativa con tutti gli altri punti del dataset. Più formalmente, per un'istanza $o_i \in D$ abbiamo:

$$out(o_i) = \frac{1}{\sum_{o_j} RF \sim(o_i, o_j)^2}$$

Dove RF sim è la similarità della random forest (non la distanza!)

RANDOM FOREST PER L'IMPUTAZIONE DEI MISSING VALUES

Il punteggio di somiglianza di Random Forest può essere sfruttato anche per l'imputazione dei valori mancanti, ossia per sostituire un valore mancante con un'ipotesi ragionevole. La logica è la seguente: riempire il valore mancante copiando quello di istanze simili.

Consideriamo un' istanza $O_I \in D$ in cui manca la caratteristica f . L'imputazione avviene attraverso la seguente procedura iterativa.

1. Sostituire il valore mancante con il valore medio di tutte le altre istanze j in cui f non è mancante, ponderando con il punteggio di similarità RF di (o_i, o_j) .

$$o'_i[f] = \frac{\sum_{\substack{o_j \in D \text{ with } o_j[f] \neq NaN}} o_j[f] * RF sim(o_i, o_j)}{\sum_{\substack{o_j \in D \text{ with } o_j[f] \neq NaN}} RF sim(o_i, o_j)}$$

2. Addestrare una nuova Foresta Casuale sui dati modificati e calcolare nuovi punteggi di somiglianza.
 - La nuova RF potrebbe essere completamente diversa
3. Ripetere fino a quando non ci saranno ulteriori miglioramenti.

7. FEATURE ENGINEERING

CATEGORICAL FEATURE

Le variabili categoriali possono essere utilizzate al meglio solo dopo essere state rimappate. Possono essere trasformate con una mappatura numerica ad esempio:

`{ 'Queen Anne': 1, 'Fremont': 2, 'Wallingford': 3 }`

Questo però potrebbe non avere molto senso nel momento in cui questi valori vengono interpretati come una quantità algebrica.

In questo caso la tecnica più nota è quella del ONE HOT ENCODING dove viene creata una colonna extra per indicare la presenza o assenza di una categoria con un valore 0 o 1. Quando un'istanza appartiene ad una ad una categoria la colonna extra corrispondente viene contrassegnata con un 1 mentre le altre con 0.

Lo svantaggio di questa tecnica è che molte categorie o molte variabili categoriali fanno esplodere il numero di colonne del dataset.

TEXT FEATURES

Un' altra tipologia di variabile è quella Text, una delle tecniche più comuni consiste nel codificare testo in una variabile numerica:

- Per ogni parola : creare una colonna e contare le occorrenze di ogni e inserire la frequenza nella colonna

```
Out[8]:   evil  horizon  of  problem  queen
      0      1        0     1       1      0
      1      1        0     0       0      1
      2      0        1     0       1      0
```

I problemi di questa tecnica sono:

- Troppa importanza a parole che appaiono molto frequentemente, ciò potrebbe essere subottimale nei contesti di classificazione.

Una tecnica per fixare ciò è conosciuta con TERM FREQUENCY-INVERSE DOCUMENT FREQUENCY(TF-IDF), in cui i pesi di ogni singola parola determinano un valore di quanto spesso una parola appare nei documenti.

```
Out[9]:      evil    horizon      of    problem    queen
      0  0.517856  0.000000  0.680919  0.517856  0.000000
      1  0.605349  0.000000  0.000000  0.000000  0.795961
      2  0.000000  0.795961  0.000000  0.605349  0.000000
```

MISSING VALUES

Le strategie per porre rimedio al problema dei missing values vanno dalle più semplici alle più sofisticate.

Le metodologie più utilizzate sono:

- Sostituire con la moda (variabili categoriali)
- Sostituire con la media
- Sostituire con la mediana
- Utilizzare una variabile binaria per indicare se è o meno un missing value

8. INVESTIGAZIONE APPROFONDITA DECISION TREE PERFORMANCE

CONFUSION MATRIX

Quando otteniamo i dati, dopo averli puliti, pre-elaborati e manipolati, il primo passo che facciamo è fornirli a un modello complesso e, naturalmente, ottenere un risultato in termini di probabilità. Ma aspettate! Come possiamo misurare l'efficacia del nostro modello? Migliore è l'efficacia, migliore è la performance, e questo è esattamente ciò che vogliamo. È qui che entra in gioco la matrice di confusione.

La matrice di confusione è una misura delle prestazioni per la classificazione di modelli di machine learning.

E' una tabella con 4 combinazioni differenti per valori predetti e valori attesi

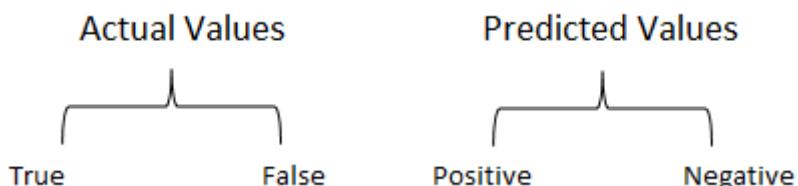
| | | Actual Values | |
|------------------|--------------|---------------|--------------|
| | | Positive (1) | Negative (0) |
| Predicted Values | Positive (1) | TP | FP |
| | Negative (0) | FN | TN |

E' estremamente utile per misurare Recall, Precision, Specificity, Accuracy, e soprattutto AUC-ROC curves.

Classifichiamo la terminologia:

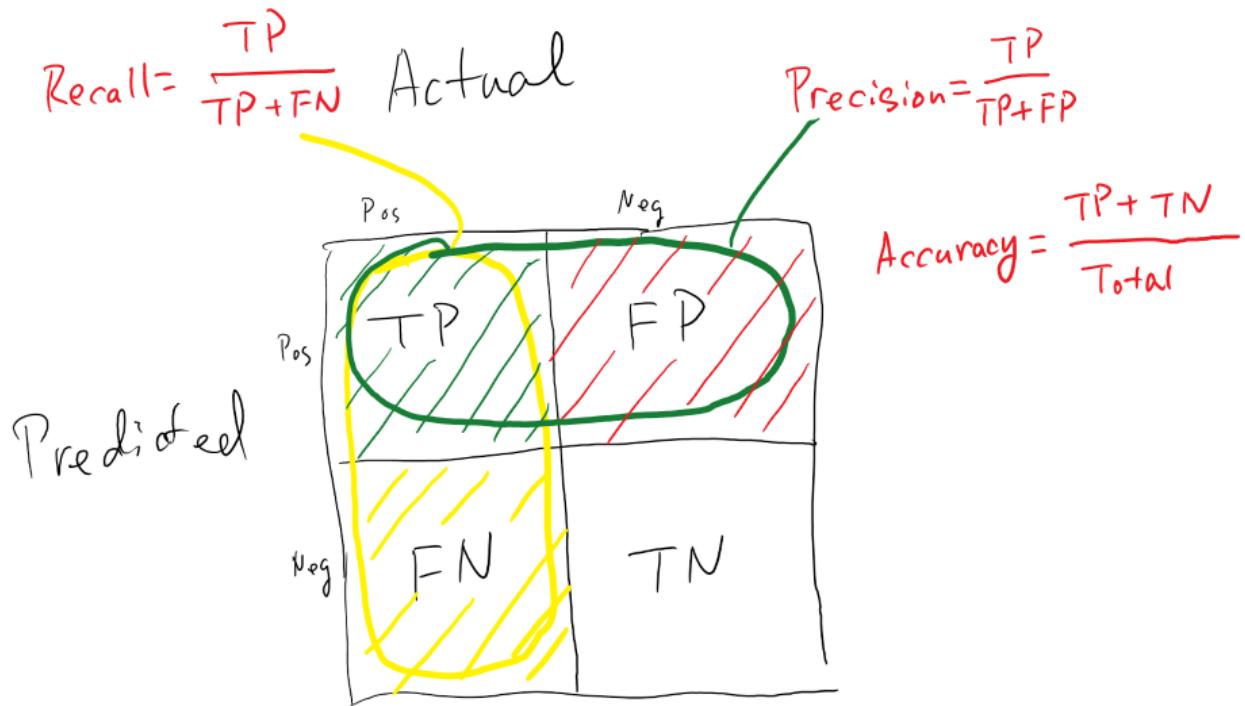
- TP : true positive → predici positive ed è vero
- TN : true negative → predici negative ed è vero
- FP(errore di tipo 1) → predici positivo ed è falso
- FN(errore di tipo 2) → predici negativo ed è falso

N.B : descriviamo valori predetti con NEGATIVE e POSITIVE e valori reali con TRUE e FALSE



Come calcoliamo una confusion matrix per un 2-class classification problem??

| y | y pred | output for threshold 0.6 | Recall | Precision | Accuracy |
|---|--------|--------------------------|--------|-----------|----------|
| 0 | 0.5 | 0 | 1/2 | 2/3 | 4/7 |
| 1 | 0.9 | 1 | | | |
| 0 | 0.7 | 1 | | | |
| 1 | 0.7 | 1 | | | |
| 1 | 0.3 | 0 | | | |
| 0 | 0.4 | 0 | | | |
| 1 | 0.5 | 0 | | | |



RECALL

La seguente equazione può essere spiegata come : "Di tutti i veri Positivi, quanti ne sono stati predetti correttamente".

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

La recall dovrebbe essere la più alta possibile.

PRECISION

La seguente equazione può essere spiegata come : "Di tutti i positivi che abbiamo predetto, quanti ne sono stati predetti correttamente."

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

La precision dovrebbe essere la più alta possibile.

ACCURACY

Di tutte le classi, quante ne ho predetto correttamente.
L'accuracy dovrebbe essere la più alta possibile.

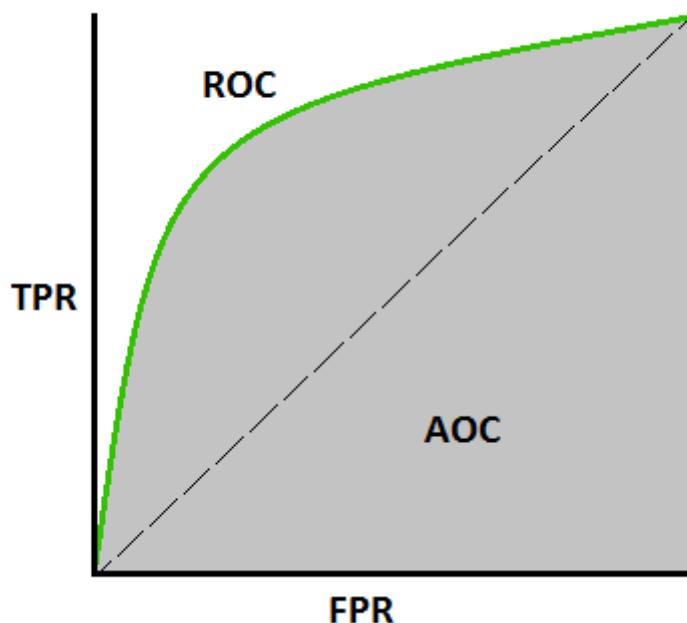
F-MEASURE

E' difficile comparare due modelli con bassa precision e alta recall o vice versa. Per renderli comparabili si utilizza F-score. F-score aiuta a misurare Recall e Precision allo stesso tempo, utilizzando una media armonica al posto di una media aritmetica dando penalizzando molto di più valori estremi.

$$F - MEASURE = \frac{2 * RECALL * PRECISION}{RECALL + PRECISION}$$

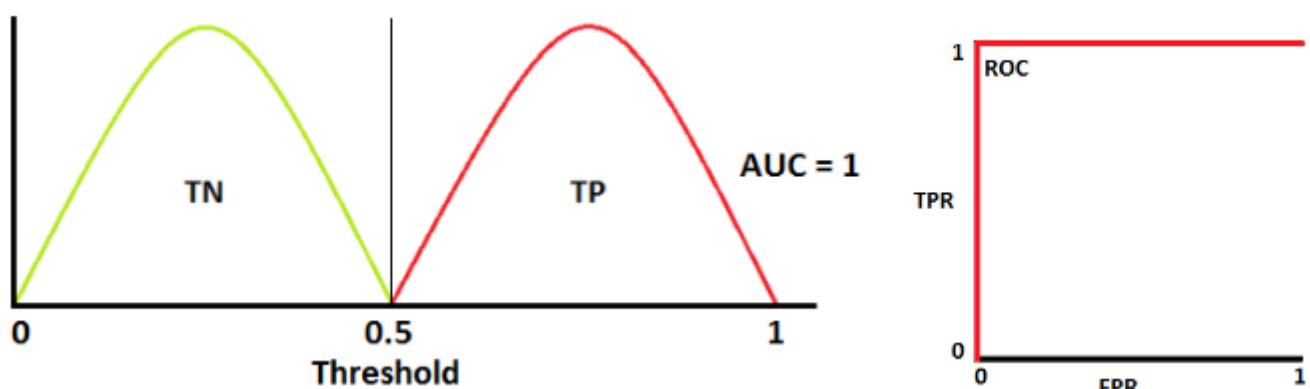
AUC - ROC CURVE

AUC - La curva ROC è una misura delle prestazioni per i problemi di classificazione con varie impostazioni di soglia. La ROC è una curva di probabilità e l'AUC rappresenta il grado o la misura della separabilità. Indica quanto il modello è in grado di distinguere tra le classi. Maggiore è l'AUC, migliore è la capacità del modello di prevedere le classi 0 come 0 e le classi 1 come 1. Per analogia, maggiore è l'AUC, migliore è la capacità del modello di distinguere tra pazienti affetti e non affetti dalla malattia.



Come sappiamo, la ROC è una curva di probabilità. Tracciamo quindi le distribuzioni di queste probabilità:

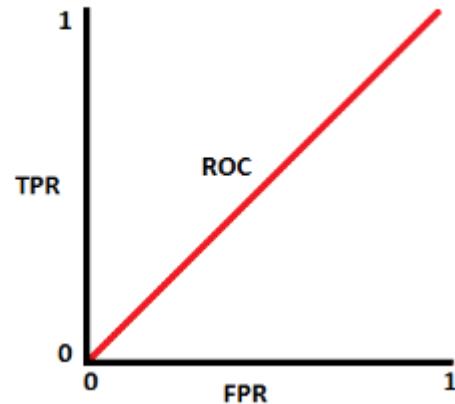
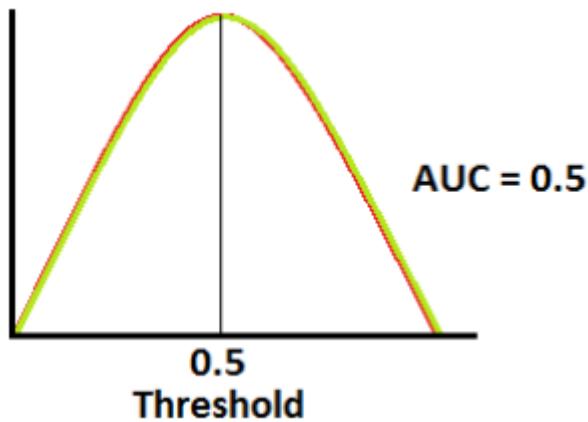
Nota: la curva di distribuzione rossa rappresenta la classe positiva (pazienti con malattia) e quella verde la classe negativa (pazienti senza malattia).



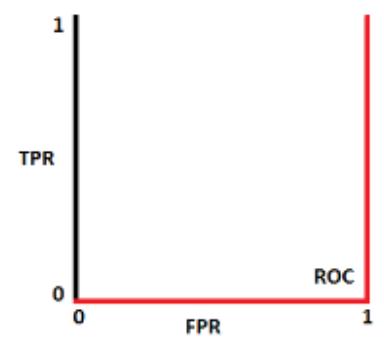
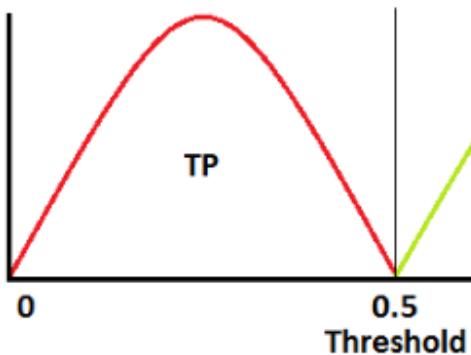
Questa è una situazione ideale. Quando due curve non si sovrappongono affatto significa che il modello ha una misura ideale di separabilità. È perfettamente in grado di distinguere tra classe positiva e classe negativa.

Quando due distribuzioni si sovrappongono, si introducono errori di tipo 1 e 2. A seconda della soglia, possiamo minimizzarli o massimizzarli. A seconda della soglia, possiamo minimizzarli o massimizzarli.

Se l'AUC è pari a 0,7, significa che c'è il 70% di possibilità che il modello sia in grado di distinguere tra classe positiva e classe negativa.



Questa è la situazione peggiore. Quando l'AUC è di circa 0,5, il modello non ha alcuna capacità di discriminazione per distinguere tra classe positiva e classe negativa.



Quando AUC è circa 0 significa che il modello sta invertendo le classi. Ovvero il modello predice una classe negativa come positiva e viceversa.

9. NAIVE BAYES CLASSIFIER

Un classificatore Naive Bayes è un modello di learning automatico probabilistico utilizzato per task di classificazione. Il fulcro del classificatore si basa sul teorema di Bayes.

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

$$\text{Dove } P(B) = \sum_{i=1}^k P(B, a_i) = \sum_{i=1}^k P(B|y_i) * P(a_i)$$

Utilizzando il teorema di Bayes, possiamo trovare la probabilità che A si verifichi, dato che B si è verificato.

In questo caso, B è l'evidenza e A è l'ipotesi. L'ipotesi fatta qui è che i predittori/caratteristiche siano indipendenti. Cioè la presenza di una particolare caratteristica non influisce sulle altre. Per questo motivo viene definita NAIVE.

ES.

Consideriamo il problema del gioco del golf. Il set di dati è rappresentato come segue.

| | OUTLOOK | TEMPERATURE | HUMIDITY | WINDY | PLAY GOLF |
|----|----------|-------------|----------|-------|-----------|
| 0 | Rainy | Hot | High | False | No |
| 1 | Rainy | Hot | High | True | No |
| 2 | Overcast | Hot | High | False | Yes |
| 3 | Sunny | Mild | High | False | Yes |
| 4 | Sunny | Cool | Normal | False | Yes |
| 5 | Sunny | Cool | Normal | True | No |
| 6 | Overcast | Cool | Normal | True | Yes |
| 7 | Rainy | Mild | High | False | No |
| 8 | Rainy | Cool | Normal | False | Yes |
| 9 | Sunny | Mild | Normal | False | Yes |
| 10 | Rainy | Mild | Normal | True | Yes |
| 11 | Overcast | Mild | High | True | Yes |
| 12 | Overcast | Hot | Normal | False | Yes |
| 13 | Sunny | Mild | High | True | No |

Le features rappresentano le condizioni per cui una giornata risulta adeguata per una partita a golf. Osserviamo che una giornata non è adatta al golf se : è troppo calda, è troppo umida, se piove, se non c'è vento.

N.B

è necessario fare 2 assunzioni:

- 1) Tutti i predittori sono considerati indipendenti (es. Una giornata calda non implica che sia anche umida).
- 2) Tutti i predittori hanno lo stesso peso-importanza sulla predizione (es. Il giorno ventoso non ha più importanza nella decisione di giocare o meno)

Per il teorema di bayes possiamo riscrivere il problema così

$$P(y|X) = \frac{P(X|y)*P(y)}{P(X)}$$

La variabile y è la variabile della classe (giocare a golf), che rappresenta se è adatto o meno a giocare a golf date le condizioni. La variabile X rappresenta i parametri/caratteristiche.

$$X = (x_1, x_2, x_3, x_4, \dots, x_n)$$

Il primo termine $P(X|y)$ è detta CLASS-CONDITIONAL PROBABILITY degli attributi data una certa etichetta.

$P(X|y)$ misura la verosimiglianza di osservare una caratteristica x da una distribuzione di istanze che appartengono alla classe y . Se x appartiene ad istanze di classe y ci aspettiamo che il valore sia alto

$$P(y|x_1, \dots, x_n) = \frac{P(x_1|y)P(x_2|y) \dots P(x_n|y)P(y)}{P(x_1)P(x_2) \dots P(x_n)}$$

Ora, è possibile ottenere i valori per ciascuno di essi osservando il set di dati e sostituirli nell'equazione. Per tutte le voci del set di dati, il denominatore non cambia, rimane statico. Pertanto, è possibile rimuovere il denominatore e introdurre una proporzionalità.

Nel nostro caso, la variabile y ha due possibilità SI o NO, multivariata ($c_1 \dots c_k$), dobbiamo trovare la classe y con la probabilità massima

$$y_{predict} = argmax_k P(Y = c_k) * \prod_{i=1}^n P(x_i|Y = c_k)$$

Dobbiamo quindi assegnare l'individuo alla classe per cui tale probabilità è massima, possiamo ottenere la classe dati i predittori.

In modo molto semplice, $P(x_j|c_i)$ può essere precalcolato dal set di dati.

TIPOLOGIE DI NAIVE BAYES CLASSIFIER

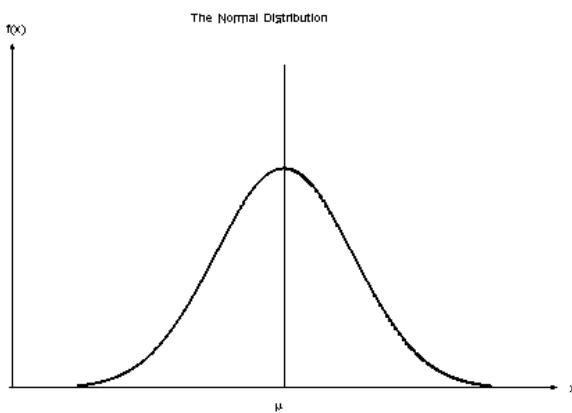
MULTINOMIAL NAIVE BAYES

Viene utilizzato soprattutto per i problemi di classificazione dei documenti, cioè per stabilire se un documento appartiene alla categoria sport, politica, tecnologia, ecc. Le caratteristiche/predittori utilizzati dal classificatore sono la frequenza delle parole presenti nel documento.

Se una feature f_i è categoriale allora $P(X_j|c_i)$ è la frequenza delle istanze di classe c_i dove la feature j ha lo stesso valore dell'istanza X_j diviso la cardinalità di c_i .

