Riassunto di Data and Web Mining

Un gruppo di buon samaritani

Dicembre 2022

Prefazione

Ci scusiamo per gli eventuali errori di battitura presenti nel documento.
Gli argomenti presenti in questo documento si rifanno a quelli spiegati nel corso dell'anno accademico 2022/2023.
Qui dentro sono riassunti i concetti che ho ritenuto più importanti da sapere.
Potremmo aver saltato qualche piccola parte di alcuni argomenti causa pigrizia, confusione o altre ragioni di cause sconosciute.
Non ci assumiamo nessuna responsabilità riguardo il voto che prenderete se studiate solo da questo riassunto.
A dirla tutta, non siamo delle persone responsabili in primo luogo quindi il problema non dovremmo manco porcelo.
Per chi stesse studiando durante le feste natalizie: tanti auguri.
Chi invece no: tanti auguri lo stesso.
Potremmo aver sviluppato una forte dipendenza da caffeina: se vedete il documento non concluso, una probabile causa potrebbe essere arresto cardiaco.
La seconda causa è che abbiamo trovato degli appunti migliori di questi: nel caso fosse, faremo un omaggio alla madonna per la persona che li ha scritti.
Se avete letto fino a questo punto, ci scusiamo per l'inutile perdita di tempo.

BUONA LETTURA$_{mandateaiutopls}$!

Indice

Capitolo 1

Introduzione

1.1 Supervised vs Unsupervised

- **Supervised learning**: È un approccio che è definito dall'uso di set di dati già etichettati: questi set di dati sono progettati per allenare (o "supervisionare") gli algoritmi nella classificazione dei dati o nella previsione accurata dei risultati.

  Si parla di Supervised Learning quando è l'umano a fornire già delle etichette ai dati, in modo da dare una mano all'al-

goritmo nella sua fase di allenamento in quanto saprà già dall'inizio saprà quali sono i dati corretti.

L'apprendimento supervisionato può essere separato in due tipi di problemi durante il data mining: classificazione e regressione.

- **Unsupervised learning**: Utilizza algoritmi di apprendimento automatico per analizzare e raggruppare set di dati non etichettati: questi algoritmi apprendono da soli i migliori pattern che rappresentano i dati, senza la necessità dell'intervento umano.

1.2 Regressione vs Classificazione

- **Regressione:** L'obiettivo principale dei problemi di regressione è quello di trovare una funzione ottimale che riesca a combinare alcune variabili X, dette anche *predittori*, per produrre un valore in output, ovvero Y, il più preciso possibile. Alcuni esempi di problemi di regressione possono essere la stima dei prezzi di vendita di immobili oppure la stima della probabilità di un certo evento binario, come la probabilità di terremoti in certe regioni di un paese.

- **Classificazione:** L'altra categoria di problemi sono quelli di classificazione, ovvero problemi che richiedono di trovare una funzione che utilizza alcuni predittori X per produrre in output un valore Y discreto (può essere un'etichetta o una categoria). Un esempio di problema di classificazione può essere la categorizzazione di un sito come affidabile oppure non affidabile.

1.3 Train, Validation and Test

L'utilizzo di un algortimo per risolvere un certo problema prevede fondamentalmente due step: l'allenamento e la classificazione. È comune anche aggiungere la fase di validazione dopo la fase di allenamento, in cui è possibile, appunto, validare il modello[1] fornito dall'algoritmo.

1.3.1 Fase di Train

Il processo di allenamento dell'algoritmo consiste nel fornirgli un dataset su cui può imparare a classificare i dati (da qui il nome "Train Set") e costruire un modello ottimale sulla base di ciò che ha appreso.

1.3.2 Fase di Validation

Nella fase di Train, l'algoritmo impara le relazioni tra input X e output Y. Per evitare di cadere nell'overfitting e avere una migliore capacità predittiva, viene dato al nostro modello dei dati che non ha mai visto e gli facciamo fare una predizione (anche questi dati devono essere già etichettati).