GAUSSIAN NAIVE BAYES

Quando i predittori assumono un valore continuo e non discreto, si assume che questi valori siano campionati da una distribuzione gaussiana.



Poiché cambia il modo in cui i valori sono presenti nel set di dati, la formula della probabilità condizionale diventa,

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

COSA SUCCIDE SE $P(x_j|c_i) = 0$?

L'intero prodotto di probabilità collassa a 0 anche se i segnali delle altre features sono molto forti.

LAPLACE CORRECTION : Facciamo finta di osservare un'istanza aggiuntiva per il valore della caratteristica, dove

$$P(x_j|c_i) = \frac{N_{ij} + 1}{N_i + v}$$

Dove v è il numero totale dei valori unici in D. Esistono altre varianti.

L'algoritmo possiede i seguenti punti di forza:

- Lavora bene in caso di "rumore" nei dati.
- Tende a non considerare gli attributi irrilevanti.
- Il training del modello è molto più semplice rispetto ad altri algoritmi.

Il rovescio della medaglia è rappresentato dall'assunzione dell'indipendenza degli attributi, che può non essere presente nella realtà.

10. ARTIFICIAL NEURAL NETWORK

Artificial neural network (ANN) sono potenti modelli di classificazione capaci di apprendere limiti decisionali decisamente complessi e non lineari solamente attraverso i dati.

Storicamente lo studio di una rete neurale artificiale aveva l'obiettivo di emulare un sistema neurale biologicamente reale formato da NEURONI e ASSONI.

Come un sistema reale una ANN è composta da un numero di processing units chiamate NODI connesse ognuna all'altra con dei link diretti.

Il peso dei link tra neuroni rappresenta la forza (l'importanza decisionale) della connessione tra neuroni.

Una ANN cercherà di adattare i pesi dei collegamenti fino a che non corrispondono alle relazioni tra gli input e gli output sottostanti.

La motivazione basica dietro all'utilizzo di una ANN è quella di estrapolare delle feature utili dagli attributi originali che risultano importanti per il task di classificazione.

Inoltre tramite l'utilizzo di nodi interni collegati, modelli ANN sono capaci di estrarre set di feature(magari nascoste), che portano a ottimi risultati nei task di classificazione.

Il paradigma delle 'ANN è contrario a quello convenzionale, in quello classico degli input, vengono dati in pasto ad una "scatola" costruita ad hoc per un compito specifico che poi genererà un output, mentre quello di una ANN è ribaltato.

Avendo a disposizione input e output raccolti in anticipo, e una "scatola" speciale, questa imparerà da sola tramite apprendimento automatico come passare da input ad output.

Questo paradigma funziona abbastanza bene solo che talvolta non abbiamo idea di cosa abbia appreso la scatola, e questo può impedire di migliorare la conoscenza di fenomeni complessi.

STRUTTURA DI UNA ANN

- Una rete neurale con un solo layer è detta PERCEPTRON. Una ANN con multi-layers è detta Artificial Neural Network
- Una ANN può possedere un numero qualsiasi di layers. Ogni layer può avere un numero arbitrario di neuroni. Ogni neurone è connesso con ogni altro neurone del layer precedente, e prende il nome di LAYER DENSO. Ogni layer può avere una diversa FUNZIONE DI ATTIVAZIONE.
- ANN consiste in 2 fasi
 - FORWARD PROPAGATION : questa fase comporta la moltiplicazione dei pesi, aggiunta di BIAS e applicazione delle funzioni di attivazione agli input e la propagazione in avanti
 - BACKPROPAGATION : lo step più importante, comporta la ricerca dei parametri ottimali per il modello propagandosi all'indietro nei layer della rete neurale. Richiede una funzione di ottimizzazione per trovare i pesi ottimali
- ANN può essere applicata a task di regressione e classificazione modificando le funzioni di attivazione.
 - Sigmoide : task classificazione binari
 - SoftMax : task classificazione multi-class
 - Lineare : task di regressione

HYPERPARAMETERS ANN

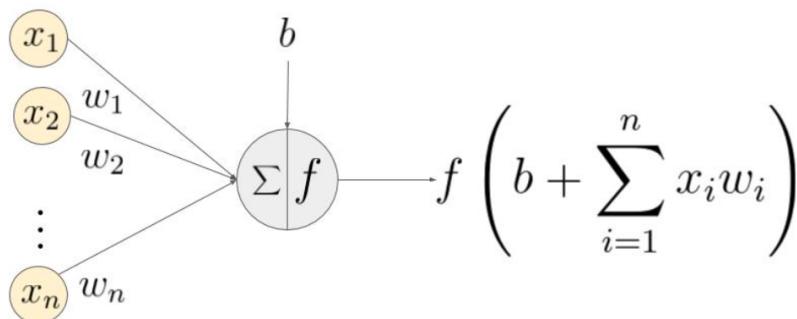
Numero di layers - Numero di neuroni in un layer - funzione di attivazione - inizializzazione dei pesi - loss function - Metrica - optimizer - Numero di epoche.

PERCEPTRON

E' una ANN con 2 tipi di nodi :

- input : usati per rappresentare attributi
- Output : usati per rappresentare l'output del modello

La rete neurale sottostante ha come input x_1, x_2, x_3 e come output



An example of a neuron showing the input ($x_1 - x_n$), their corresponding weights ($w_1 - w_n$), a bias (b) and the activation function f applied to the weighted sum of the inputs.

Dove $f(W^T x) = f(\sum_{i=1}^3 W_i x_i + b)$ dove $f: R \rightarrow R$ è la funzione di attivazione

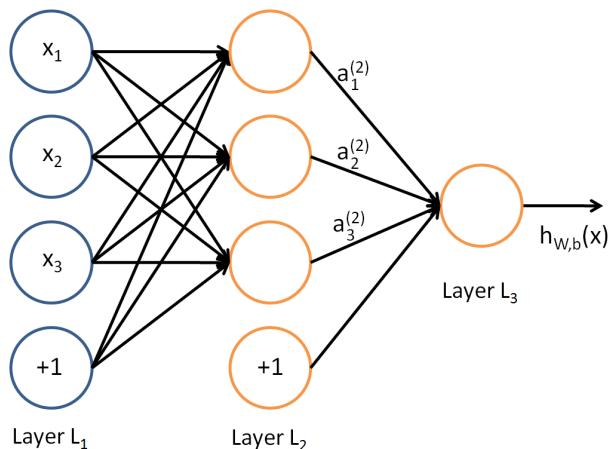
In questo esempio $f(\cdot)$ è la funzione sigmoide:

$$f(z) = \frac{1}{1 + \exp(-z)}$$

Che corrisponde all'input-output di una regressione logistica

MULTI-LAYERS

Una rete neurale è formata da molti neuroni organizzati in layers densi, in modo che l'output di un neurone diventi l'input di un altro neurone nel layer successivo.



I cerchi in blu indicano l'INPUT LAYER, il cerchio più a destra indica l'OUTPUT LAYER, mentre il layer centrale è detto HIDDEN LAYER.

Maggiori sono i numeri di layers e maggiore sarà la capacità espressiva della ANN

FUNZIONAMENTO

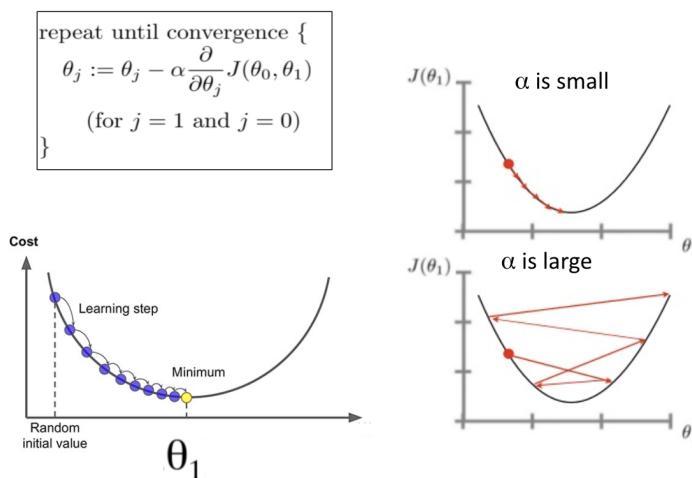
In termini di apprendimento automatico: - un'istanza di train viene presentata all'algoritmo di apprendimento automatico:

- l'algoritmo produce un output o previsione per quell'istanza
- la differenza tra la previsione e l'output desiderato o la etichetta dell'istanza è chiamata LOSS(o errore)
- la perdita viene utilizzata per aggiornare i parametri (pesi) dell'algoritmo di apprendimento automatico - nelle Reti Neurali Artificiali (ANN) questa operazione è chiamata back propagation e, come in molti compiti di apprendimento automatico, si tratta di una discesa del gradiente della funzione di perdita
- il feedback dell'errore è ponderato dal tasso di apprendimento: - un piccolo tasso di apprendimento significa passi piccoli e più cauti
- ripetere su diverse istanze di addestramento, cioè su un insieme di dati di addestramento.
- ripetere l'intero processo più volte: iterazioni di apprendimento o epoch di addestramento.

DISCESA DEL GRADIENTE

La discesa del gradiente è una tecnica che permette di trovare i parametri ottimali che vanno a minimizzare una funzione di loss(per qualunque modello), come ad esempio i Residuals sum of squares per una regressione lineare.

- 1 step : è quello di scegliere una funzione di loss per determinare quanto il nostro modello fitta bene con i dati (es. RSS)
- 2 step : prendere la derivata della funzione di loss per ogni parametro("prenderne il GRADIENTE"),
- 3 step : scegliere inizialmente dei valori casuali per i parametri da ottimizzare, e inserire i parametri nelle derivate calcolate.
- 4 step : una volta calcolata la derivata con i parametri , utilizzarla per calcolare lo Step size: $\text{Step size} = \text{derivata(parameter}_j\text{)} * \text{learning rate}$,
lo step size non è altro che la differenza tra vecchio valore del parametro e nuovo valore del parametro con cui verrà calcolata la derivata della loss function,
- 5 step: i nuovi valori saranno: $\text{new parameter}_j = \text{old parameter}_j - \text{step size}$ e si riparte dallo step3.
Questo perché dobbiamo cercare di effettuare balzi più piccoli man mano che ci si avvicina al punto di minimo.
- Il processo di discesa del gradiente si ferma quando lo step size si avvicina di molto allo 0, solitamente il minimo step size = 0.001 o più piccolo



AGGIORNAMENTO DEI PESI CON DISCESA GRADIENTE

Quando si parla di ottimizzazione la BACKPROPAGATION(algoritmo numerico per il calcolo del gradiente) viene in aiuto. Per prima cosa è necessario trovare una rigorosa RAPPRESENTAZIONE DELL'ERRORE, come ad esempio l'errore quadratico medio MSE. L'MSE calcola la distanza al quadrato tra l'output desiderato e il risultato prodotto dalla ANN.

Avendo ora una funzione che ci dice quanto la rete sta sbagliando possiamo cercare di ottimizzarla. Quello che vogliamo è quindi capire come i pesi w_j interferiscono nel risultato finale, in modo da modificarli.

Quello quindi che vogliamo fare è propagare l'errore da destra verso sinistra cercando di capire come modificare i parametri per ottimizzare il comportamento

Queste quantità sono definibili nella discesa del gradiente come le derivate della loss function rispetto ai pesi.

$$\frac{\delta E}{\delta w_i} \text{ dove } E = \text{loss function} \text{ e } w_i = \text{peso}$$

L'aggiornamento dei pesi sarà dunque:

$$w_i^+ = w_i - \eta * \frac{\delta E}{\delta w_i}, \text{ ovvero sottrarre ai pesi originari, la derivata della funzione di costo}$$

Rispetto al peso in questione (equivalente alla formula della step_size), η è il learning rate, ed è il peso che vogliamo dare al gradiente, più è alto, più l'aggiornamento dei pesi sarà rilevante (grande differenza tra peso vecchio e nuovo).

L'apprendimento della rete viene suddiviso in epoch, un'epoca termina quando vengono elaborati tutti i punti del dataset al fine di ottimizzare i pesi e ridurre l'errore della funzione di costo.

PROBLEMI

Nelle reti neurali durante la propagazione posteriore, ogni peso riceve un aggiornamento proporzionale alla derivata parziale della funzione di errore

VANISHING GRADIENT

Negli strati profondi della rete neurale, l'aggiornamento è ottenuto moltiplicando varie derivate parziali. Se queste derivate parziali sono molto piccole, l'aggiornamento complessivo diventa molto piccolo e si avvicina allo zero. In tal caso, i pesi non saranno in grado di aggiornarsi e quindi la convergenza sarà lenta o assente. Questo problema è noto come problema del VANISHING GRADIENT

EXPLODING GRADIENT

Allo stesso modo, se il termine derivato è molto grande, anche gli aggiornamenti saranno molto grandi. In tal caso, l'algoritmo supererà il minimo e non potrà convergere. Questo problema è noto come problema del gradiente esplosivo. Esistono vari metodi per evitare questi problemi. La scelta della funzione di attivazione appropriata è una di queste.

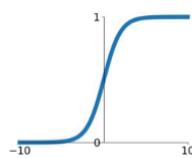
ACTIVATION FUNCTION

Quindi un uso importante della funzione di attivazione è

- mantenere l'uscita limitata a un intervallo particolare.
- Un altro uso della funzione di attivazione è l'aggiunta di non linearità nei dati

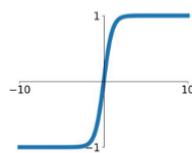
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



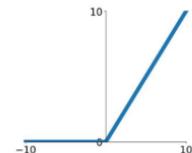
tanh

$$\tanh(x)$$



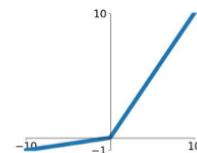
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

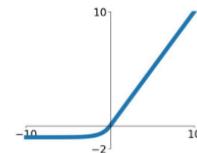


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



SIGMOIDE :

Pro:

- limita output tra 0 - 1
- Previsioni chiare per classificazioni binarie

TANH

Pro:

- Centrata attorno allo zero
- Limita output tra -1 e 1

RELU

Pro:

- Facile da calcolare
- Non ha prob. Vanishing gradient

Contro:

- Exploding gradient
- Non centrata su 0

LEAKY RELU

Pro:

- Facile calcolare
- Non ha prob. Vanishing gradient

ELU

Pro:

- La rete apprende da sola alpha
- Non ha prob. Vanishing gradient

Contro:

- Può causare vanishing gradient
- Non centrata attorno allo zero
- costosa

Contro:

- Problema vanishing gradient
- Costosa

- Attività non totale dei neuroni → rete veloce e efficiente
- Può annullare dei neuroni per sempre → da sempre 0 per valori negativi

- Non annulla neuroni

Contro:

- Exploding gradient
- Non centrata su 0

Contro:

- Difficile da calcolare
- Prestazioni dipendono dal problema

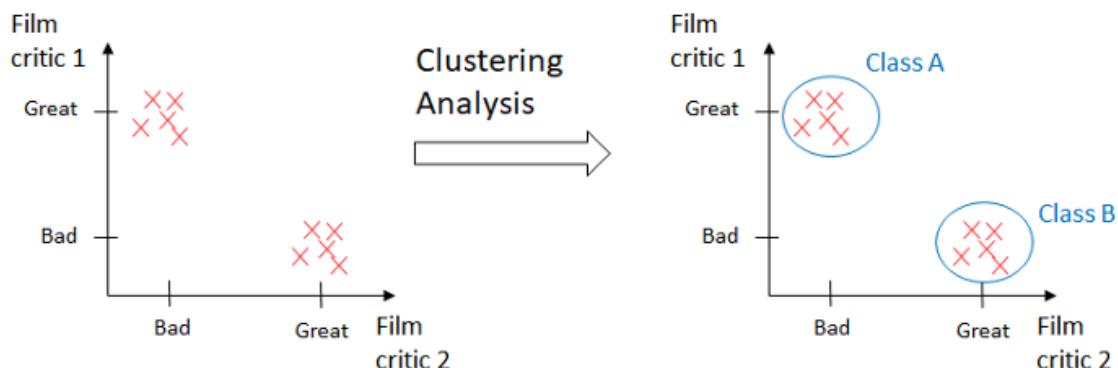
11. CLUSTERING

Quando parliamo di cluster parliamo di una collezione di oggetti dove gli elementi possono essere:

- RELATED → Simili all'interno dello stesso gruppo
- UNRELATED → dissimili all'interno dello stesso gruppo

L'azione di clustering è l'azione di suddivisione degli elementi in cluster differenti

In parole poche, l'obiettivo del clustering è quello di trovare gruppi di elementi all'interno dei dati. Per fare ciò gli algoritmi di clustering trovano delle strutture nei dati, in modo tale che elementi dello stesso cluster siano molto più simili tra loro rispetto a elementi di un cluster differente.



Nell'esempio sopra riportato l'algoritmo di learning riesce ad inferire che esistono due classi differenti senza conoscere nessun'altra informazione dei dati

Gli algoritmi di clustering trovano spazio e utilità nella risoluzione di problemi legati a :

- anomaly detection
- recommendation system
- document grouping
- Raggruppare clientela con gusti simili

Alcuni dei modelli più comuni i clustering sono:

- K-MEANS
- HIERARCHICAL CLUSTERING
- DENSITY BASED SCAN CLUSTERING (DBSCAN)
- GAUSSIAN CLUSTERING MODEL

Quando possiamo dire di essere di fronte ad un buon clustering??

Un buon metodo di clustering produce cluster di alta qualità ove

- 1) alta similarità intra-classe
- 2) bassa similarità intra-classe

CRITERI DI ANALISI DEL CLUSTER

- Criterio di partizionamento :
 - Single level : classi allo stesso livello
 - Hierarchical(Gerarchico) : classi che si specializzano (PREFERIBILE)

- Separazione del cluster
 - Esclusiva: 1 oggetto in una classe
 - Non esclusiva : 1 oggetto in più classi
- Misura di similarità :
 - Distance-based (euclidea, vettoriale, ...)
 - Connectivity-based (densità o continuità)
- Spazio di clustering :
 - Full space : tutte le feature
 - Sub Space : sottoinsieme di feature

K-MEANS ALGORITHM - SINGLE LEVEL

Vantaggi:

- Algoritmo di semplice implementazione,
- molto efficiente in termini computazionali, motivo per cui è molto popolare
- Termina ad un ottimo locale : soluzione ottimale

Svantaggi:

- Non molto bravo nell'identificazione di classi la cui distribuzione non è sferica
- Sensibile ad outliers e dati rumore
- Applicabile solo ad oggetti "continui"
- Il centroide calcolato non è necessariamente un punto del dataset, e può essere un bottleneck per l'interpretabilità.

L'algoritmo punta a trovare e raggruppare in classi gli elementi che hanno una similarità molto alta tra loro. Nei termini dell'algoritmo, questa similarità è intesa come l'opposto della distanza tra i punti dati. Più i dati sono vicini, più sono simili e più è probabile che appartengano allo stesso gruppo

Concetti CHIAVE

- DISTANZA EUCLIDEA : è la metrica più comune utilizzata da k-means. Un esempio di distanza tra 2 punti x e y in uno spazio di m dimensione è :

$$d(x, y)^2 = \sum_{j=1}^m (x_j - y_j)^2 = \|x - y\|_2^2$$

dove , j è la j-esima dimensione(feature column) del campione di punti x e y.

- SUM of SQUARE ERROR (SSE) : definiamo SSE nel seguente modo

$$SSE = \sum_{i=1}^k \sum_{p \in C_i} \|p - c_i\|^2$$

K : n° cluster

c_i : cluster i-esimo

c_i : centro del cluster i-esimo

P : punto appartenente al cluster i-esimo

Per avere il minimo errore di clustering minimizziamo SSE e estraiamo il centro ideale per ogni cluster (c_i) troviamo :

$$c_i = \frac{\sum_{p \in C_i} p}{|c_i|}$$

ALGORITMO

Algorithm 1 Lloyd

- 1: choose k as the number of clusters
 - 2: randomly choose k datapoints as centroids
 - 3: **repeat**
 - 4: **for each** centroid **do**
 - 5: reassign each point to its closest centroid
 - 6: recalculate centroid as mean over all points assigned
 - 7: **end for**
 - 8: **until** convergence
-

1. Non conoscendo le classi presenti nel dataset di ingresso, la prima cosa da fare è decidere il numero di classi (o meglio cluster, in questo caso) in cui si vuole suddividere il dataset stesso. Questo numero è detto K , da cui il nome del metodo K-means (il termine means sottintende l'uso dei centroidi, cioè di punti medi).
2. Si scelgono in modo casuale K centroidi appartenenti allo spazio delle features. L'unica condizione è che non siano coincidenti, anzi solitamente ci si assicura che siano abbastanza distanti tra loro. In caso contrario l'algoritmo potrebbe avere problemi a convergere.
3. Si calcola la distanza di ogni punto del dataset rispetto ad ogni centroide.
4. Ogni punto del dataset viene associato al cluster collegato al centroide più vicino.
5. Si ricalcola la posizione di ogni centroide facendo la media delle posizioni di tutti i punti del cluster associato (solo di questi punti!)
6. Si itera dal punto 3 fino a quando non ci sarà più alcun ingresso che cambia di cluster.

Efficienza :

- efficiente $O(t*K*n)$ dove : $\rightarrow K, t \ll n \Rightarrow O(n)$
 - t : numero iterazioni
 - K : n° cluster
 - n : n° di oggetti

K-MEDOID METHODS (PAM ALGORITHM) - SINGLE LEVEL

Vantaggi:

- Risolve il problema di sensibilità al rumore/outliers
- Non viene più calcolato un centroide (che può non essere un punto del dataset) ma viene preso un punto reale del cluster come "nuovo" punto detto \leftarrow MEDOIDE
- Possibilità di utilizzare metriche differenti oltre alla distanza euclidea

N.B: ogni punto del dataset può essere preso come medoide solo e solo se la sua dissimilarità da tutti gli altri punti nel cluster è minima (STEP 7)

Note dissimilarity is calculated as the summation of the absolute difference between medoids and data points. E.g., $D = |M_x - \text{Feature } X| + |M_y - \text{Feature } Y|$

ES. let $K=2$ $M_1=(4,5)$ and $M_2=(9,10)$

| Data Points | Feature X | Feature Y | Dissimilarity from M1 | Dissimilarity from M2 |
|-------------|-----------|-----------|-----------------------|-----------------------|
| 0 | 7 | 8 | 6 | 4 |
| 1 | 9 | 10 | 10 | 0 |
| 2 | 11 | 5 | 7 | 7 |
| 3 | 4 | 9 | 4 | 6 |
| 4 | 7 | 5 | 3 | 7 |
| 5 | 2 | 3 | 4 | 14 |
| 6 | 4 | 5 | 0 | 10 |

ALGORITMO

Require: K , number of clusters; D , a data set of N points

Ensure: A set of K clusters

- 1: Arbitrarily choose K points in D as initial representative points.
- 2: **repeat**
- 3: **for** each non-representative point p in D **do**
- 4: find the nearest representative point and assign p to the corresponding cluster.
- 5: **end for**
- 6: randomly select a non-representative point p_{rand} ;
- 7: compute the overall cost C of swapping a representative point p_i with p_{rand} ;
- 8: **if** $C < 0$ **then**
- 9: swap p_j with p_{rand} to form a new set of K representative points.
- 10: **end if**
- 11: **until** stop-iteration criteria satisfied
- 12: **return** clustering result.

L'aggiornamento dei centroidi in K-mean avveniva computando la media di tutti i punti presenti nel cluster.

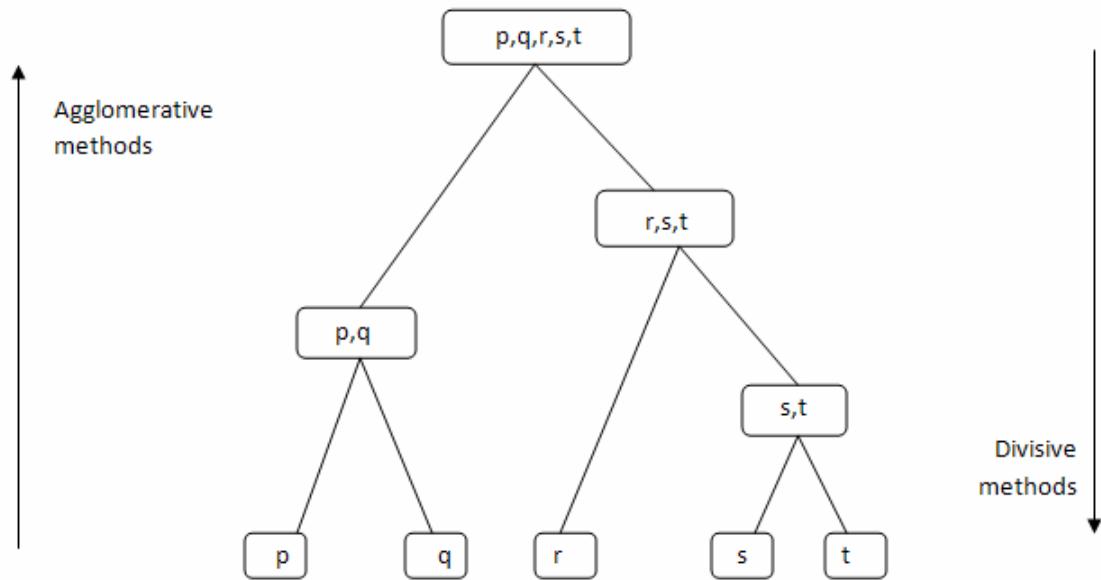
In PAM invece l'aggiornamento è differente.

Se nel cluster ci sono m -punti, scambia il medoide con tutti gli altri $m-1$ punti del cluster e finalizza come nuovo medoide, il punto che ha lo loss minima.

HIERARCHICAL CLUSTERING

Esistono due tipologie di cluster gerarchico:

- Agglomerativo : approccio BOTTOM UP. Si inizia con molti cluster di piccole dimensioni e poi si uniscono assieme creando un cluster più grande.
- Divisivo : approccio di tipo TOP DOWN. si parte da una singolo grande cluster e si spezza in cluster più piccoli agendo in maniera ricorsiva fino ad ottenere il numero di cluster desiderati.



Vantaggi:

- Non necessitiamo di fare assunzioni su un numero preciso di cluster
- Potrebbero corrispondere a tassonomie significative

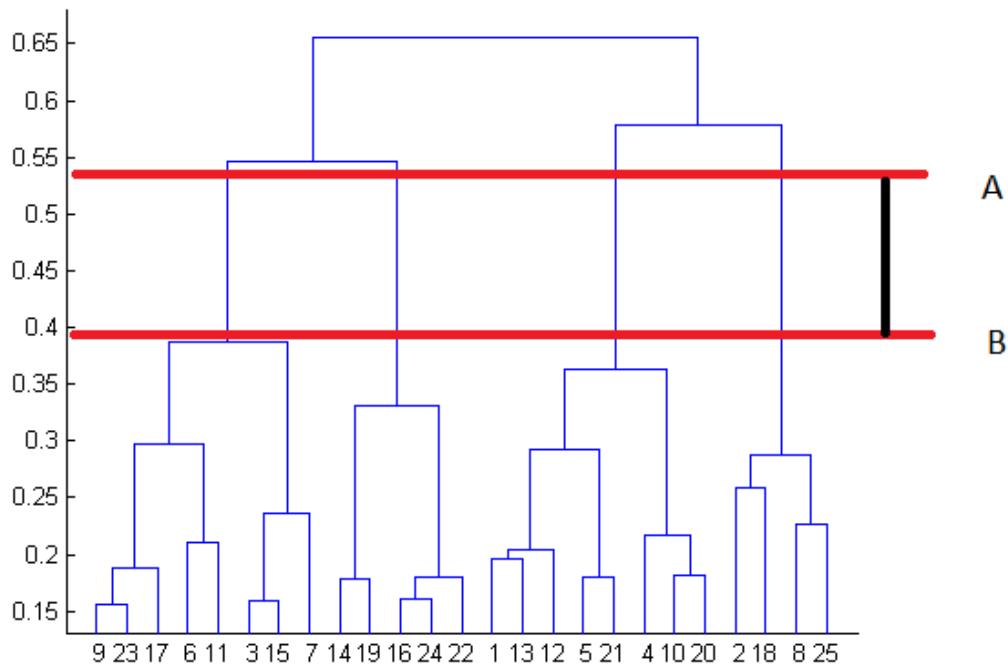
Svantaggi:

- Una volta deciso di unire 2 cluster, non si può tornare indietro, no backtracking
- Troppo lento per dataset grandi $\Rightarrow O(n^2 \log(n))$

E' possibile utilizzare un DENDOGRAMMA per visualizzare la storia di raggruppamento e cercare il numero ottimale di cluster.

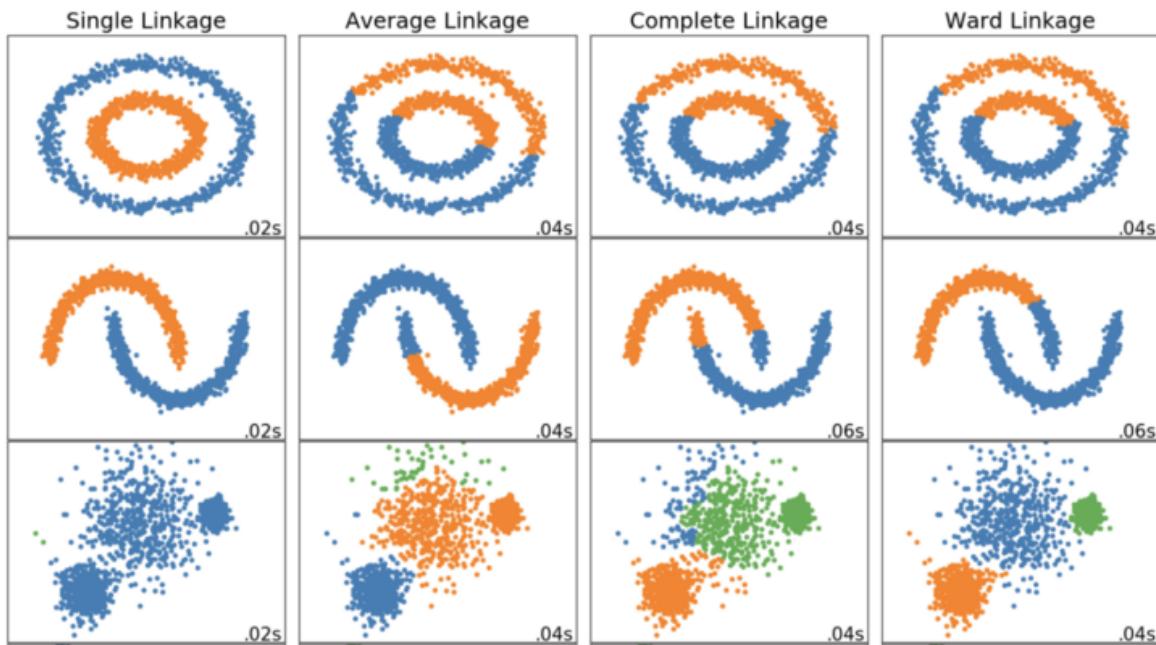
- Cercare la sezione del dendogramma con maggiore ampiezza verticale che non interseca in nessuno degli altri cluster.
- Disegnare 2 linee alle estremità della sezione
- Il numero ottimale di cluster è uguale alla somma delle linee verticali che sezionano la zona delimitata dalle linee orizzontali

Nell'esempio seguente il numero di cluster ottimale è = 4



LINKAGE CRITERIA

Similarmente alla discesa del gradiente è possibile modificare alcuni parametri per ottenere risultati drasticamente diversi



Il linkage criteria fa riferimento a com'è calcolata la distanza tra cluster

METODI PER CALCOLARE SIMILARITA' TRA CLUSTER:

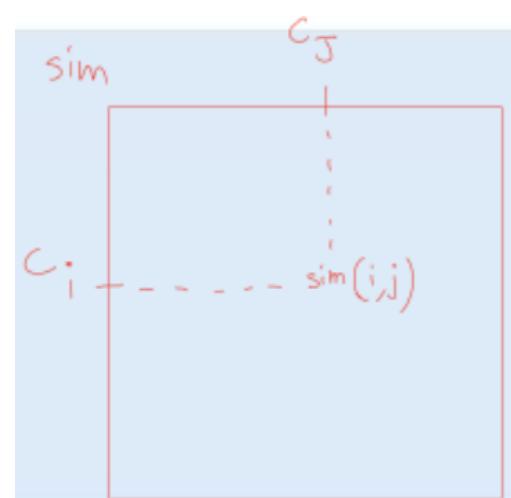
- Distanza tra punti più vicini di cluster diversi (Single Linkage)
- Distanza tra punti più distanti in cluster diversi (Complete Linkage)
- Distanza media tra tutti i punti in 2 cluster diversi (Average Linkage)
- Distanza tra centroidi di 2 cluster diversi
- Distanza è la somma delle differenze al quadrato all'interno di tutti i cluster (Ward Linkage)

| Linkage Method | Pros | Cons |
|----------------|--|--|
| MIN | This approach can separate non-elliptical shapes as long as the gap between two clusters is not small. | MIN approach cannot separate clusters properly if there is noise between clusters. |
| MAX | MAX approach does well in separating clusters if there is noise between clusters. | Max approach is biased towards globular clusters and Max approach tends to break large clusters. |
| Group Average | The group Average approach does well in separating clusters if there is noise between clusters. | The group Average approach is biased towards globular clusters. |
| Ward's Method | Ward's method approach also does well in separating clusters if there is noise between clusters. | Ward's method approach is also biased towards globular clusters. |

HIERARCHICAL-AGGLOMERATIVE CLUSTERING ALGORITHM

HAC Algorithm.

1. § Initialize every single point as a cluster
2. **for** $i = 1 \dots |D|$:
3. $C_i = \{o_i\}$
4. § set of clusters
5. $\mathcal{C} = \bigcup_i \{C_i\}$
6. § Aggregation loop
7. **while** $|\mathcal{C}| > 1$:
8. § Find most similar cluster pair
9. $C_i, C_j = \arg \max_{C_x, C_y} sim(C_x, C_y)$
10. § Replace C_i and C_j with the merge of the two
11. $C_{new} = C_i \cup C_j$
12. $\mathcal{C} = ((\mathcal{C} \setminus \{C_i\}) \setminus \{C_j\}) \cup \{C_{new}\}$



Step 11-12 :
Aggiungo l'unione e tolgo
 C_i e C_j

This is usually implemented through a similarity matrix S of size $|D| \times |D|$:

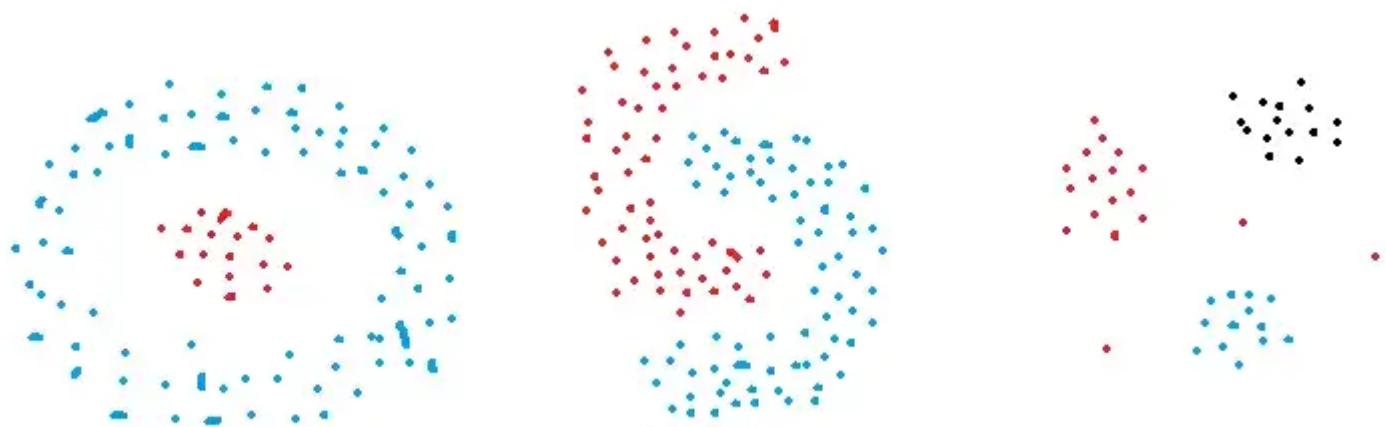
- initially this is initialized as $S[i, j] = sim(o_i, o_j)$
- when merging C_i with C_j :
 - the column and the row corresponding to cluster C_j are invalidated, e.g., by marking $\forall k, S[j, k] = -\infty$ and $\forall k, S[k, j] = -\infty$
 - the column/row corresponding to cluster C_i is used for the new cluster $C_{new} = C_i \cup C_j$, i.e., by setting $\forall k, S[i, k] = sim(C_{new}, C_k)$ and $\forall k, S[k, i] = sim(C_k, C_{new})$
- note that, as similarity measures are typically symmetric, a upper/lower triangular matrix would suffice.

COMPLESSITA'

- Implementazione semplice: computare n volte similarità di n cluster
 - Poiché dobbiamo eseguire n iterazioni e in ogni iterazione dobbiamo aggiornare la matrice di similarità e ripristinare la matrice, la complessità temporale è molto elevata. La complessità temporale è dell'ordine del cubo di n.
 - $\rightarrow (n * n^2) \Rightarrow O(n^3)$
 - Solo per single linkage invece $\rightarrow O(n^2)$ dove n è il numero di punti
- Migliore implementazione: costruire un min Heap per ogni oggetto
 - un heap in cui sono memorizzate le distanze di tutte le coppie di cluster. La coppia di cluster più vicina è data dall'elemento del nodo radice dell'albero binario corrispondente all'heap.
 - L'aggiornamento dell'heap a ogni stadio della gerarchia è facilmente attuabile spostando verso l'alto o verso il basso gli elementi dell'heap lungo il percorso dell'albero dell'heap. Il tempo di calcolo dell'algoritmo è al massimo $O(N^2 \log(N))$ quando devono essere classificati N oggetti.
 - $\rightarrow n * n \log n \Rightarrow O(n^2 \log n)$ dove n è il numero di punti

DENSITY BASED CLUSTERING

Clustering gerarchico e partition clustering sono efficienti con cluster di forma regolare. Tuttavia quando si tratta di cluster di forma arbitraria o individuare outliers, tecniche density-based sono molto più efficiaci.



I Punti in queste figure sono distribuiti in maniera arbitraria, gli algoritmi density-based sono molto efficaci per trovare le regioni di densità e gli outliers.

DBSCAN ALGORITHM - DENSITY

Dbscan sta per density-based-spatial clustering of application with noise.
È in grado di trovare forme arbitrarie in cluster e in cluster con rumore.

L'idea principale di DBSCAN è che un punto appartiene ad un cluster se è vicino a molti punti di quel cluster.

Quindi:

- q è Vicino del punto p sono :

$$N_{\epsilon}(p) = \{ q \in \text{Dataset} \mid \text{dist}(q, p) \leq \epsilon \}$$

Ci sono due parametri chiave in DBSCAN :

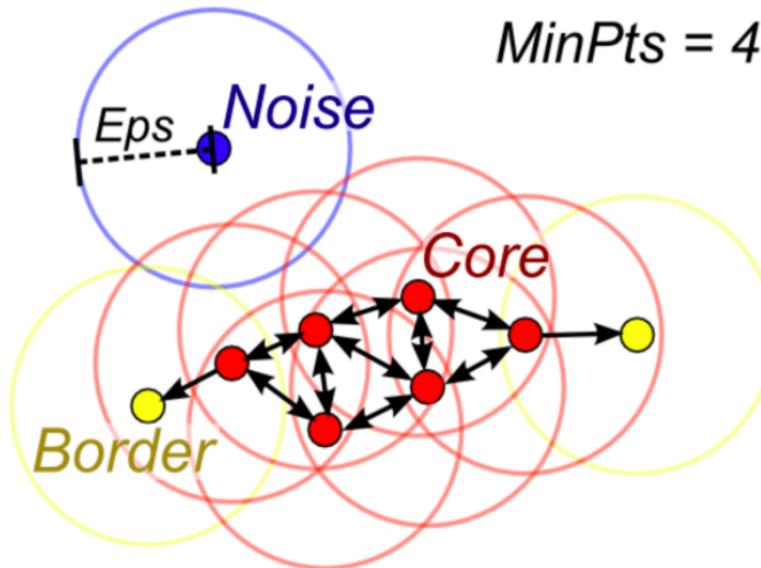
- eps: la distanza che definisce un vicino. 2 punti sono vicini se la loro distanza è $\leq \epsilon$.
- minPts: numero minimo di punti per definire un cluster

Sulla base di questi due parametri i punti vengono classificati in :

- Core Point: un punto è un core se ci sono almeno un numero $\geq \text{minPts}$ di punti considerando il core stesso nella sua area di intorno di raggio ϵ

$$|N_{\epsilon}| \geq \text{MinPts}$$

- Border point : un punto è un border se è raggiungibile da un core e ci sono un numero $< \text{minPts}$ nella sua area di intorno.
- Outliers: un punto è un outlier se non è un core e non è raggiungibile da nessun altro core.