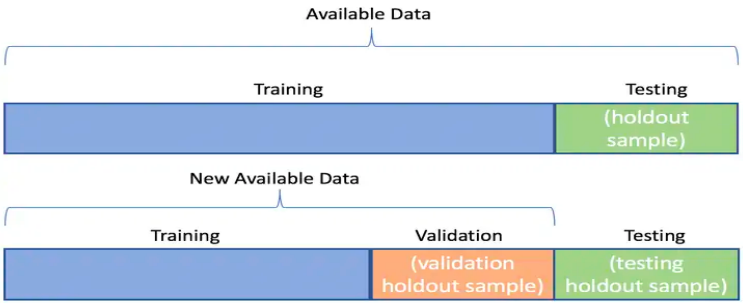
Dopo le predizioni su questi dati, verranno confrontati i risultati del modello (che denomineremo con $\hat{y}$) con le y reali per vedere quanto il modello riesce a prevedere con una buona approssimazione la variabile target. Nel caso di performance scarse, bisogna

aggiustare gli iperparametri del modello e ripartire dalla fase di train, finché il risultato ottenuto dalla fase di validation non sarà soddisfacente.

1.3.3 Fase di Test

Il processo di classificazione consiste nell'utilizzo del modello, precedentemente costruito nella fase di training, per predire le categorie o stimare i valori di nuovi dati.
In questa fase viene fornito al modello un set di dati su cui valutare il modello; viene utilizzato solo dopo che il modello è stato completamente allenato. Il Test Set contiene dati accuratamente campionati che cercano di coprire le varie casistiche che il modello dovrà affrontare se utilizzato nel mondo reale.

Figura 1.1: Esempio di divisione del dataset

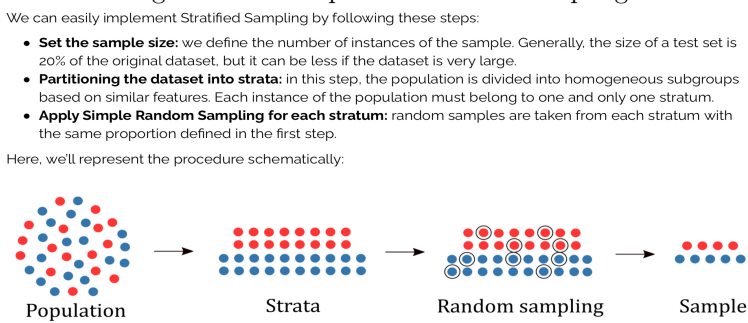


1.4 Stratified Sampling

Il campionamento casuale generalmente va bene se il set di dati originale è sufficientemente grande proprio perché ogni sottoinsieme di n istanze ha la stessa probabilità di essere selezionato come campione di qualsiasi altro sottoinsieme di n istanze.
Ma se il dataset non è sufficientemente grande, viene introdotta una distorsione a causa dell'errore di campionamento.

Il **Stratified Sampling** è un metodo di campionamento che riduce l'errore di campionamento nei casi in cui la popolazione può essere suddivisa in sottogruppi: la popolazione viene divisa in sottogruppi omogenei, chamati *strati*, per poi applicare il campionamento casuale semplice all'interno di ciascun sottogruppo.

Figura 1.2: Esempio di Stratified Sampling



1.5 Cross-Validation

Il **Cross-Validation** è una procedura di ricampionamento utilizzata per valutare i modelli di machine learning su un campione di dati limitato. La procedura ha un singolo parametro, ovvero k, che si riferisce al numero di gruppi in cui deve essere suddiviso un dato campione di dati.
Pertanto, la procedura viene spesso chiamata
K-Fold Cross-Validation.

Scelta di k

Il valore di k deve essere scelto con cura per il campionamento di dati: un valore scelto in modo errato può risultare in una valutazione errata dell'abilità del modello.