Red: Core Points

Yellow: Border points. Still part of the cluster because it's within epsilon of a core point, but not does not meet the min_points criteria

Blue: Noise point. Not assigned to a cluster

Un punto q è detto directly Density Reachable da p sse:

- p è core
- $q \in N_\epsilon(p)$

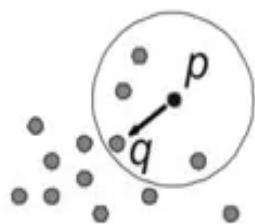
Un punto q è detto Density Reachable da p sse:

- \exists una catena di punti p_1, \dots, p_n con $p_1 = p$ e $p_n = q$
- $\forall i: p_{i+1}$ è direttamente density reachable da p_i

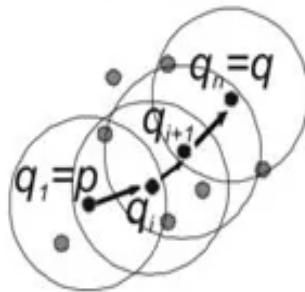
Un punto q è detto Density connected da p sse:

- \exists un punto o tale per cui p è density reachable da o e q è density reachable da o

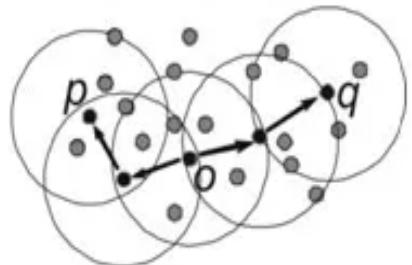
a Directly density-reachable



b Density-reachable



c Density-connected



ALGORITMO

DBScan Algorithm

```

1. § Empty set of clusters
2.  $C \leftarrow \emptyset$   $\rightarrow$  insieme di diversi cluster
3. § Initially all objects are marked as unvisited, and not noisy
4. for each  $p \in D$ :
5.   visited[ $p$ ]  $\leftarrow$  False } non visitato
6.   noise[ $p$ ]  $\leftarrow$  False } non è rumore
7. § Process all unvisited points
8. for each  $p \in D$ :
9.   if visited[ $p$ ]:
10.    continue
11.   visited[ $p$ ]  $\leftarrow$  True
12.   § Check neighborhood density
13.   if  $|N_\epsilon(p)| < MinPts$ :
14.     Mark as noise and go to the next point
15.     Note that the noise label might be updated later
16.     noise[ $p$ ] = True
17.     continue
18.   § Build a cluster starting from  $p$ 
19.    $C \leftarrow \{p\}$   $\rightarrow$  cluster corrente
20.   § Create a queue of candidate points for a recursive visit
21.    $Q \leftarrow N_\epsilon(p)$ 
22.   while  $Q \neq \emptyset$ :
23.     § Get an point from  $Q$ 
24.      $q = Q.pop()$ 
25.     § Add to  $Q$  unvisited core points in the neighborhood
26.     if !visited[ $q$ ]:
27.       visited[ $q$ ] = True
28.       if  $|N_\epsilon(q)| \geq MinPts$ :
29.          $Q \leftarrow Q \cup N_\epsilon(q)$ 
30.       § If not in any cluster then add to the current cluster
31.       § This might include noisy but density-reachable points
32.       if  $q \notin C, \forall C \in C$ : }  $\rightarrow$   $q$  per stare nel cluster
33.          $C \leftarrow C \cup \{q\}$  } non deve appartenere
34.         noise[ $q$ ]  $\leftarrow$  False } a nessun altro
35.       § Add  $C$  to the set of clusters
36.        $C \leftarrow C \cup \{C\}$ 

```

} initializzazione

} \rightarrow se non è un core point, passa al prossimo

} \rightarrow density-reachable
(aggiungiamo a Q i vicini dei vicini che sono core points)

} \rightarrow q per stare nel cluster non deve appartenere a nessun altro

Complessità $\rightarrow O(n \log n)$

VANTAGGI:

- Lavora bene e gestisce molto bene il rumore
- Gestisce forme arbitrarie di cluster
- Non necessita di specificare il numero di cluster
- Necessita di solamente 2 parametri

SVANTAGGI:

- Molto sensibile ai valori degli iperparametri
- Difficile trovare i migliori iperparametri in dataset con alta varianza
- Se il dataset è molto grande e la metrica di misura della distanza è la distanza euclidea si può facilmente incorrere nella problematica della CURSE OF DIMENSIONALITY (capitolo successivo)

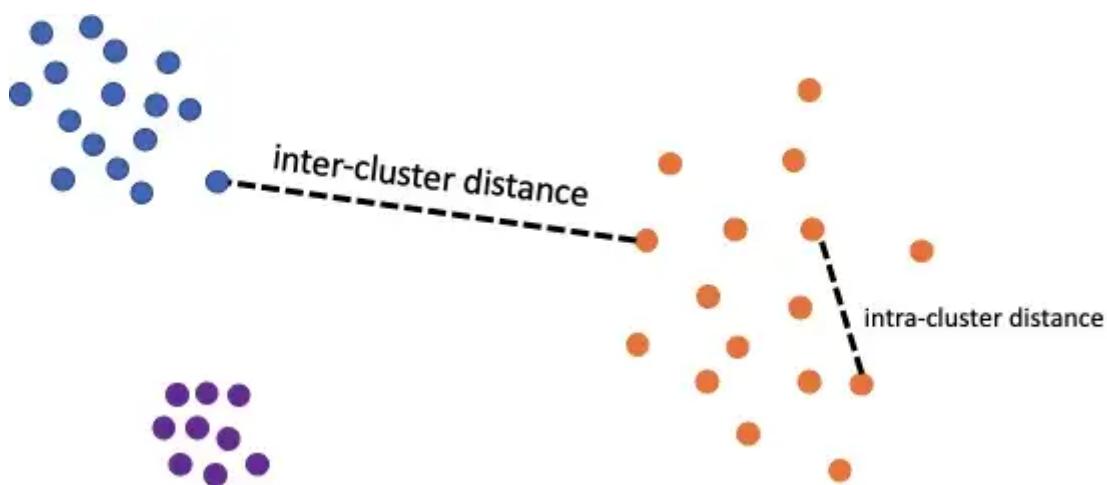
12. CLUSTERING EVALUATION

La valutazione di bontà del clustering può essere :

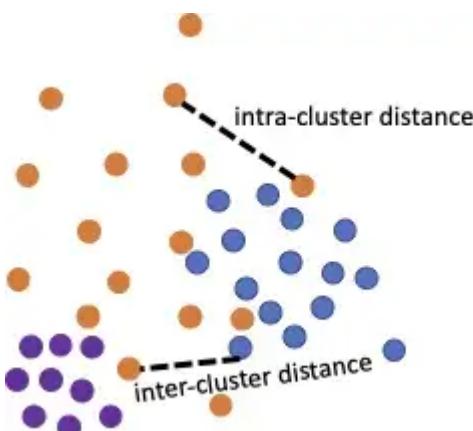
- INTRINSECA : non abbiamo nessuna verità oggettiva con cui confrontare il nostro risultato
- ESTRINSECA : abbiamo qualche tipo di verità oggettiva

L'idea della valutazione di bontà INTRINSECA è semplice. Si utilizza la DISTANZA come VERITA' ASSOLUTA. Infatti viene comparata l' INTRA-CLASS DISTANCE e la INTER-CLASS distance, per decidere quanto bene i cluster sono separati.

Un buon clustering dovrebbe avere una piccola intra-class distance e una larga inter-class distance



Tuttavia , se il clustering è fatto male, l'inter-class distance e l'intra-class distance non sono distinguibili.



Notiamo che quando parliamo di clustering buono o cattivo, il clustering si riferisce all'assegnazione dell'etichetta del cluster al punto.

Per lo stesso set di dati il clustering con un metodo potrebbe essere buono, ma con un altro metodo potrebbe non esserlo.

Anche la metrica utilizzata dal processo di clustering è molto importante.

INTRA-CLASS SIMILARITY (intrinseca)

$\forall o_i$ in un cluster C_h possiamo misurare :

$$a(o_i) = \frac{\sum_{\substack{o_j \in C_h, o_j \neq o_i}} dist(o_i, o_j)}{|C_h| - 1}$$

Definiamo la intra class similarity del punto o_i come la somma delle distanze tra il punto o_i e tutti i punti $o_j \in C_h$ fratto il numero di punti nel cluster meno se stesso quindi 1.

INTER-CLASS SIMILARITY (intrinseca)

$$b(o_i) = \min_{\forall C_k \neq C_h} \left\{ \frac{\sum_{o_j \in C_k} dist(o_i, o_j)}{|C_k|} \right\}$$

Definiamo la inter class similarity del punto o_i come la distanza minima tra il punto o_i del cluster C_h e i punti del cluster C_k fratto la cardinalità (numero di punti totali) del cluster C_k

SILHOUETTE COEFFICIENT (intrinseca)

Il coefficiente di Silhouette, una delle metriche di valutazione del clustering più utilizzate, sintetizza il confronto tra le distanze intra/inter cluster in un punteggio compreso tra -1 e 1. Un valore vicino a 1 indica un ottimo risultato di clustering, in cui le distanze inter-cluster sono molto più grandi di quelle intra-cluster.

Un valore vicino a 1 indica un ottimo risultato di clustering, in cui le distanze inter-cluster sono molto più grandi di quelle intra-cluster; mentre un valore vicino a -1 indica un'assegnazione di cluster completamente sbagliata, in cui le distanze inter-cluster non sono nemmeno paragonabili a quelle intra-cluster.

$$s(o_i) = \frac{a(o_i) - b(o_i)}{\max\{a(o_i) - b(o_i)\}}$$

SVANTAGGIO: il coefficiente di Silhouette è computazionalmente costoso. Il tempo di esecuzione estremamente lungo su un set di dati relativamente grande lo rende meno utile nelle applicazioni del mondo reale.

NB: possiamo dire che gli algoritmi di machine learning imparano tutti a loro modo una misura di similarità

EXTRINSIC EVALUATION

Definiamo come:

- $C(o_j)$: id del cluster associato all'oggetto o_j
- $L(o_j)$: id label vera di o_j

Possiamo quindi definire :

$$TP = |\{o_i, o_j \mid C(o_i) = C(o_j) \wedge L(o_i) = L(o_j)\}|$$

$$FP = |\{o_i, o_j \mid C(o_i) = C(o_j) \wedge L(o_i) \neq L(o_j)\}|$$

$$TN = |\{o_i, o_j \mid C(o_i) \neq C(o_j) \wedge L(o_i) \neq L(o_j)\}|$$

$$FN = |\{o_i, o_j \mid C(o_i) \neq C(o_j) \wedge L(o_i) = L(o_j)\}|$$

Utilizzate in :

$$RAND\ STATISTIC = \frac{TP}{TP+FP+TN+FN}$$

$$JACCARD\ COEFFICIENT = \frac{TP}{TP + FP + FN}$$

La statistica Rand misura il numero di coppie per le quali è stata presa una decisione di clustering corretta.

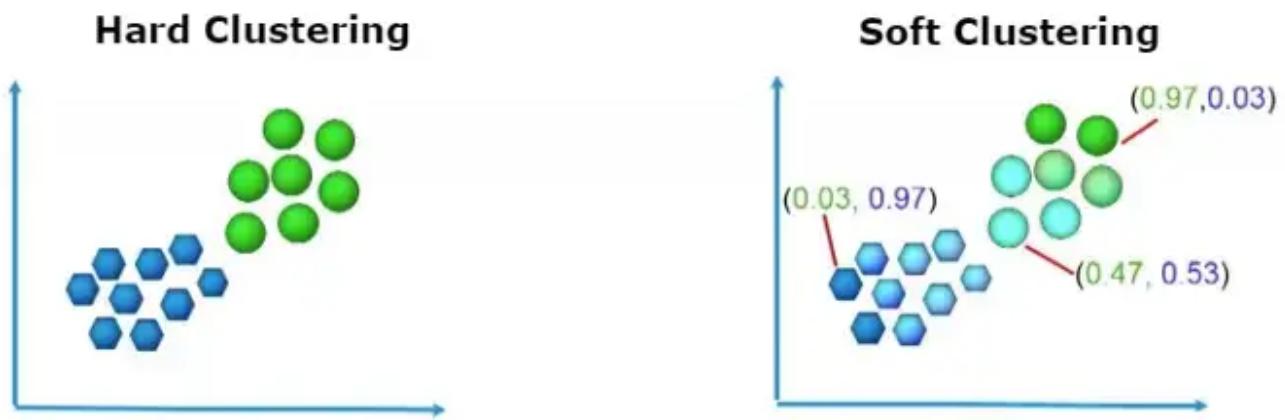
Quando il numero di classi/cluster è elevato, è probabile che il numero di Veri Negativi aumenti (in un clustering casuale è probabile che ci sia un gran numero di TN), portando il denominatore ad aumentare e quindi la rand statistic diventerebbe generalmente grande. Il coefficiente di Jaccard non tiene invece conto dei veri negativi.

13. CLUSTER ANALYSIS

Fin'ora abbiamo visto forme di partitioning (hard clustering)

Talvolta però in determinati scenari il partitioning è un'assunzione troppo forte da fare.

Preferiamo invece avere un determinato grado di inclusione di un oggetto all'interno dei cluster, il grado di inclusione viene espresso attraverso una probabilità di appartenenza al cluster.



Dato un dataset di m istanze e un set di K clusters $C = \{C_1, \dots, C_k\}$, contrassegnamo con w_{ij} il grado di appartenenza dell'istanza x_i al cluster C_j .

Vengono seguite le seguenti condizioni:

- Per ogni istanza x_i : $\sum_{j=1}^k w_{ij} = 1$
- Per ogni cluster C_j : $0 < \sum_{i=1}^m w_{ij} < m$

Il valore di w_{ij} definisce il cosiddetto FUZZY CLUSTERING.

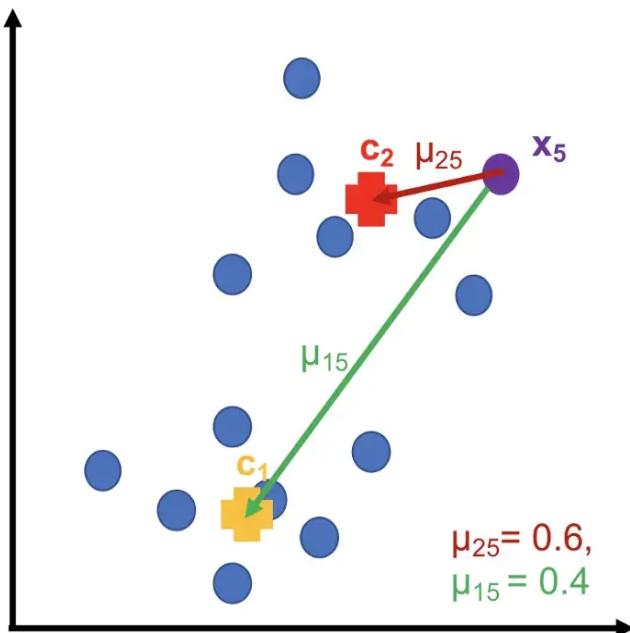
FUZZY C-MEANS

Il termine "Fuzzy" sta per "non certo", questo indica che siamo in presenza di un metodo di soft clustering.

Fuzzy C-means è un algoritmo di soft clustering dove ogni punto possiede un grado di appartenenza a ogni cluster, espresso in forma di probabilità con valore : $0 < x < 1$. Questo algoritmo presenta 2 parametri:

- w_{ij} : grado di appartenenza (membership value),
- c_j : centro del j-esimo cluster
- p : fuzzyfier che controlla la fuzzy-ness, quando p aumenta, c'è più fuzzy-ness sui margini del cluster

ES.



In questo esempio, ponendo attenzione al punto x_5 , e supponendo che conosciamo ci siano solo 2 cluster:

- $\text{prob}_{x_5}(c1) = 0.6$
- $\text{prob}_{x_5}(c2) = 0.4$

$0.6 + 0.4 = 1$ il vincolo di distribuzione della probabilità del punto x_5 è rispettato.

FUNZIONE OBIETTIVO:

$$SSE = \sum_{j=1}^K \sum_{x_i \in C_k} w_{ij}^p * \text{dist}(x_i, c_j)^2$$

Dalla funzione riportata sopra possiamo capire come sia la somma delle distanze dei punti x_i ai centroidi dei cluster c_i .

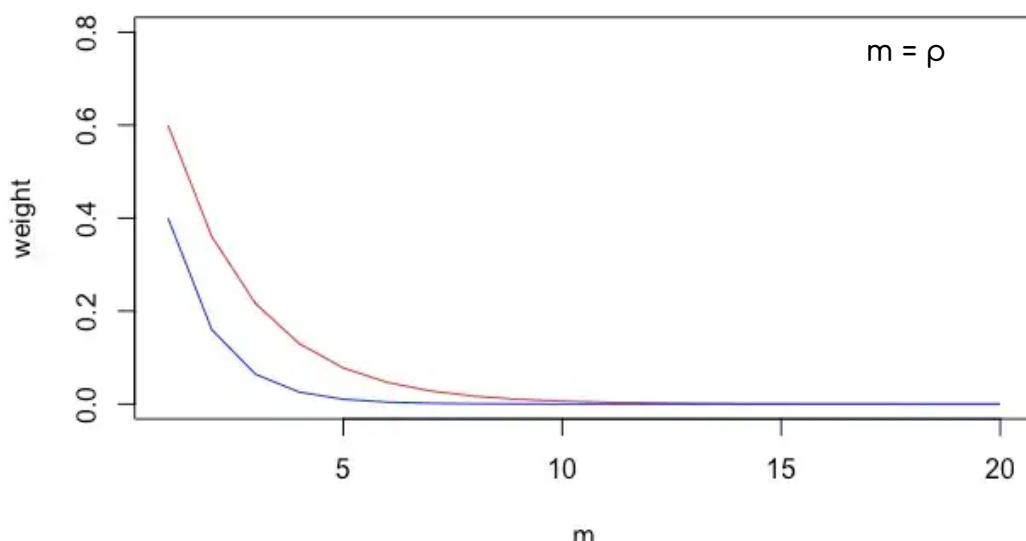
Se $p=1$, allora se facciamo riferimento come nell'esempio sopra ai vettori che rappresentano delle probabilità, la formula non è altro che la probabilità-pesata della somma delle distanze tra i punti e i centroidi.

Ciò significa che i punti molto vicini ad un centroide avranno pesi maggiori per quel cluster.

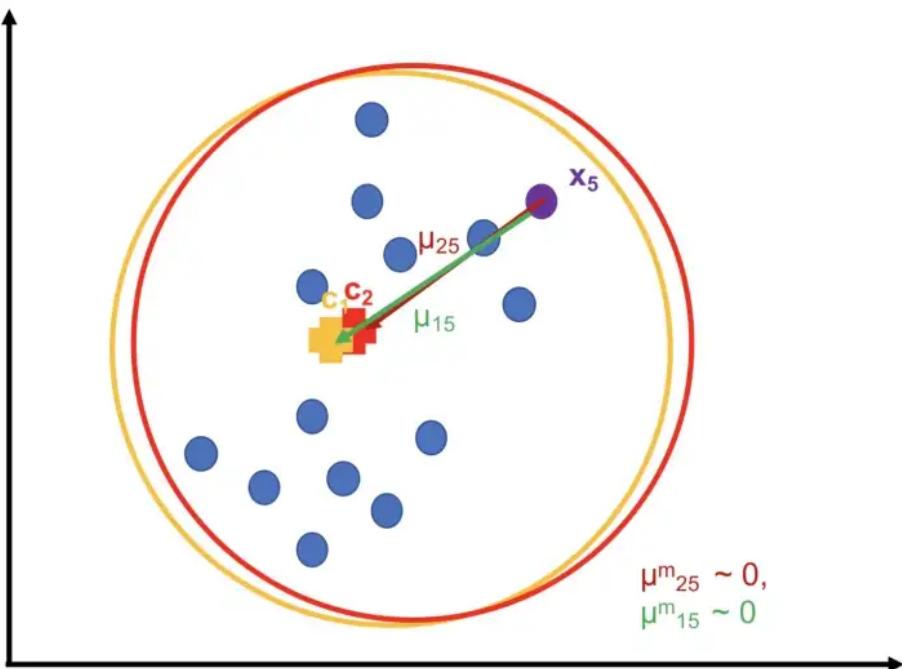
Vogliamo ancora una volta minimizzare la funzione obiettivo, quindi è bene avere probabilità basse per centroidi c_j significativamente distanti dalle istanze x_i .

Se p aumenta ad esempio $p=2,3\dots$, la differenza di contributo delle distanze diventa sempre più piccola,

Curva di decadimento del peso. la curva rossa è per μ_{25} e la curva blu è per μ_{15} .



Se ρ è troppo grande, con tutti i centroidi posizionati nel centro di tutti i punti, il clustering sarà molto "fuzzy", poiché non clusterizza affatto



Infine, per riuscire a minimizzare la funzione obiettivo SSE i centroidi e le distanze vengono calcolate come segue :

$$w_{ij} = \frac{1}{dist(x_i, c_j)^{2/(p-1)}} / \sum_{q=1}^k \frac{1}{dist(x_i, c_q)^{2/(p-1)}}$$

$$c_j = \sum_{i=1}^m w_{ij}^p x_i / \sum_{i=1}^m w_{ij}^p$$

L'equazione di c_j rappresenta la media pesata di X dal centroide del cluster j . Deve essere diviso tutto per la sommatoria delle distanze elevate alla p , perchè la somma dei pesi dev'essere uguale a 1 ma ciò si può ottenere naturalmente solo quando $p=1$, quindi il denominatore serve a normalizzare i pesi in una scala da 0 a 1 quando $p>1$.

SELF ORGANIZING MAP (SOM)

I SOM sono algoritmi di clustering basati sui centroidi, dove i centroidi hanno un'organizzazione spaziale (ad esempio, su un rettangolo 2D). Inoltre, i centroidi vicini sono simili. In questo senso, una SOM 2D è una proiezione del set di dati ad alta dimensionalità su una superficie 2D che cerca di preservare la somiglianza.

Si tratta di una tecnica di clustering e di visualizzazione.

ALGORITMO

Algorithm 8.3 Basic SOM Algorithm.

-
- 1: Initialize the centroids.
 - 2: **repeat**
 - 3: Select the next object.
 - 4: Determine the closest centroid to the object.
 - 5: Update this centroid and the centroids that are close, i.e., in a specified neighborhood.
 - 6: **until** The centroids don't change much or a threshold is exceeded.
 - 7: Assign each object to its closest centroid and return the centroids and clusters.
-

- L'inizializzazione consiste nel decidere la dimensione della mappa (2D, quadrata rettangolare) e la forma (le celle possono essere esagonali). Ogni cella della mappa è un centroide, tipicamente inizializzato come un punto dati casuale. Alla fine, un insieme di centroidi viene disposto su una mappa scalata a una dimensione inferiore.
- L'algoritmo aggiorna i centroidi in modo che la mappa possa essere considerata una vista ri-scalata del set di dati
- La fase più complessa è la fase 5: l'aggiornamento. L'obiettivo è quello di rendere ogni centroide più simile ai punti del dataset che gli sono vicini.
- Ogni iterazione t è un'epoca del processo di addestramento.
Dato il punto corrente $p(t)$, il centroide più vicino $m(t)$ (ovvero il vincitore), il centroide viene aggiornato come segue:

$$m_j(t + 1) = m_j(t) + h_j(t)(p(t) - m_j(t))$$

dove h_j è una funzione di prossimità del centroide che pondera l'aggiornamento. Si noti che in linea di principio tutti i centroidi possono essere aggiornati. Una tipica h è:

$$h_j(t) = \alpha(t) \text{ if } dist(m_j, m_k) \leq threshold, 0 \text{ otherwise}$$

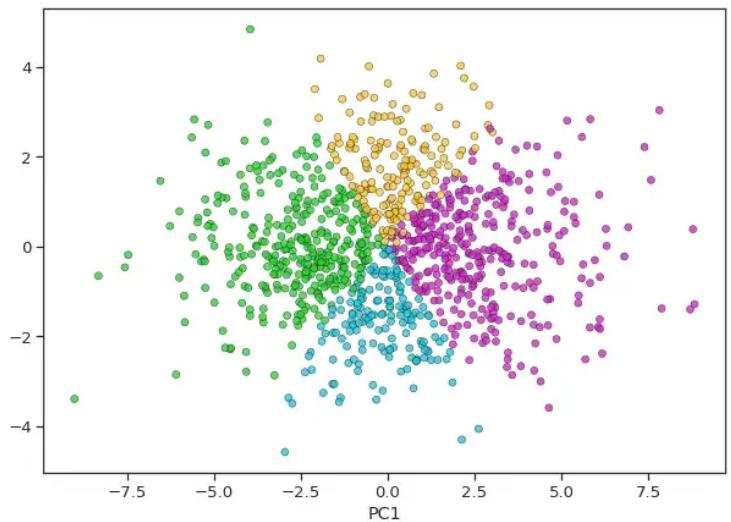
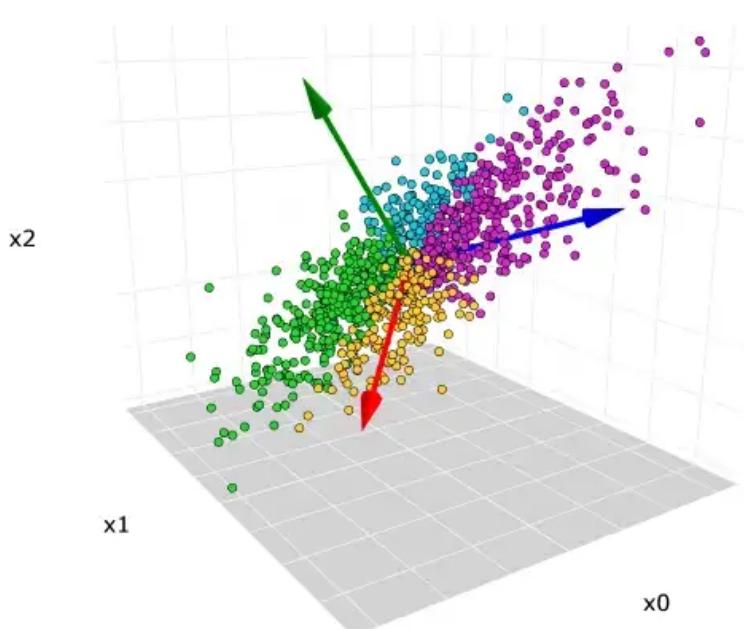
dove la distanza $dist(m_j, m_k)$ è misurata sulla griglia SOM e non nello spazio delle caratteristiche originale e $0 < \alpha(t) <$ diminuisce nel corso delle epoche.