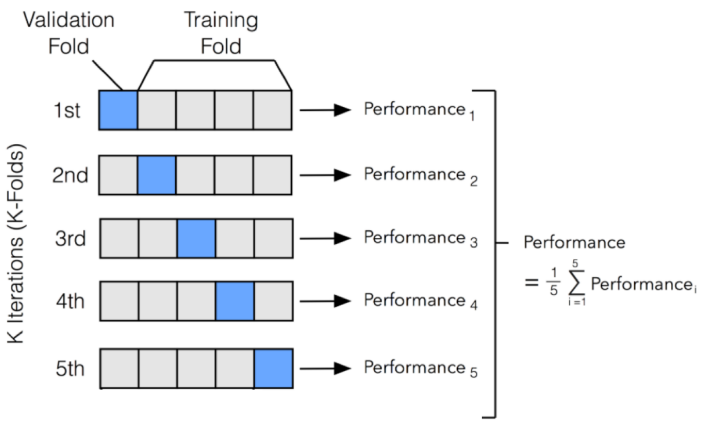
Le tattiche più comuni per scegliere il valore di k sono:

- **Rappresentativo**: il valore di k è scelto in modo tale che ciascun gruppo di campioni di dati train/test sia sufficientemente grande da essere statisticamente rappresentativo del set di dati più ampio;

- $\boldsymbol{k = 10}$: il valore per k è fissato a 10, un valore che è stato trovato attraverso la sperimentazione e che è risultato generalmente buono nella stima dell'abilità del modello;

- $\boldsymbol{k = n}$: il valore per k è fissato a n, dove n è la dimensione del dataset, per dare a ciascun campione l'opportunità di essere utilizzato nel validation set.
  Questo approccio è chiamato
  Leave-One-Out Cross-Validation (LOOCV).

Figura 1.3: Esempio di Cross-Validation (k = 5)



1.6 Underfitting vs Overfitting

L'overfitting e l'underfitting sono due problemi tipici in cui il modello raggiunge scarse performance nella classificazione dopo l'allenamento ma per motivi diversi.

- **Underfitting** : ci sono pochi parametri nel modello e un'elevata discrepanza nella classificazione (Alto Bias). Il processo di apprendimento è troppo semplice.
- **Overfitting** : ci sono troppi parametri nel modello e un'elevata variabilità della classificazione. Il modello è troppo complesso e sensibile ai dati di training (Alta Varianza).

Figura 1.4: Esempio di modello che sottostima

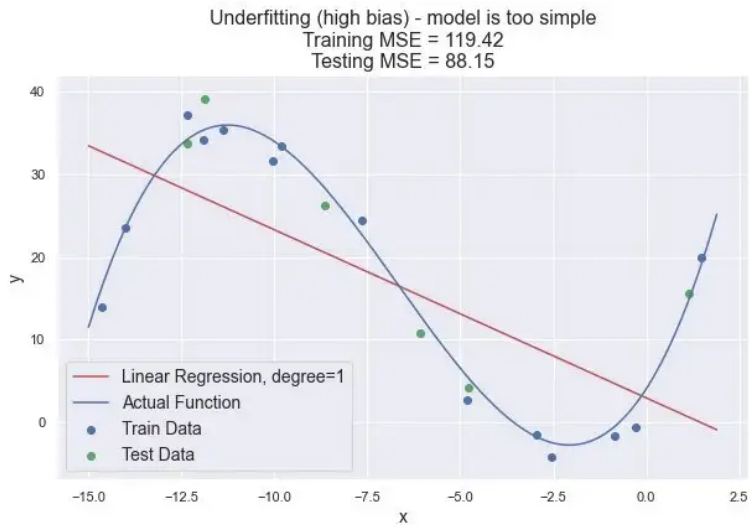


Figura 1.5: Esempio di modello che sovrastima

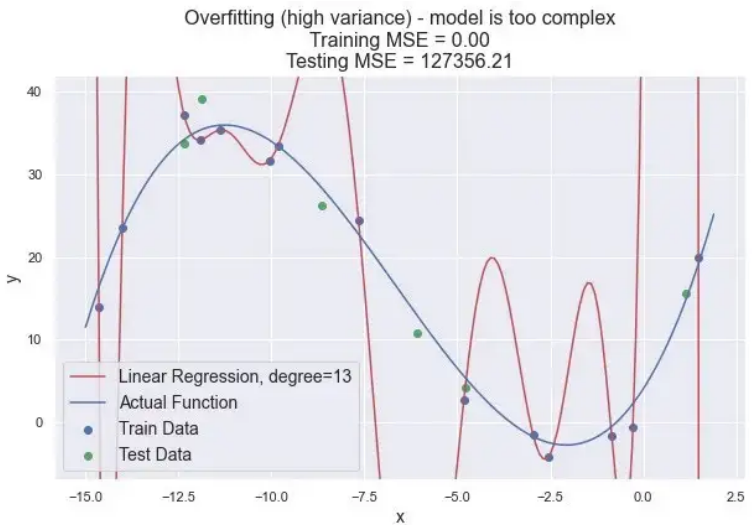
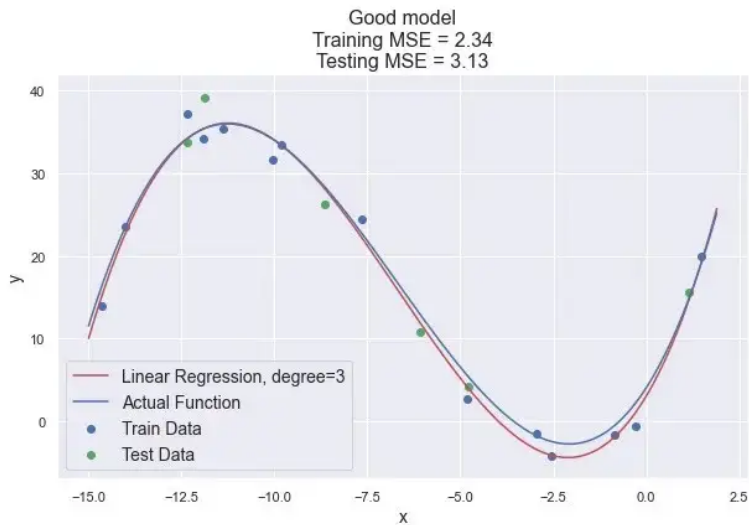


Figura 1.6: Esempio di modello buono, che non sovrastima o sottostima



1.7 ROC: Receiver Operating Characteristics

L'analisi delle **Receiver Operating Characteristic** (ROC) è un approccio grafico per analizzare le prestazioni di un classificatore e studiarne l'output.

Utilizza una coppia di statistiche, tasso di Veri Positivi e tasso di Falsi Positivi. Le statistiche sono tracciate su un grafico bidimensionale, con tasso di falsi positivi sull'asse delle x e tasso di veri positivi sull'asse delle y.

Poiché il classificatore può essere un valore arbitrario (output continuo), il confine tra le classi deve essere determinato da un valore di soglia oppure un'etichetta di classe discreta, che indica una delle classi.