14. DIMENSIONALITY REDUCTION

PRINCIPAL COMPONENT ANALYSIS

PCA è un tool indispensabile per la visualizzazione e la dimensionality reduction

PCA è una tecnica che trasforma "grandi dimensioni" in "piccole dimensioni" cercando di mantenere più informazione possibile.



PCA diventa molto utile quando si ha a che fare con dataset con un numero di features molto elevato. Trova utilizzo nell'ambito dell' image-processing o genome-research.

Mentre avere molti dati è sempre positivo, talvolta hanno talmente tanta informazione intrinseca che i tempi di addestramento del modello si allungano a dismisura e la curse of dimensionality comincia a diventare un problema, in questo caso si dice "meno è meglio".

COME FUNZIONA

E' un processo sviluppato in 2 fasi:

- Understand → capire
- Summarize → riassumere

Ci potremmo chiedere come PCA possa capire quale parte dei nostri dati sia importante, ed inoltre, possiamo quantificare matematicamente l'informazione contenuta nei dati?

Sì, attraverso la VARIANZA.

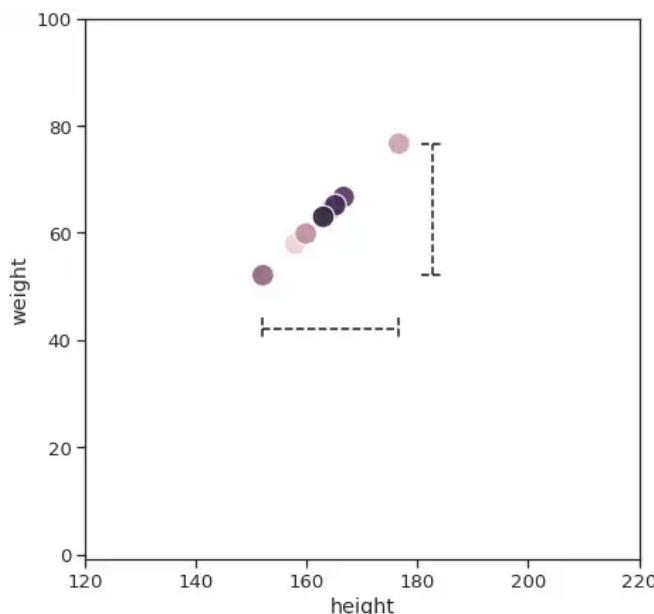
Questo perché quando i dati hanno varianza molto alta contengono maggiore informazione e questo rende più facile differenziare i dati tra loro.

PCA vede la varianza come INFORMAZIONE.

Intuitivamente, andremmo a considerare le feature che contengono maggiore informazione e quindi più varianza, l'idea è quindi quella di selezionare e considerare le variabili con alta varianza e dimenticarci di quelle con bassa varianza.

Ma cosa succede se le features contengono informazioni con varianza molto simile se non uguale?

Esempio.



| Feature | Variance |
|--------------|-------------|
| Height | 1.11 |
| Weight | 1.11 |
| TOTAL | 2.22 |

Nel caso riportato è molto difficile scegliere una variabile da eliminare, se eliminiamo una delle 2 stiamo buttando via metà dell'informazione.

Possiamo tenerle entrambe?

Probabilmente, utilizzando una prospettiva diversa.

Utilizzare una prospettiva diversa può risultare efficace poiché talvolta interpretando individualmente le singole feature non si riesce a contestualizzare l'informazione, mentre analizzandole in maniera congiunta è possibile far emergere dell'informazione intrinseca quindi riassumere l'informazione di più variabili in una sola.

Finora abbiamo guardato alle singole feature Height e Weight individualmente che sappiamo avere un pattern di correlazione. Proviamo a combinare assieme le due.

Proponiamo di combinare assieme peso e altezza in una nuova informazione chiamata BMI(body mass index) attraverso una combinazione lineare del tipo:

$$BMI = \alpha * Weight + \beta * Height$$

Dove α e β sono i pesi che vengono dati alle features.

Ipotizzando di fare la media delle 2, avremo :

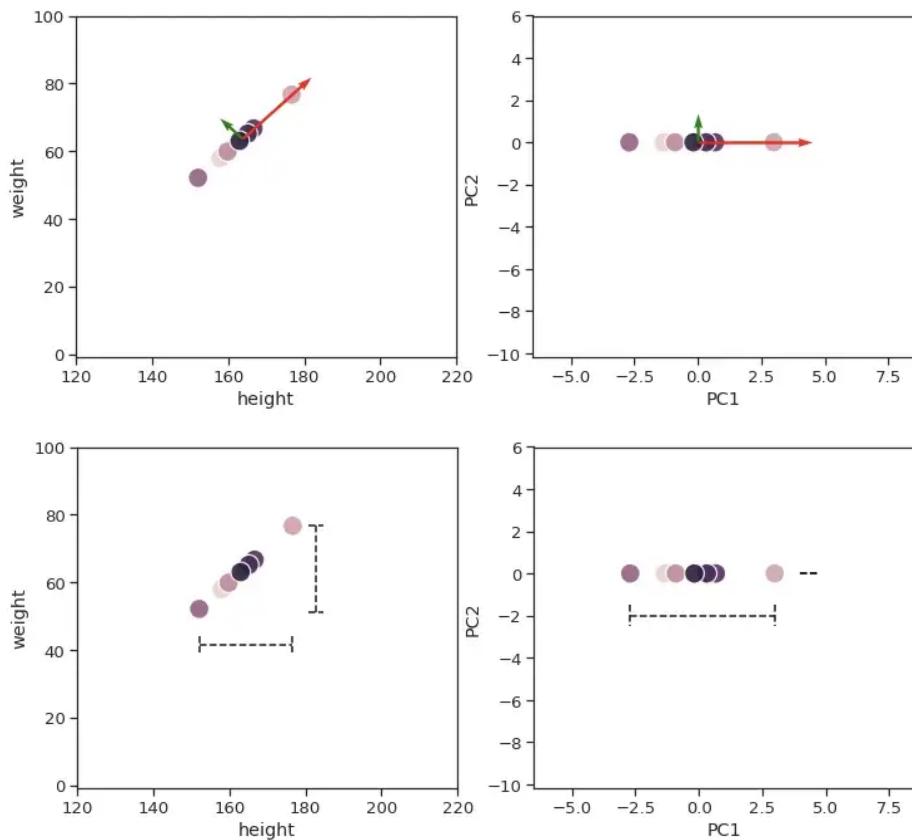
$$BMI = \frac{Weight + Height}{2}$$

Che analizzandola equivale alla combinazione lineare riportata sopra ma con pesi = $\frac{1}{2}$:

$$BMI = \frac{1}{2}Weight + \frac{1}{2}Height$$

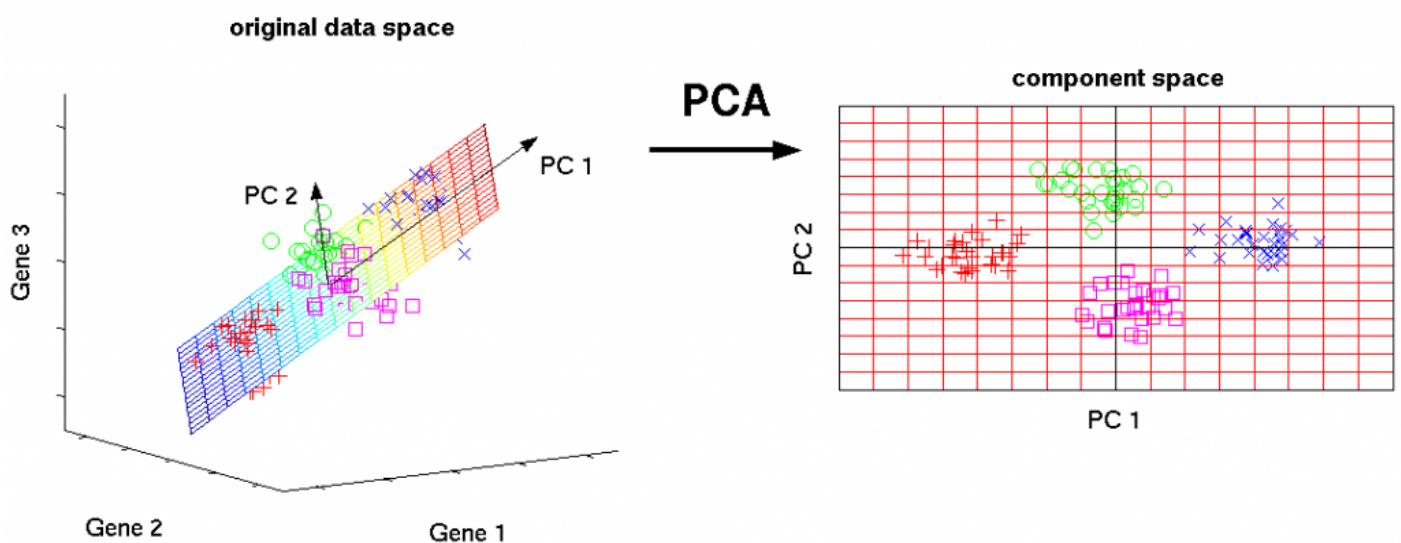
Modificando i pesi possiamo bilanciare l'informazione che stiamo riassumendo con la combinazione lineare riportata, sulla base delle varianze delle features, quindi determinando variabili trainanti.

La PCA1 sarà la componente principale che andrà a spiegare la maggior parte della varianza, nel caso sopra sarà la diagonale che minimizza SSE, e la PCA2 sarà quella che a scalare spiegherà la variabilità rimanente.



Poi la PCA1 sarà la nuova ascissa, mentre la PCA2 la nuova ordinata, nel caso sopra la PCA2 ha varianza 0 mentre la PC1 riesce a spiegare tutta la variabilità di height e weight

| Feature | Variance | Feature | Variance |
|--------------|-------------|--------------|-------------|
| Height | 1.11 | PC1 | 2.22 |
| Weight | 1.11 | PC2 | 0.00 |
| TOTAL | 2.22 | TOTAL | 2.22 |



15. ASSOCIATION ANALYSIS

L'association analysis è un tema caldo in campo Data Science. Riuscendo a scoprire relazioni tra item all'interno di grandi quantità o reti di dati, è possibile ricavare un sacco di informazioni in molte aree. Tra questi vi sono:

- la scoperta di modelli di acquisto inconsapevoli da parte dei consumatori, in negozio o app,
- ricerca di modelli nei dati relativi al text mining
- scoperta di modelli nei dati relativi ad assistenza sanitaria trasporti o sondaggi.

Supponiamo che :

- $I = \{i_1, i_2, \dots, i_n\}$ sia un set di tutti gli articoli in un carrello della spesa
- $T = \{t_1, t_2, \dots, t_n\}$ un set di tutte le transazioni.

Ogni transazione t_i contiene un sottoinsieme di articoli scelti da I .

Si dice che una transazione t_j contiene un itemset X se X è un sottoinsieme di t_j

Nell'association analysis una collezione di 0 o più elementi è detta item-set. Se un item-set contiene k elementi, prende il nome di k-itemset.

ES: {BIRRA, PANNOLINI, LATTE} è un 3-itemset.

Table 4.2. A binary 0/1 representation of market basket data.

| TID | Bread | Milk | Diapers | Beer | Eggs | Cola |
|-----|-------|------|---------|------|------|------|
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 1 | 1 | 0 |
| 3 | 0 | 1 | 1 | 1 | 0 | 1 |
| 4 | 1 | 1 | 1 | 1 | 0 | 0 |
| 5 | 1 | 1 | 1 | 0 | 0 | 1 |

TID: 2

Contiene {BREAD,DIAPERS} ,
ma NONcontiene {BREAD,MILK}

Una proprietà importante di un itemset è il suo SUPPORT COUNT, che fa riferimento al numero di transazioni che contengono un particolare itemset.

{BEER,DIAPERS,MILK} ha un support count = 2 poiché solo due transazioni contengono tutti e tre gli elementi.

Le relazioni possono essere rappresentate da delle regole di associazione, scritte in forma :

- $\{A\} \rightarrow \{B\}$, dove A e B presentano una qualche relazione.

La forza della relazione viene misurata in termini di SUPPORT e CONFIDENCE

SUPPORT

Questa misura dà un'idea di quanto frequente sia un itemset all'interno di tutte le transazioni.

Consideriamo :

- Itemset1 = {BREAD} e Itemset2 = {SHAMPOO}

A logica ci saranno maggiori transazioni contenenti BREAD di quelle contenenti SHAMPOO
Quindi Itemset1 avrà un SUPPORTO maggiore rispetto a itemset2.

- Itemset3= {BREAD,BUTTER} e itemset4 = {BREAD,SHAMPOO}

A logica ci saranno più transazioni di BREAD e BUTTER assieme che BREAD e SHAMPOO.
Quindi l'itemset3 avrà un supporto più alto di itemset4.

Matematicamente, il supporto è il rapporto tra il numero totale di transazioni e il numero di transazioni in cui A e B compaiono.

$$SUPPORT(\{A\} \rightarrow \{B\}) = \frac{\text{TRANSACTION containing both } A \text{ and } B}{\text{TOTAL NUMBER OF TRANSACTION}}$$

Il valore del supporto aiuta a identificare regole che vale la pena considerare per un'ulteriore analisi.

CONFIDENCE

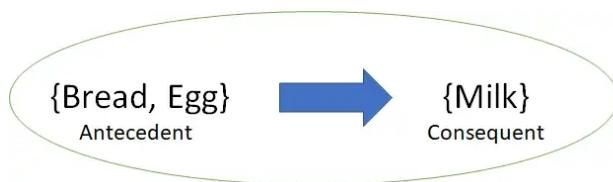
Questa misura definisce la probabilità, considerando la relazione A → B che si verifichi B dato che nel carrello sono già presenti tutti gli A. Si tratta di rispondere alla domanda - "di tutte le transazioni che contengono {OREO} quante di queste contengono anche {MILK}?"

Per conoscenza possiamo dire che {OREO} → {MILK} dovrebbe avere CONFIDENCE alta. Tecnicamente quindi la confidence non è altro che la probabilità condizionata di B dato A.

$$CONFIDENCE(\{A\} \rightarrow \{B\}) = \frac{\text{TRANSACTION containing both } A \text{ and } B}{\text{TRANSACTION containing } A}$$

Consideriamo degli esempi prima di proseguire :

- {BUTTER} → {BREAD} : molte transazioni che contengono BUTTER hanno anche BREAD
- {YOGURT} → {MILK} : molte transazioni anche in questo caso
- {TOOTHBRUSH} → {MILK} : non ne siamo sicuri, MA anch'essa sarà alta poiché MILK è un itemset molto frequente e potrebbe essere presente in tutte le altre transazioni.



Itemset = {Bread, Egg, Milk}

N.B : Non importa cosa c'è nel ANTECEDENTE per un CONSEGUENTE molto frequente. La confidence per una associazione rule con un conseguente molto frequente sarà sempre alta.

Considerare solo il valore di confidence limita la nostra capacità di fare dell'inferenza di business

RULE MINING

La rule generation è un processo che si sviluppa in 2 fasi:

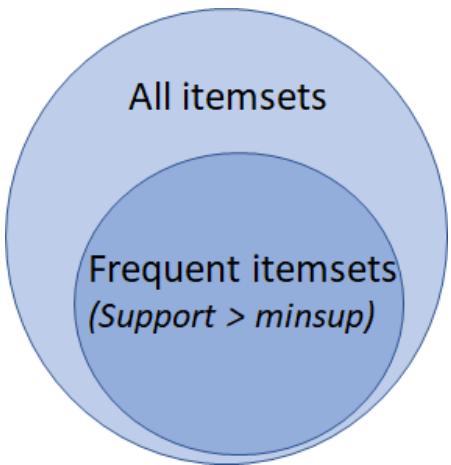
- 1) Generazione di un itemset
- 2) Generazione di regole da ogni itemset

Generazione di itemset da una lista di item

Il primo step mira ad ottenere tutti gli itemset più frequenti sui quali è possibile eseguire partizioni binarie per ottenere l'antecedente e il conseguente.

Esempio : {Bread, Butter, Egg, Milk, Notebook, Toothbrush} combinando tutte le transazioni otterremo itemset del tipo {Bread}, {Butter}, {Bread, Notebook}, {Milk, Toothbrush}, {Milk, Egg, Vegetables} etc.

La dimensione di un itemset varia da 1 al totale degli item che abbiamo. Ora cerchiamo solo quelli FREQUENTI da questi, in modo da controllare il numero di itemset generati.



FREQUENT ITEMSET : sono gli itemset che occorrono almeno un minimo numero di volte nelle transazioni. Ovvero gli itemset il cui valore di supporto è $>$ di una soglia minsup

Quindi, {BREAD, NOTEBOOK} potrebbe non essere un insieme di elementi frequenti se si verifica solo 2 volte su 100 transazioni e $(2/100) = 0,02$ è inferiore al valore di minsup .

E' possibile effettuare un approccio bruteforce per formare tutti gli itemsets e controllare per ognuno il valore di supporto. PRINCIPIO di APRIORI aiuta nel ricercarli in maniera più efficiente.

APRIORI PRINCIPLE

Il principio di Apriori afferma che : tutti i sottoinsiemi di un itemset frequente devono anch'essi essere frequenti.

Questo equivale a dire che il numero di transazioni che contengono items {BREAD, EGG} è maggiore o uguale al numero di transazioni che contengono {BREAD, EGG, VEGETABLES}. Se quest'ultima compare in 30 transazioni, la prima apparirà in tutte e 30 e possibilmente anche in altre.

Il support value di {BREAD, EGG, VEGETABLES} è $\frac{30}{100} = 0,3$ maggiore di minsup , quindi possiamo assumere che il supporto di {BREAD, EGG} è $\geq 0,3$.

Questa proprietà è chiamata PROPRIETA' ANTI-MONOTONA DEL SUPPORTO:

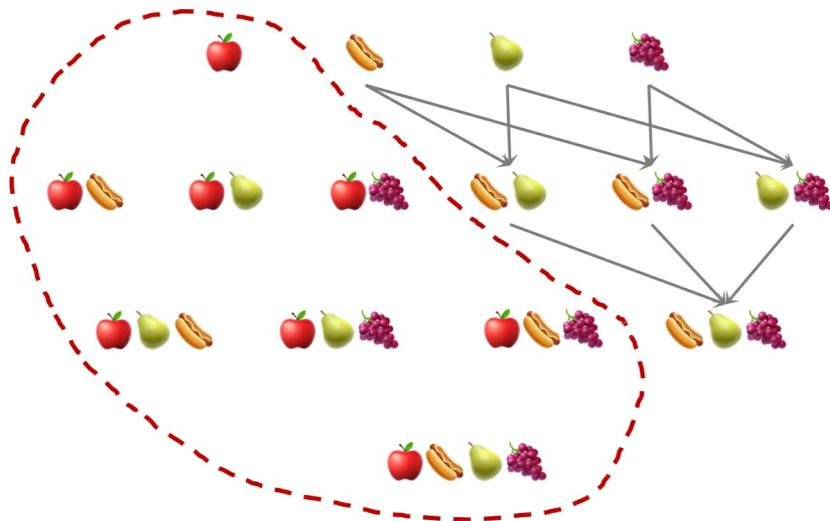
- Se si elimina un item da un itemset, il valore del supporto del nuovo itemset generato sarà uguale o maggiore.

Il principio di Apriori ci permette quindi di eliminare tutti i superset che contengono elementi che non superano il minsup threshold.

APRIORI ALGORITHM

- 1) Generare tutti gli itemsets frequenti (con support $>$ minsup) di lunghezza 1
- 2) Generare tutti gli itemsets di lunghezza 2 combinando tutti gli itemset di lunghezza 1.
- 3) Scartare (Pruning) tutti quelli il cui supporto è $<$ minsup
- 4) Generare tutti gli itemsets di lunghezza 3 tramite tutte le combinazioni di itemsets di lunghezza 2 (quelli rimasti dal pruning) e controllare il loro supporto.
- 5) Ripetere la generazione di itemsets di lunghezza N utilizzando le combinazioni di itemsets non scartate di lunghezza N-1, fino ad esaurimento itemsets.

Come si può vedere dal grafico, il pruning degli insiemi poco frequenti riduce il numero di insiemi da considerare di oltre la metà.



Generazione di tutte le regole per itemsets frequenti

Una volta che gli itemsets sono stati generati, identificare le regole è meno dispendioso. Le regole sono formate da una partizione binaria per ogni itemset.

Se $\{\text{Bread}, \text{Egg}, \text{Milk}, \text{Butter}\}$ è un itemset Frequent, le regole candidate saranno :

- $(\text{Egg}, \text{Milk}, \text{Butter} \rightarrow \text{Bread})$,
- $(\text{Bread}, \text{Milk}, \text{Butter} \rightarrow \text{Egg})$,
- $(\text{Bread}, \text{Egg} \rightarrow \text{Milk}, \text{Butter})$,
- $(\text{Egg}, \text{Milk} \rightarrow \text{Bread}, \text{Butter})$,
- $(\text{Butter} \rightarrow \text{Bread}, \text{Egg}, \text{Milk})$

Da una lista di possibili candidati cerchiamo di identificare regole che hanno un livello di confidenza $>$ di un minimo minconf.

Come la proprietà anti-monotona del supporto, il supporto delle regole generate dallo stesso itemsets segue la medesima proprietà. È anti-monotona rispetto al numero di elementi nel conseguente (le regole con minori elementi nel conseguente avranno livello di support più alto)

Questo vuol dire che la confidenza di $(A, B, C \rightarrow D) \geq (B, C \rightarrow A, D) \geq (C \rightarrow A, B, D)$.

Come remind, la confidenza di $\{X \rightarrow Y\} = \text{support count of } \{X, Y\} / \text{support count of } \{X\}$.

Come sappiamo, il supporto di tutte le regole generate dallo stesso itemset rimane lo stesso, la differenza invece si rileva solo nel denominatore per il calcolo della confidenza.

Quando il numero di items nell'itemset X diminuisce,

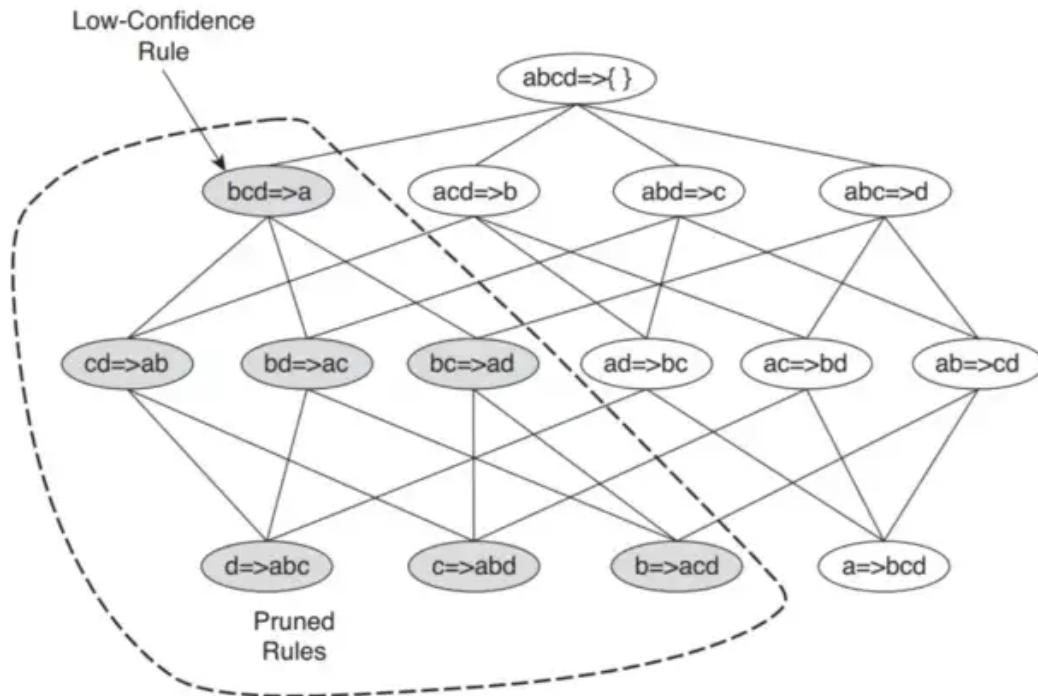
- il supporto di X aumenta ($\text{support} = \text{transaction with ALL item} / \text{ALL possible transaction}$)
- la confidenza diminuisce
 $\text{confidence} = \text{transaction with ALL antecedent e consequent} / \text{transaction with antecedent}$)

ESEMPIO:

Se $\{\text{ACD}\} \rightarrow \{\text{B}\}$ e $\{\text{ABD}\} \rightarrow \{\text{C}\}$ hanno confidenza alta, allora poi la regola candidata $\{\text{ad}\} \rightarrow \{\text{bc}\}$ sarà generata unendo i conseguenti di entrambe le regole.

Se un qualunque nodo si rivela avere confidenza bassa allora tutto il suo sottografo può essere rimosso immediatamente.

Supponiamo che la confidenza di $\{BCD\} \rightarrow \{A\}$ sia bassa. Allora tutte le regole che contengono A nel conseguente possono essere scartate



Algorithm 4.2 Rule generation of the *Apriori* algorithm.

```

1: for each frequent  $k$ -itemset  $f_k$ ,  $k \geq 2$  do
2:    $H_1 = \{i \mid i \in f_k\}$       {1-item consequents of the rule.}
3:   call ap-genrules( $f_k, H_1$ .)
4: end for

```

Algorithm 4.3 Procedure ap-genrules(f_k, H_m).

```

1:  $k = |f_k|$     {size of frequent itemset.}
2:  $m = |H_m|$     {size of rule consequent.}
3: if  $k > m + 1$  then
4:    $H_{m+1} = \text{candidate-gen}(H_m)$ .
5:    $H_{m+1} = \text{candidate-prune}(H_{m+1}, H_m)$ .
6:   for each  $h_{m+1} \in H_{m+1}$  do
7:      $conf = \sigma(f_k)/\sigma(f_k - h_{m+1})$ .
8:     if  $conf \geq minconf$  then
9:       output the rule  $(f_k - h_{m+1}) \longrightarrow h_{m+1}$ .
10:    else
11:      delete  $h_{m+1}$  from  $H_{m+1}$ .
12:    end if
13:   end for
14:   call ap-genrules( $f_k, H_{m+1}$ .)
15: end if

```

ESERCIZIO: trovare gli itemsets frequenti e generare le loro regole di associazione.
Assumiamo che il minsup = 33.33% e minconf = 60%

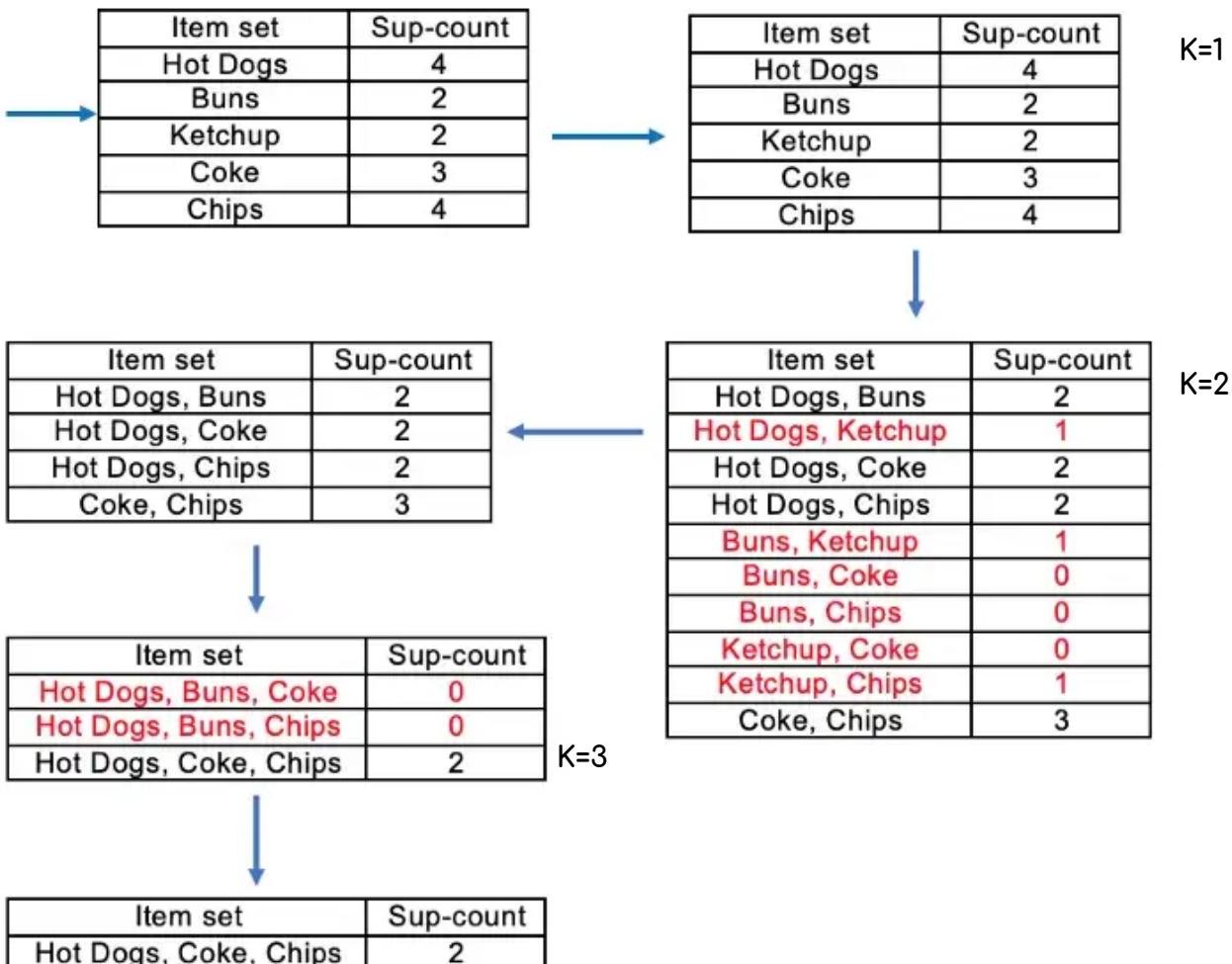
| Transaction ID | Items |
|----------------|-------------------------|
| T1 | Hot Dogs, Buns, Ketchup |
| T2 | Hot Dogs, Buns |
| T3 | Hot Dogs, Coke, Chips |
| T4 | Chips, Coke |
| T5 | Chips, Ketchup |
| T6 | Hot Dogs, Coke, Chips |

Come prima cosa calcoliamo il minimum support count nel seguente modo:

$$\text{minimum SUPPORT COUNT} = \frac{33.33}{100} * 6 \leftarrow \text{numero di transazioni}$$

$$\text{minimum SUPPORT COUNT} = 2$$

Procediamo poi a generare gli itemsets frequenti:



Notiamo che l'unico itemset frequente è :

- *Frequent Itemset (I) = {Hot Dogs, Coke, Chips}*

Ora calcoliamo le regole di associazione:

Frequent Itemset (I) = {Hot Dogs, Coke, Chips}

Numero di consequenti = 1

- [Hot Dogs[^]Coke] → [Chips]
//confidence = sup(Hot Dogs[^]Coke[^]Chips)/sup(Hot Dogs[^]Coke) = 2/2*100=100%
//SELECTED
- [Hot Dogs[^]Chips] → [Coke]
//confidence = sup(Hot Dogs[^]Coke[^]Chips)/sup(Hot Dogs[^]Chips) = 2/2*100=100%
//SELECTED

Numero di consequenti = 2

- [Coke[^]Chips] → [Hot Dogs]
//confidence = sup(Hot Dogs[^]Coke[^]Chips)/sup(Coke[^]Chips) = 2/3*100=66.67%
//SELECTED
- [Hot Dogs]=>[Coke[^]Chips]
//confidence = sup(Hot Dogs[^]Coke[^]Chips)/sup(Hot Dogs) = 2/4*100=50%
//REJECTED

Numero di consequenti = 3

- [Coke]=>[Hot Dogs[^]Chips]
//confidence = sup(Hot Dogs[^]Coke[^]Chips)/sup(Coke) = 2/3*100=66.67%
//SELECTED
- [Chips]=>[Hot Dogs[^]Coke]
//confidence = sup(Hot Dogs[^]Coke[^]Chips)/sup(Chips) = 2/4*100=50%
//REJECTED

Ci sono 4 regole forti come risultato (con una confidenza minima maggiore del 60%)

SUPPORT COUNTING

Il support counting è il processo di conteggio della frequenza di una occorrenza per ogni itemset candidato che sopravvive alla fase di pruning.

L'approccio brute force è quello di confrontare ogni transazione con ogni itemset candidato e aggiornare i counter dei candidati contenuti nella transazione.
Questo approccio è computazionalmente costoso, soprattutto quando il numero di transazioni e di insiemi di elementi candidati è elevato.

ENUMERATING

Un approccio alternativo consiste nell'enumerare gli insiemi contenuti in ogni transazione e utilizzarli per aggiornare i support counter dei rispettivi insiemi candidati.

ESEMPIO

si consideri una transazione t che contiene 5 elementi, $\{1, 2, 3, 5, 6\}$.

In questa transazione sono contenuti $\binom{5}{3} = 10$ itemsets di dimensione 3.

Alcuni degli insiemi possono corrispondere a insiemi di 3-itemset candidati.

Alcuni degli insiemi possono corrispondere agli 3-itemset candidati sotto osservazione, in qual caso il loro support count viene incrementato.

Altri sottoinsiemi di t che non corrispondono a nessun candidato possono essere ignorati.

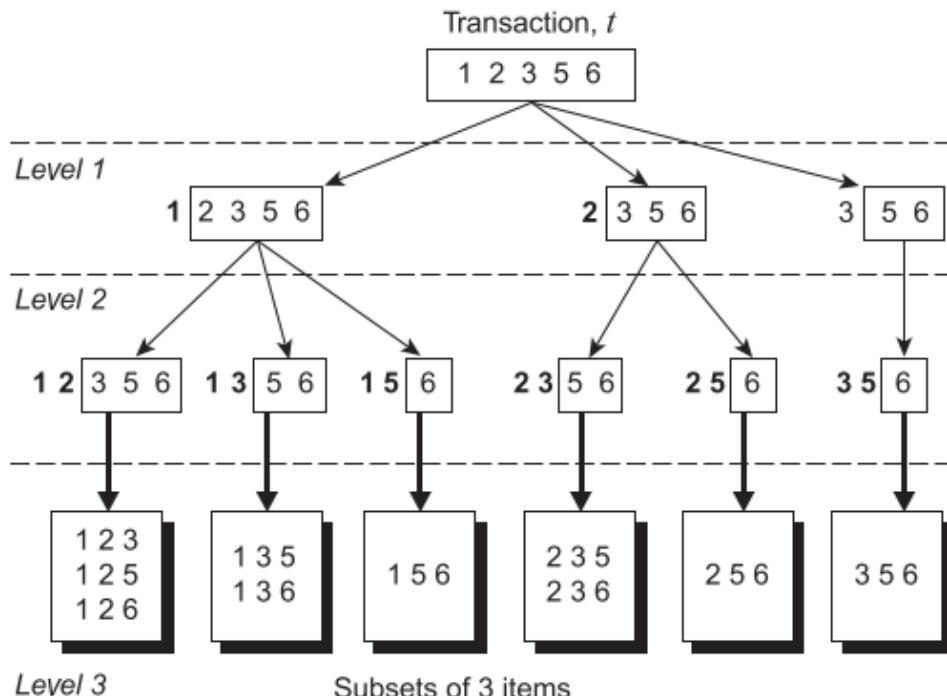


Figure 4.9. Enumerating subsets of three items from a transaction t .

Vediamo un metodo schematico per enumerare i 3-itemsets contenuti in t .

Assumiamo che ogni itemset contenga i suoi item in ordine lessicografico, quindi un itemset può essere enumerato specificando prima il suo item più piccolo seguito poi dal più grande.

Per $t=\{1,2,3,5,6\}$ tutti i 3-itemsets contenuti in t dovranno necessariamente iniziare con 1-2-3. Non sarà possibile costruire 3-itemsets che iniziano per 5-6 perché l'unico elemento più grande è 6 o nessuno.

Il numero di modi in cui è possibile specificare il PRIMO elemento è il LIVELLO 1 dell'albero.
ES.

[1][2-3-5-6] rappresenta un itemset che inizia con 1 seguito da altri 2 item in ordine lessicografico.

Il secondo possibile elemento è specificato dal LIVELLO 2 dell'albero.

ES.

[1-2][3-5-6] corrisponde a un itemset che inizia con 1-2 seguito da uno degli altri 3 item rimasti.

Infine tutti gli elementi al LIVELLO 3 rappresentano i 3-itemsets relativi all'albero dei prefissi del livello 2, contenuti in t .

ES.

Prefix = [1-2] $\Rightarrow [1,2][3] - [1-2][5] - [1-2][6]$

Prefix = [1-3] $\Rightarrow [1-3][5] - [1-3][6]$

Prefix = [1-5] $\Rightarrow [1-5][6]$

Prefix = [2-3] $\Rightarrow [2-3][5] - [2-3][6]$

Prefix = [2-5] $\Rightarrow [2-5][6]$

Prefix = [3-5] $\Rightarrow [3-5][6]$

HASH TREE

Nell'algoritmo Apriori, gli insiemi di elementi candidati vengono suddivisi in differenti insiemi e memorizzati in un albero di hash.

Durante il support counting, anche gli insiemi di elementi contenuti in ogni transazione vengono sottoposti a hashing nei rispettivi insiemi. In questo modo, invece di confrontare ogni insieme di elementi della transazione con ogni insieme candidato, viene confrontato solo con gli insiemi di elementi candidati che appartengono allo stesso insieme.

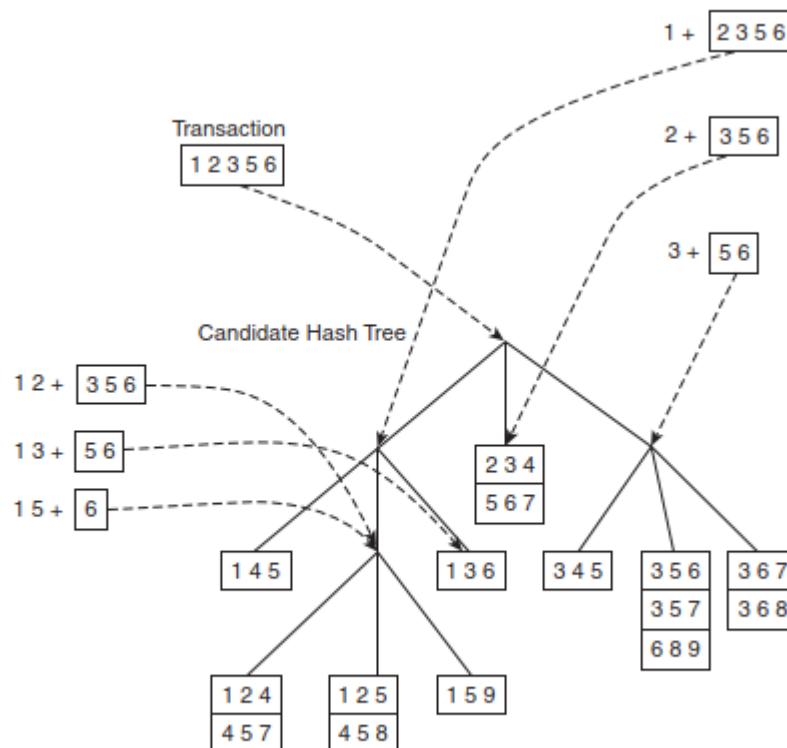


Figure 4.12. Subset operation on the leftmost subtree of the root of a candidate hash tree.

La complessità computazionale dell'algoritmo APRIORI può essere influenzata dai seguenti fattori:

- support threshold
- Numero di items(dimensionalità)
- Numero di transazioni
- Dimensione media delle transazioni
- Generazione degli 1-itemsets
- Generazione dei candidati
- Support counting

FP-GROWTH ALGORITHM

Un approccio algoritmico radicalmente differente per il calcolo degli itemsets frequenti è FP-growth algorithm. Questo algoritmo non segue il paradigma di genera e testa di Apriori. Invece fa encoding delle informazioni usando una struttura chiamata FP-TREE ed estrae gli itemsets frequenti da questa struttura.

FP-TREE

Un fp-tree è una rappresentazione compressa dei dati di input, è costruito leggendo i set di dati una transizione alla volta e mappando ogni transazione in un cammino del FP-TREE. Come transazioni diverse possono avere item in comune, anche i cammini possono sovrapporsi. Più cammini si sovrappongono e più compresso sarà l'FP-TREE. Se la dimensione è abbastanza contenuta da poter stare in memoria centrale, questo permette di estrarre gli itemsets frequenti direttamente dalla struttura in memoria invece di effettuare operazioni ripetute sui dati salvati nel disco.

ESEMPIO

Ogni nodo dell'albero ha un'etichetta con un counter di quante transazioni sono mappate su quel cammino in quel nodo. Inizialmente l'albero contiene solo un nodo null, e poi si estende così:

- Scansione del dataset una prima volta per determinare il support count di ogni item. Item non frequenti sono scartati, ognuno dei rimanenti è ordinato in ordine decrescente in base al valore del supporto all'interno di ogni transazione.
ES. supp_count : a>b>c>e | t : {c,a,b} diventa → {a,b,c}.
- Dopo aver letto la prima transazione {a,b}, i nodi etichettati come a e b vengono creati. La transazione è rappresentata dal cammino $\text{NULL} \rightarrow a(1) \rightarrow b(1)$, ogni nodo nel cammino avrà frequenza = 1.
- Dopo la seconda transazione {b,c,d} nuovi nodi verranno creati per b c d. Un nuovo cammino creato $\text{NULL} \rightarrow b(1) \rightarrow c(1) \rightarrow d(1)$, e frequenze = 1.
Anche se le due transazioni hanno b in comune, i loro cammini sono diversi perché le transazioni non hanno un prefisso in comune.
- La terza transazione {a,c,d,e} condivide il prefisso a con {a,b,c}. Il cammino diverrà $\text{NULL} \rightarrow a(2) \rightarrow c(1) \rightarrow d(1) \rightarrow e(1)$.
- Il processo continua fino a che ogni transazione non viene mappato, inoltre FP-tree contiene una lista di puntatori che collegano nodi degli stessi oggetti (linee tratteggiate) per facilitare e velocizzare l'accesso agli item nell'albero.

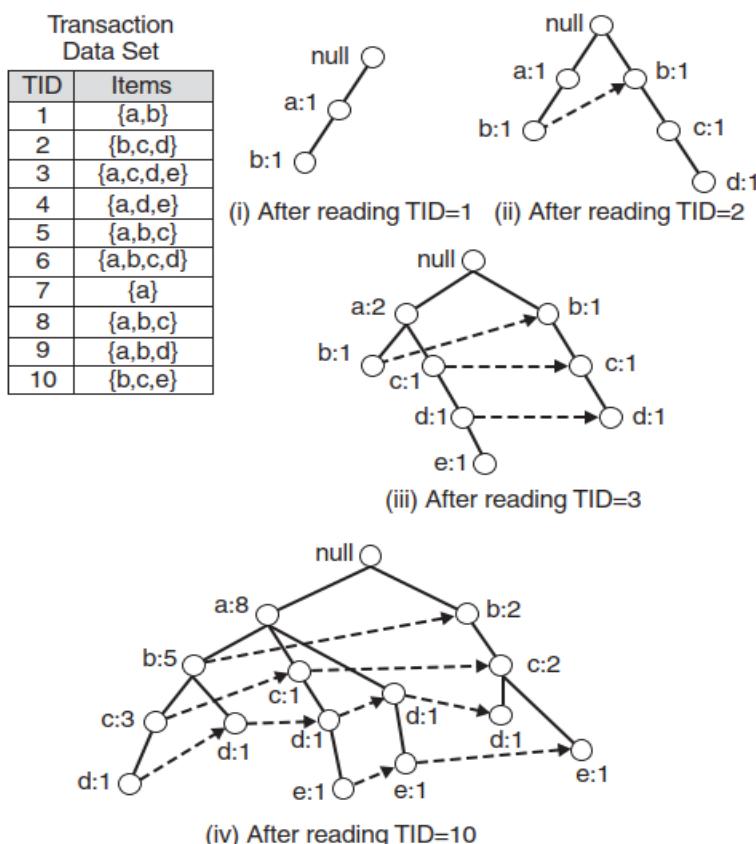


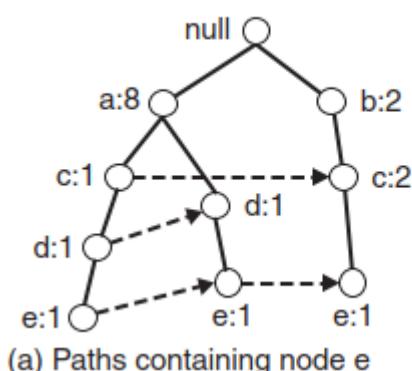
Figure 4.24. Construction of an FP-tree.

FREQUENT ITEMSETS GENERATION

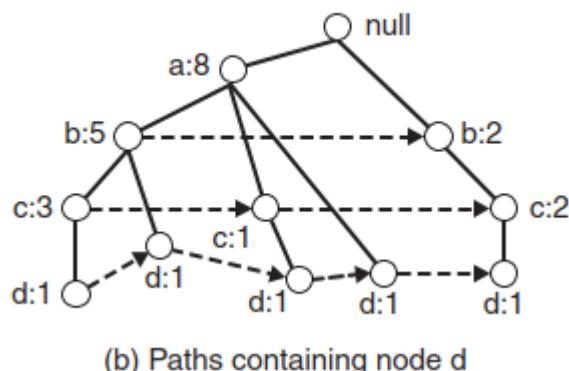
La generazione degli itemsets frequenti viene effettuata esplorando l'albero bottom-up partendo dagli item più frequenti. Siccome ogni transazione è mappata in un cammino dell'albero, per derivare gli itemsets frequenti è sufficiente esplorare solo i cammini che contengono quegli item. E questi cammini possono essere acceduti velocemente tramite i puntatori ai loro nodi.

FP-Growth trova tutti gli itemsets che terminano con un particolare suffisso utilizzando una strategia di divide-and-conquer per dividere il problema in sottoproblemi.

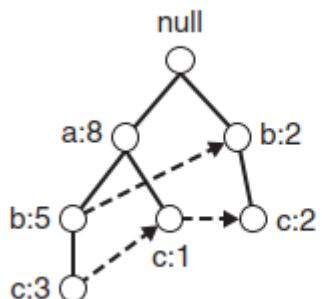
Ad esempio, supponiamo di essere interessati a itemset che terminano con item E. Per farlo dobbiamo prima controllare che item E sia frequente. Per farlo consideriamo il sottoproblema di trovare gli itemsets frequenti che terminano in DE, poi CE e AE. A turno ognuno dei sottoproblemi è ulteriormente scomposto, e unendo le soluzioni dei sottoproblemi, tutti gli itemsets frequenti che terminano in E vengono trovati. Infine il set di itemsets viene generato unendo tutte le soluzioni dei sottoproblemi di trovare itemsets frequenti che terminano in E,D,C,B,A.



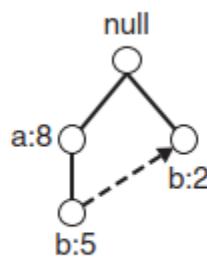
(a) Paths containing node e



(b) Paths containing node d



(c) Paths containing node c



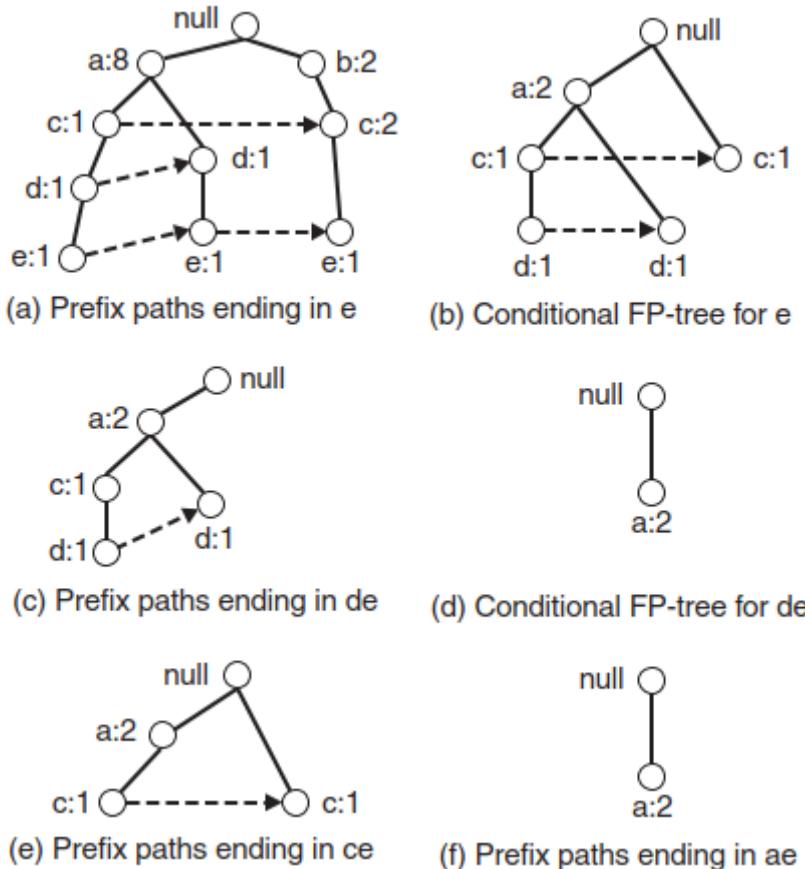
(d) Paths containing node b



(e) Paths containing node a

ESEMPIO : trovare itemsets frequenti che terminano con E, minsupp=2

- 1) Ottenere tutti i path che contengono E, questi path sono chiamati PREFIX PATH
- 2) Calcolare il support count di E, questo viene fatto sommando le frequenze dei nodi E
- 3) S.C(E) = 3 quindi è frequente, ora bisogna risolvere il sottoproblema degli itemsets che terminano in DE - CE - BE - AE, prima bisogna convertire i PREFIX PATH in dei CONDITIONAL FP-TREE, simili a FP-TREE ma permettono di trovare itemsets che finiscono con un determinato suffisso.



Un CONDITIONAL FP TREE si costruisce come segue:

-
- 4) FP-Growth usa Conditional tree per risolvere i sottoproblemi in DE,CE,BE,AE. Per trovare gli itemsets che finiscono in DE, il prefix path per D è ingrandito dal conditional tree per E (immagine c). Aggiungendo il frequency count associato a D, possiamo ottenere il support count per {D,E}.
Finché il support count è = 2, {D,E} è considerato frequente.
Dopodichè l'algoritmo costruisce un FP-TREE per DE usando lo step 3. Dopo aver aggiornato il support count e rimosso C, il conditional tree per DE è (immagine d). Finché il conditional tree contiene solo A, il cui supporto è = a minsupp, l'algoritmo estrae l'itemset {A,D,E} e passa a prossimo sottoproblema, che è quello di generare gli itemsets frequenti che finiscono in CE.
Dopo aver processato i prefix path per c, {C,E} viene trovato. Tuttavia il conditional tree per CE non ha item frequenti e quindi vengono eliminate.
L'algoritmo procede a risolvere il prossimo sottoproblema e trova {A,E} come ultimo itemset frequente.

TODO: COMPLETARE CON ESEMPI

16. RECOMMENDER SYSTEM

In linea generale definiamo come recommender system gli algoritmi che suggeriscono prodotti rilevanti agli utenti (i prodotti possono essere dal film alla canzone ad un libro, o articoli vari).

Abbiamo varie misure per definire la qualità di un recommender system :

- Efficienza nella costruzione del modello
- Efficienza nel generare suggerimenti
- SERENDIPITY → la capacità di suggerire prodotti inaspettati all'utente ma che comunque potrebbe apprezzare
- COLD-START → suggerire contenuti quando non si ha conoscenza dei gusti dell'utente o non si sa nulla dell'utente stesso (problema di tutti i recommender system), migliorabile facendo specificare all'utente dei gusti iniziali (ad es. Generi di film preferiti, serie preferite, genere musicali, ecc.) oppure delle loro caratteristiche (sesso, età, ...).

CONTENT BASED RECOMMENDER SYSTEM

Si basano esclusivamente sulle attività svolte dall'utente e i suoi gusti, solitamente utilizzano delle informazioni aggiuntive riguardo all'utente, come l'età, il sesso, il lavoro. Considerando un sistema di suggerimento di film tengono conto anche dei generi, degli attori, trama o altre caratteristiche.

Nei sistemi Content based l'utente è definito come l'insieme dei film che guarda. Cercheranno di suggerire quindi i film più simili al profilo di un utente.

ESEMPIO

Supponiamo di voler modellare il fatto che le ragazze tendono a dare un voto alto a determinati film, invece i ragazzi ad altri film e così via. Se riusciamo ad ottenere un modello simile, allora fare una predizione per l'utente sarà abbastanza semplice una volta che abbiamo guardato alle caratteristiche dell'utente stesso (sesso, età, ...), determinando quali siano i film più rilevanti da suggerire.

VARIABILI E TERMINOLOGIE UTILIZZATE :

- $U \rightarrow$ utenti
- $I \rightarrow$ item da raccomandare
- Rating → feedback dell'utente : stelline, like, minuti guardati

Modelliamo un sistema utilizzando uno spazio vettoriale sulla base della trama di un film

Ogni $x \in I$ è un vettore di dimensione N. Dove ipotizziamo:

$N \rightarrow \#$ termini nel lessico

E $x[t] \rightarrow tf(t,x)$, pesato con $\rightarrow idf(t)$

- $tf(t,x)$: è la frequenza del termine t per l'item x
- $idf(t)$: è la inverse document frequency di t in I

| | t_1 | t_2 | t_3 | t_4 | ... | t_n |
|-----|-----|-----|-----|-----|-----|-----|
| x_1 | | | | | | |
| ... | | | | | | |
| x_n | | | | | | |

INVERSE DOCUMENT FREQUENCY

Definiamo $\text{idf}(t)$ come uno score ottenuto dalla INVERSE DOCUMENT FREQUENCY, ovvero una tecnica che assegna uno score di importanza ad ogni termine t in un documento tra una raccolta di documenti. Questa tecnica produce un vettore che rappresenta l'importanza del termine t in ogni documento x .

PROFILE USER

Infine il PROFILO UTENTE viene definito come la media degli items I_u con cui lui ha interagito :

$$u = \frac{1}{|I_u|} \sum_{x \in I_u} x$$

Dove:

- $I_u \rightarrow$ set di item con cui l'utente ha interagito positivamente
- $u \rightarrow$ sarà la canzone ideale per quel utente

Otteniamo quindi un vettore di N elementi t , dove ogni t_i è la media dei t_i degli I_u con cui l'utente ha maggiormente interagito

U:

| t_1 | t_2 | t_3 | t_4 | ... | t_n |
|----------------------------|----------------------------|----------------------------|----------------------------|-----|----------------------------|
| $\text{mean}(t_1 \in I_u)$ | $\text{mean}(t_2 \in I_u)$ | $\text{mean}(t_3 \in I_u)$ | $\text{mean}(t_4 \in I_u)$ | ... | $\text{mean}(t_n \in I_u)$ |

COSINE SIMILARITY

Per ottenere i K elementi più simili ad u , si utilizza la COSINE SIMILARITY, una misura di similarità tra un utente u ed un elemento x

$$\cos(u, x) = \frac{u \cdot x}{\|u\| \|x\|} = \frac{\sum_t u[t] * x[t]}{\sqrt{\sum_t u[t]^2} * \sqrt{\sum_t x[t]^2}}$$

Il recommender system ritorna i migliori K item simili al profilo utente u .

RIASSUMENDO

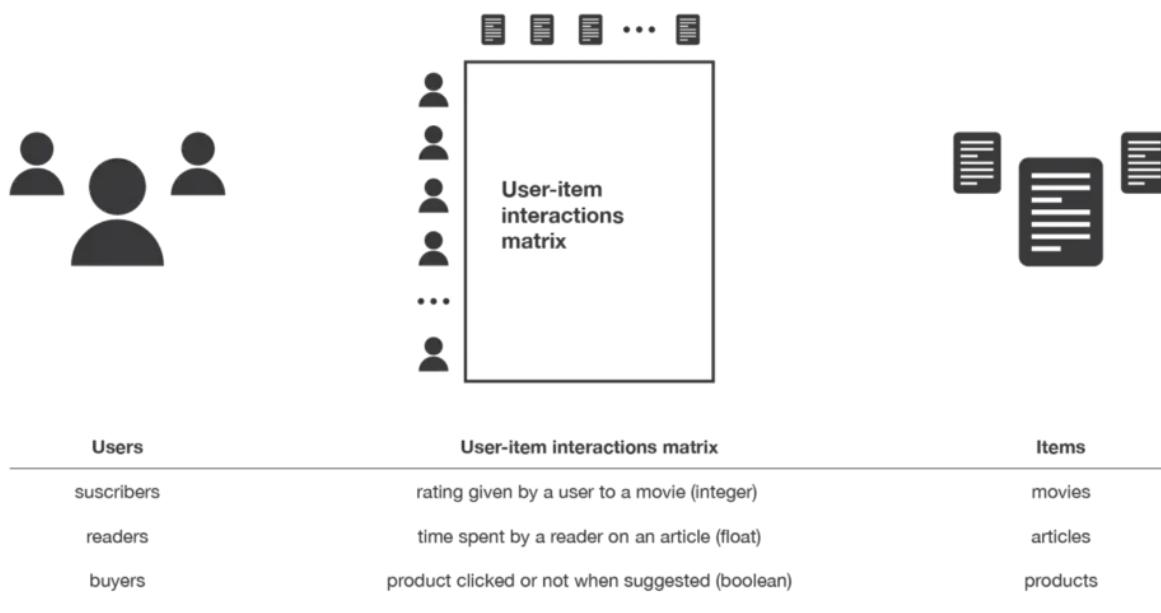
- Efficienza nella costruzione del modello : modello semplice da creare
- Efficienza nel generare suggerimenti : risultati da calcolare su tutta la collezione
- SERENDIPITY \rightarrow bassa, poiché i suggerimenti rimarranno sempre correlati a ciò con cui l'utente stesso interagisce e non saranno mai dei suggerimenti potenzialmente slegate dalle informazioni che abbiamo del profilo utente.
- COLD-START \rightarrow parziale, l'utente deve aver espresso almeno un parere, preferenza di qualche tipo

USER BASED COLLABORATIVE FILTERING METHODS

I collaborative methods per i recommender system sono metodi basati esclusivamente sulle interazioni passate registrate tra utenti e item, per produrre nuovi suggerimenti. Queste interazioni vengono memorizzate nella “user-item interaction matrix”. L’idea è quella che le interazioni passate di altri utenti sono sufficienti per scovare utenti simili e/o items simili e fare buoni suggerimenti.

VARIABILI E TERMINOLOGIE UTILIZZATE

- $U \rightarrow$ utenti
- $I \rightarrow$ item da raccomandare
- Rating \rightarrow feedback dell’utente : stelline, like, minuti guardati $\rightarrow R[u,i]$
- Rating\interaction Matrix : $|U| \times |I|$ dove ogni entry corrisponde al rating dell’utente u al film i . La matrice sarà molto vuota perché l’utente non può aver visto tutti i film.

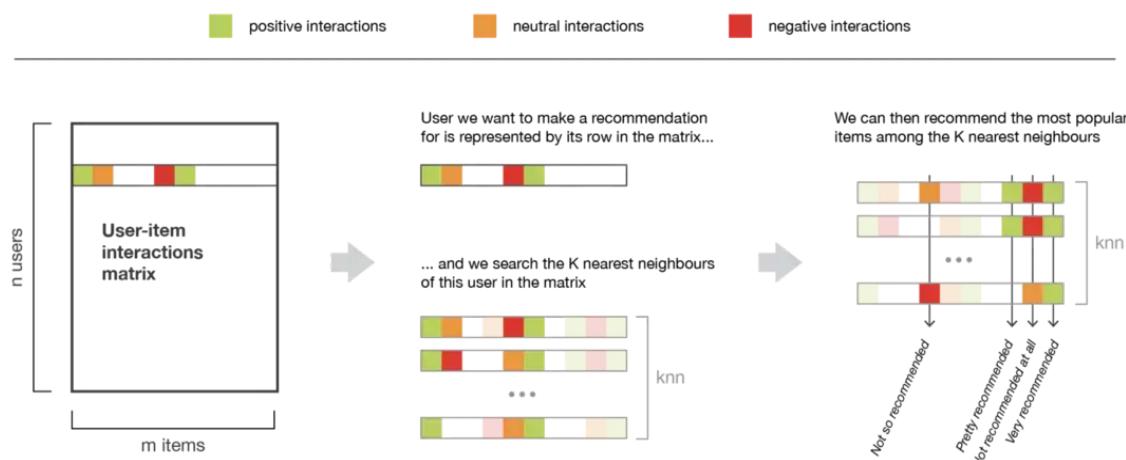


Invece di cercare gli item più simili si fa una ricerca di **SIMILAR USERS** \Rightarrow + serendipity
I metodi di rating degli user possono essere:

- Esplicativi : stelle - votazioni
- Impliciti: tempo trascorso in una pagina, tempo di visione film o ascolto di una canzone

ALGORITMO:

- Trovare un set di utenti simili(Neighbors) a $u \rightarrow N(u)$
- Esplorare le loro interazioni/rating per costruire dei suggerimenti per u



USER PROFILE

Lo user profile è un vettore di dimensione I

- $u[i] = r$ se l'utente U ha dato un rating all'item i .
- u è la riga $R[u::]$ della rating matrix

USER SIMILARITY - PEARSON CORRELATION

Come troviamo gli utenti più simili ad u $N(u)$?

Viene utilizzata la CORRELAZIONE DI PEARSON, che rappresenta una misura normalizzata della correlazione lineare, è molto grande se u e v aumentano/diminuiscono in maniera simili sopra/sotto la loro media \bar{u} e \bar{v} .

$$p(u, v) = \frac{\text{cov}(u, v)}{\sigma_u \sigma_v} = \frac{\sum_i (u[i] - \bar{u})(v[i] - \bar{v})}{\sqrt{\sum_i (u[i] - \bar{u})^2} \sqrt{\sum_i (v[i] - \bar{v})^2}}$$

Come computiamo i suggerimenti?

$\forall i \in I$ computiamo lo score dell'utente u , considerando i rating degli utenti per l'item i e pensando il rating per la similarità degli utenti con u .

$$s_{ui} = \frac{\sum_{v \in N(u)} (v[i] - \bar{v}) * p(u, v)}{\sum_{v \in N(u)} |p(u, v)|}$$

Eventualmente s_{ui} è usato per il rank di ogni oggetto in I

Aggiorniamo la formula per predirre il rating dell'utente u

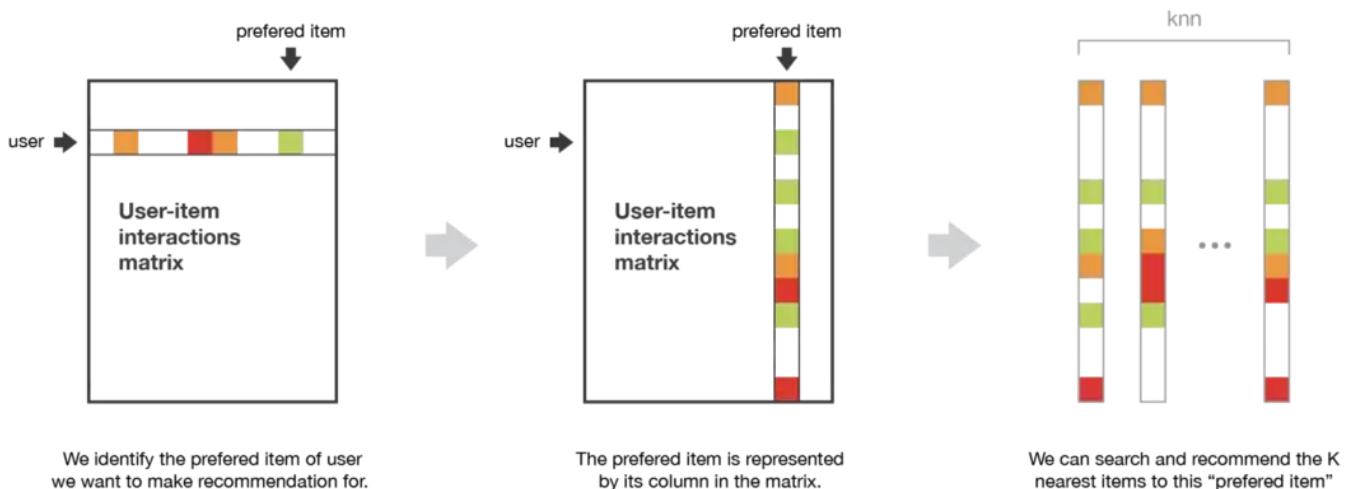
$$r_{ui} = \bar{u} + s_{ui} = \bar{u} + \frac{\sum_{v \in N(u)} (v[i] - \bar{v}) * p(u, v)}{\sum_{v \in N(u)} |p(u, v)|}$$

RIASSUMENDO

- Efficienza nella costruzione del modello : user similarity costosa $O(|U|^2)$ e molto costoso anche se calcolata offline, per ogni interazione nuova dovrebbe essere ricalcolata (solitamente viene ricalcolata ogni tot tempo)
- Efficienza nel generare suggerimenti : calcolare il vicino non è necessario se fatto offline
- SERENDIPITY \rightarrow molto grande
- COLD-START \rightarrow parziale, necessitiamo di alcuni rating per poter iniziare
- SPARSITY : pochi rating \Rightarrow matrice vuota

ITEM BASED COLLABORATIVE FILTERING METHODS

Per creare nuovi suggerimenti, l'idea dell'item based è quella di trovare degli item simili a quelli con cui l'utente ha già interagito positivamente. 2 item sono considerati simili se la maggior parte degli utenti che hanno interagito con entrambi lo hanno fatto in maniera simile. Questo metodo è item-centred poiché rappresenta gli item in base alle interazioni che hanno avuto con loro e valuta le distanze tra questi elementi.



Consideriamo gli items più piaciuti all'utente, rappresentato dal vettore di interazione di tutti gli utenti con quel item.

Calcoliamo poi la similarità tra questi item e tutti gli altri

Una volta calcolata possiamo tenere i $k-N(i)$ vicini a questi item e raccomandarli all'utente u .

Un Item i è rappresentato da un vettore di dimensione $|U|$.

$i[u] = r$ se l'user U da un voto r all'item i .

i è la colonna $R[:, i]$ della rating matrix

ALGORITMO calcolo similarità

Assumiamo di voler generare dei suggerimenti per un utente.

- 1) Per stimare lo score di un item i
- 2) Troviamo i vicini $N(i)$, di altri items votati da u e simili ad i .
- 3) Verifichiamo i ratings dell'utente u agli item $N(i)$ per calcolare uno score per i .

ITEM SIMILARITY - ADJUSTED COSINE SIMILARITY

Consideriamo il fatto di misurare la similarità di due item sulla base dei rating degli utenti a questi item. Pensandoci però, due utenti potrebbero avere scale di valutazioni differenti, ad esempio uno tende a valutare in maniera più ottimistica/soggettiva, mentre un altro in maniera più pessimistica/oggettiva, questo porta ad un problema di scala nella considerazione delle valutazioni.

Quindi per ogni rating di un item i di un utente u , verrà sottratta la media di rating dell'utente u , normalizzando così i ratings sulla stessa scala ci aiuta a superare questa problematica.

$N(i)$: vicini di i

$\forall j \in I: a - \cos(i, j) \rightarrow$ troviamo i top k vicini.

$$a - \cos(i, j) = \frac{\sum_u (i[u] - \bar{u})(j[u] - \bar{u})}{\sqrt{\sum_u (i[u] - \bar{u})^2} * \sqrt{\sum_u (j[u] - \bar{u})^2}}$$

Esiste anche un metodo simile in cui, invece di sottrarre con la valutazione media dell'utente, si sottrae con la valutazione media degli articoli. Questo aiuta a capire quanto le valutazioni degli utenti si discostano dalla valutazione media degli item.

Come computiamo i suggerimenti?

$\forall i \in I$ computiamo lo score dell'utente u per gli item i

$$s_{ui} = \frac{\sum_{j \in N(i)} j[u] * (a - \cos(i, j))}{\sum_{j \in N(i)} |a - \cos(i, j)|}$$

RIASSUMENDO

- Efficienza nella costruzione del modello : user similarity costosa $O(|I|^2)$ più efficiente di USER-BASED in quanto $|I| \ll |U|$ solitamente.
- Efficienza nel generare suggerimenti : calcolare il vicino non è necessario se fatto offline
- SERENDIPITY → più bassa rispetto a USER-BASED in quanto il ranking dipende dai rating dell'utente u
- COLD-START → parziale, necessitiamo di abbastanza ratings per nuovi item/utenti
- SPARSITY : più ampia rispetto a USER-BASED in quanto gli items hanno una probabilità maggiore di condividere i ratings

BONTÀ DELLA RACCOMANDAZIONE

$$MAE = \frac{\sum_{i=1}^N |R[u, i] - r_{ui}|}{N}$$

$R[u, i]$ → valore che sappiamo, rating dato dallo user u all'item i

r_{ui} → valore stimato

17. DOCUMENT SIMILARITY

Supponiamo di essere interessati a trovare documenti simili ad un altro documento usato come query. Un approccio basico è di rappresentare i documenti come set di parole distinte e poi affidarsi alla Jaccard Similarity.

Questo è un problema comune in vari scenari:

- Trovare duplicati/pagine mirror nel web
- Clusterizzare news simili
- Trovare user/item simili per sistemi di recommendation

Focalizzandosi su un documento di tipo testuale, vogliamo ottenere i seguenti obiettivi:

- Efficienza in tempo e spazio
- Andare oltre lo string matching

Fare semplice string matching risulta molto dispendioso in termini di efficienza. E' chiaro che è necessario applicare qualche tipo di mapping per le stringhe in ID numerici. Questo perché operazioni di uguaglianza, ricerca, e molte altre sono molto più efficienti e richiedono molta meno memoria se si ha a che fare con numeri invece che stringhe.

Inoltre vogliamo andare oltre allo string matching semplicistico.

ESEMPIO.

“Players are ready for the next match” e “Player is ready for the next matches”

Facendo semplice string matching queste frasi hanno solo 4 parole in comune e le rimanenti parole sono identiche fino al suffisso.

K-SHINGLES

K-shingles sono un metodo sintattico per riuscire a raggiungere gli obiettivi precedentemente riportati.

Consideriamo un documento di stringhe, un K-shingles è definito come un set di sottostringhe di lunghezza K che appaiono nel documento

ESEMPIO.

Given the document "Players are ready for the next match", its 3-shingles are defined as
K=3

{“Pla”, “lay”, “aye”, “yer”, “ers”, “rs”, “s a”, “ar”, “are”, “re”, “e r”, “re”, “rea”, “ead”, “ady”, “dy”, “y f”, “fo”, “for”, “or”, “r t”, “th”, “the”, “he”, “e n”, “ne”, “nex”, “ext”, “xt”, “t m”, “ma”, “mat”, “atc”, “tch”, “che”, “hes”}

La similarità tra due documenti può essere misurata applicando la JACCARD tra i loro shingles.

Il vantaggio di utilizzare shingles è quello di memorizzare parti di stringhe e queste contribuiscono alla similarità quando i documenti ne condividono.

La lunghezza K degli shingles dovrebbe essere deciso sulla base dei documenti che vogliamo valutare.

- BASSO VALORE K : aumenta la probabilità di avere degli shingles match, aumentando così la JACCARD SIMILARITY.
- ALTO VALORE K : aumenta notevolmente la selettività tra shingles per effettuare dei match.
- Tipicamente un valore compreso tra 5(documenti brevi) e 9(documenti lunghi) sono considerati delle buone opzioni.
- Il valore di K ha un impatto sul numero di shingles. Su un alfabeto di 27 caratteri (compreso lo spazio) ci sono 27^K shingles. Ma non tutti esistono nel lessico linguistico. Una buona approssimazione è quella a 20^K shingles.

Notiamo che $2^7 < 2^{32}$ e $2^9 << 2^{64}$ il che permette 32 e 64 bit per rappresentarli tutti.

Questa rappresentazione può essere ottenuta attraverso un'enumerazione e map degli shingles oppure facendo hashing degli shingles in bistring di 32 o 64 bit (scartando collisioni)

Infine, il numero di shingles di un documento d è ottenuto da $|d| - k + 1$, dove $|d|$ è la lunghezza del documento in caratteri. In questo modo la rappresentazione del documento rimane gestibile indipendentemente da K.

ALTRI CONTESTI

In altri contesti gli shingles sono definiti sulla base delle parole invece che sui caratteri, inoltre una strategia che ha dimostrato di funzionare bene con la ricerca di similarità delle notizie, è quella di definire gli shingles come una sequenza di parole che si verificano dopo una stop-word (la logica è che le stop-word introducono concetti rilevanti).

MIN HASHING

Anche se gli shingles introducono qualità nello string matching sono comunque troppo dispendiosi.

Introduciamo quindi la strategia dell SIGNATURE(firma), chiamata MIN HASHING per compattare significativamente la rappresentazione di un documento, garantendo un supporto efficace alla ricerca di similarità

Rappresentiamo le informazioni in una matrice booleana M

- Righe → l'universo di tutti i possibili set di shingles , ipotizziamo 64bit, quindi $[0, \dots, 2^{64} - 1]$
- Colonne → i documenti
- $M(R,C) = 1 \rightarrow$ in riga r e colonna c se e solo se r è un membro di s

La matrice è sparsa in memoria

Supponiamo di avere 2 documenti= A={a,d} e B={b,d,e}, possiamo rappresentarli così

| | A | B | | A | B |
|---|---|---|---|---|---|
| a | 1 | 0 | c | 0 | 0 |
| b | 0 | 1 | d | 1 | 1 |
| c | 0 | 0 | b | 0 | 1 |
| d | 1 | 1 | e | 0 | 1 |
| e | 0 | 1 | a | 1 | 0 |

Chiamiamo π una permutazione randomica degli elementi nel documento, quindi le righe della matrice ottenendo una nuova matrice. La permutazione è globale e identica per ogni documento, grazie alla permutazione delle righe.

Definiamo funzione "hash" per un DOCUMENTO D(colonne)::

- $h(D) =$ l'indice nella prima riga post permutazione, dove la colonna D = 1, oppure
- $h(D) =$ l'indice del primo elemento di D post permutazione

Usare K permutazioni indipendenti estraendo il min hash per ognuna per creare una MIN-HASH SIGNATURE(firma) di ogni documento D.

La MIN HASH SIGNATURE può essere utilizzata per dare una stima della similarità tra 2 documenti.

Supponiamo di avere:

- Universo shingles U = {A,B,C,D,E,F,G}
- Documenti X = {A,B,F,G} e Y = {A,E,F,G}
- $X \cup Y = \{A, B, E, F, G\}$
- $X \cap Y = \{A, F, G\}$

Rows C,D could be anywhere
they do not affect the probability

The * rows belong to the union

| | X | Y |
|---|---|---|
| A | 1 | 1 |
| B | 1 | 0 |
| C | 0 | 0 |
| D | 0 | 0 |
| E | 0 | 1 |
| F | 1 | 1 |
| G | 1 | 1 |

The question is what is the value
of the **first** * element

| | X | Y |
|---|---|---|
| D | * | * |
| C | * | * |
| | | |
| | | |
| | | |
| | | |
| | | |

| | X | Y |
|---|---|---|
| A | 1 | 1 |
| B | 1 | 0 |
| C | 0 | 0 |
| D | 0 | 0 |
| E | 0 | 1 |
| F | 1 | 1 |
| G | 1 | 1 |

| | X | Y |
|---|---|---|
| D | * | * |
| C | * | * |
| | | |
| | | |
| | | |
| | | |
| | | |

If it belongs to the intersection
then $h(X) = h(Y)$

| | X | Y |
|---|---|---|
| A | 1 | 1 |
| B | 1 | 0 |
| C | 0 | 0 |
| D | 0 | 0 |
| E | 0 | 1 |
| F | 1 | 1 |
| G | 1 | 1 |

| | X | Y |
|---|---|---|
| D | * | * |
| C | * | * |
| | | |
| | | |
| | | |
| | | |
| | | |

| | X | Y |
|---|---|---|
| A | 1 | 1 |
| B | 1 | 0 |
| C | 0 | 0 |
| D | 0 | 0 |
| E | 0 | 1 |
| F | 1 | 1 |
| G | 1 | 1 |

| | X | Y |
|---|---|---|
| D | * | * |
| C | * | * |
| | | |
| | | |
| | | |
| | | |
| | | |

Every element of the union is equally likely
to be the * element

$$\Pr(h(X) = h(Y)) = \frac{|[A, E, G]|}{|[A, B, E, F, G]|} = \frac{3}{5} = \text{Sim}(X, Y)$$

| | X | Y |
|---|---|---|
| A | 1 | 1 |
| B | 1 | 0 |
| C | 0 | 0 |
| D | 0 | 0 |
| E | 0 | 1 |
| F | 1 | 1 |
| G | 1 | 1 |

| | X | Y |
|---|---|---|
| D | * | * |
| C | * | * |
| | | |
| | | |
| | | |
| | | |
| | | |

| | S ₁ | S ₂ | S ₃ | S ₄ |
|---|----------------|----------------|----------------|----------------|
| A | 1 | 0 | 1 | 0 |
| B | 1 | 0 | 0 | 1 |
| C | 0 | 1 | 0 | 1 |
| D | 0 | 1 | 0 | 1 |
| E | 0 | 1 | 0 | 1 |
| F | 1 | 0 | 1 | 0 |
| G | 1 | 0 | 1 | 0 |

Signature matrix

| | Actual | Sig |
|------------------------------------|--------|-----|
| (S ₁ , S ₂) | 0 | 0 |
| (S ₁ , S ₃) | 3/5 | 2/3 |
| (S ₁ , S ₄) | 1/7 | 0 |
| (S ₂ , S ₃) | 0 | 0 |
| (S ₂ , S ₄) | 3/4 | 1 |
| (S ₃ , S ₄) | 0 | 0 |

Zero similarity is preserved
High similarity is well approximated

La similarità delle SIGNATURES è quindi il rapporto :

$$P(\min(\pi_A) = \min(\pi_B)) = \frac{\#shingles\ in\ comune}{\#shingles\ in\ comune + \#shingles\ solo\ di\ A + \#shingles\ solo\ di\ B}$$

tra le hash functions dove corrispondono.

Ma quante permutazioni dobbiamo utilizzare?

Per avere una misura precisa eseguo il test con diverse permutazioni(indipendenti)
Conto le uguaglianze K e trovo:

$J \approx \frac{K}{M}$ dove K è il numero di uguaglianze e M il numero di permutazioni.

Più M è grande e più $\frac{K}{M} \rightarrow J$, ma anche più costoso(potrei fare anche hashing delle permutazioni per migliorare in complessità spaziale)

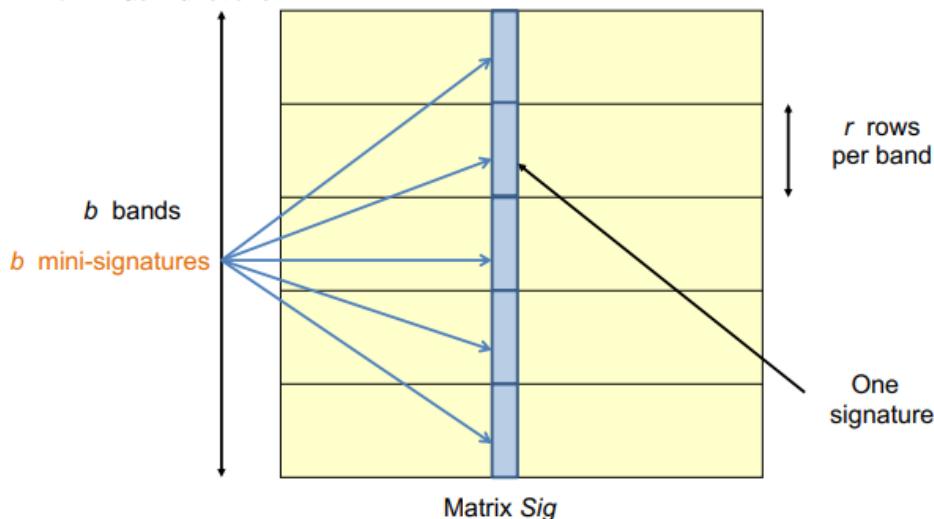
Quello che vogliamo è un giusto compromesso tra accuracy e costo di computazione e memorizzazione. In generale vorremmo un M che sia significativamente piccolo rispetto al numero medio di termini in un documento.

LHS: LOCALITY SENSITIVE HASHING FOR JACCARD DISTANCE

Anche con l'uso delle min-hash-signature, il problema di trovare documenti simili rimane troppo costoso. Utilizziamo la min-hashing-signature per definire una funzione di hash per cui 2 documenti simili hanno lo stesso hash con alta probabilità.

Dunque, l'idea è che documenti simili hanno firme simili e possiamo usare la hash function per accedere alla zona di hash(bucket) e trovare i documenti simili a questo poiché hashati nella stessa zona.

$n = b * r$ hash functions



PROCEDIMENTO:

La matrice di Signature viene suddivisa in

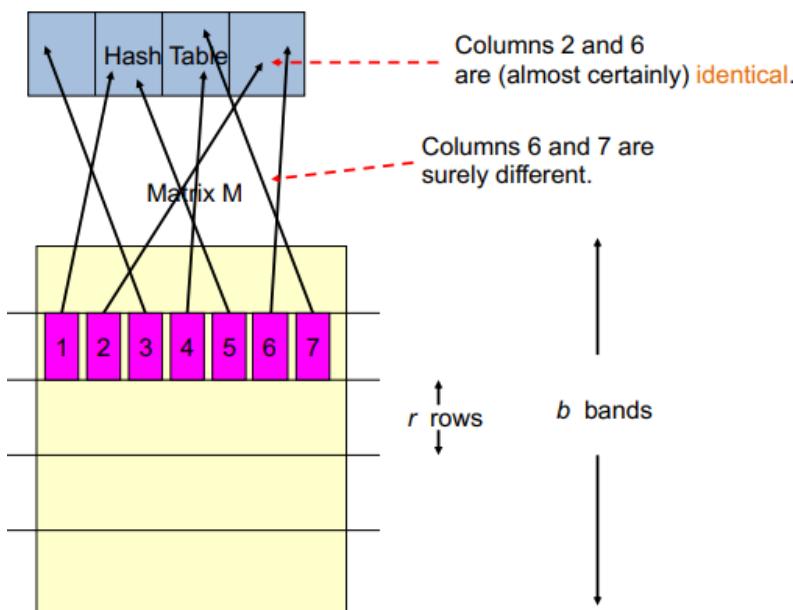
- b BANDE dove ogni banda ha :
 - r righe
 - c colonne.

Ogni colonna è una porzione di una min-hash-signature più grande di un documento D e prende il nome di MINI-SIGNATURE con r MIN-HASH functions(permutazioni).

Per ogni banda, viene effettuato l'hashing di tutte le sue MINI-SIGNATURE(colonne), prende il nome di SUPER SIGNATURE, in una hash table con K zone di hash(chiamati bucket, es. Se k=3 , tutti i valori dati in pasto alla funzione vengono hashati in 1,2 o 3).

- Maggiore è K e maggiore è la probabilità che documenti negli stessi bucket siamo identici(utilizzato in LHS)
- Più k è piccolo e più è alta la probabilità di falsi positivi

Coppie di colonne sono candidate quando hanno hash identici almeno in una banda. L'idea è quindi quella che 2 documenti sono simili, se hanno delle colonne candidate in almeno 1 banda (MINI-SIGNATURE hashate nello stesso bucket in almeno 1 banda).



Potrebbe succedere che colonne non hashate assieme in una banda siamo nello stesso bucket in una banda diversa questo non le preclude dal fatto di essere coppie candidate.

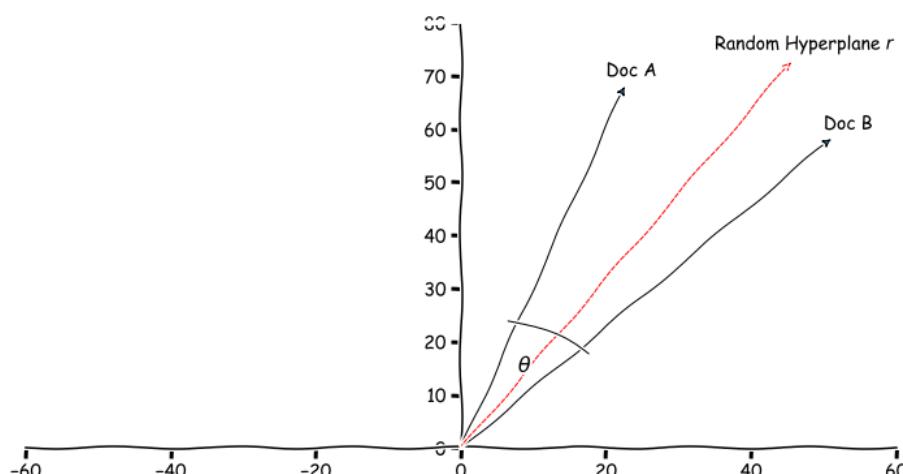
ALGORITMO di RICERCA DOC SIMILI

- Dato un nuovo documento, viene calcolato il MIN-HASH e le sue b-SUPER-SIGNATURES(un hash per banda delle sue MINI-SIGNATURE)
- Accediamo alle hash table di ogni banda attraverso le SUPER-SIGNATURE
- L'unione dei documenti a cui appartengono le MINI-SIGNATURE hashate negli stessi bucket della SUPER SIGNATURE del nostro documento è l'insieme dei documenti simili.

SIM-HASHING for COSINE SIMILARITY

Sim hashing è una funzione di hash la cui proprietà è che più simili sono gli input tra loro e minore è la loro distanza di Hamming dei loro hash.

Consideriamo due documenti A e B rappresentati da uno spazio vettoriale a 2 dimensioni. I 2 documenti-vettori sono separati da un angolo θ .



Prendiamo ora un vettore casuale r nell'intervallo $[0,180]$ quindi:

- Probabilità che r cada tra i due documenti è $\theta/180$
- Probabilità che r cada nello stesso lato per entrambi è $(180 - \theta)/180$

L'iper piano r rappresenta una singola feature che viene hashata in 1 o 0 a seconda della hash function, in questo caso

- La signature di A è 1 se A è sopra,
- 0 altrimenti

Quindi con la sim-hashing signature possiamo stimare la similarità dei documenti, all'aumentare del numero di iper-piani casuali provati, l'accuratezza aumenta

STRATEGIA

- Prendere un campione di iperpiani r come una sequenza di (+1,-1) distribuiti randomicamente in modo uniforme
- Dato un documento A , computare il dot product $r * A$ (simile al vedere se A è sopra o sotto l'iperpiano r)
- Se positivo la signature è 1, altrimenti 0
- Ripetere m volte per computare una signature di m bit
- Dati due documenti, calcolare la distanza di hamming tra le loro signatures per stimare l'angolo tra loro e quindi la cosini similarity.