Figura 1.7: Esempio di grafico ROC

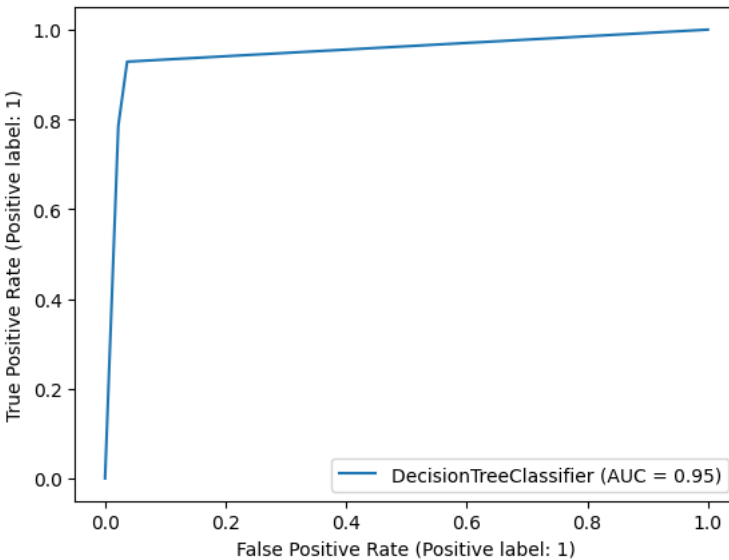
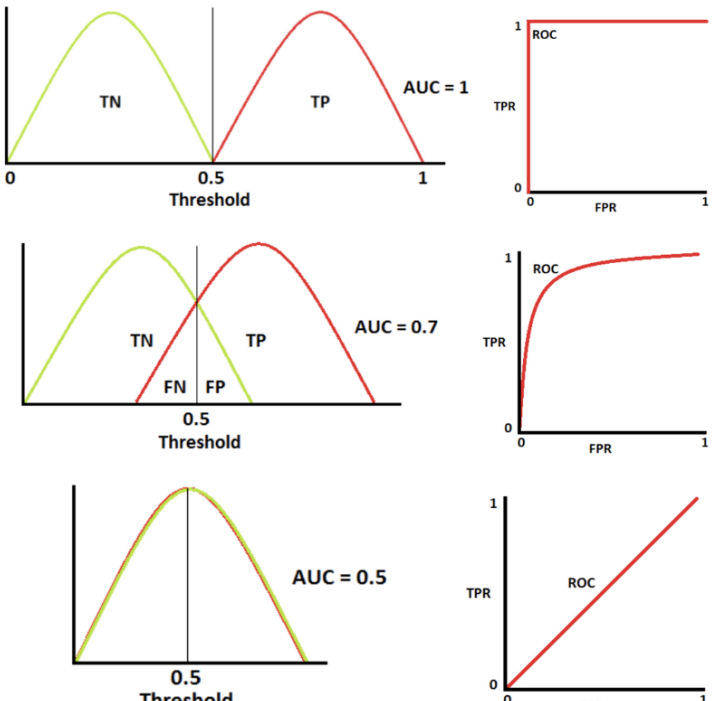


Figura 1.8: Situazione ideale (1), situazione tipica (2) e situazione peggiore (non risce a discriminare) (3)



Capitolo 2

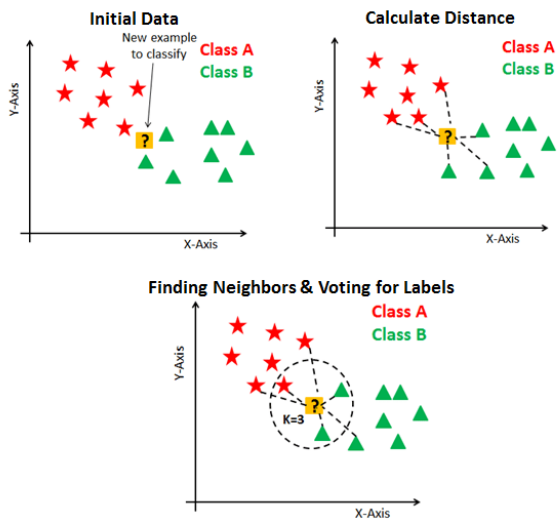
Supervised Learning

2.1 KNN: K-Nearest-Neighbor

L'algoritmo KNN è un classificatore di Supervised Learning che utilizza la vicinanza dei dati per effettuare classificazioni o previsioni sul raggruppamento di un singolo punto. Sebbene possa essere utilizzato per problemi di regressione o classificazione, viene generalmente utilizzato come algoritmo di classificazione, basandosi sul presupposto che punti simili possono essere trovati l'uno vicino all'altro.

Per problemi di classificazione, un'etichetta di classe viene assegnata sulla base di un voto a maggioranza, ad esempio viene utilizzata l'etichetta più frequentemente rappresentata attorno a un determinato punto. I problemi di regressione utilizzano un concetto simile al problema di classificazione, ma in questo caso viene presa la media dei k vicini più vicini per fare una previsione su una classificazione.

Figura 2.1: Esempio di algoritmo kNN



2.1.1 Distanza

Ovviamente, basandosi sul concetto di vicinanza, bisogna definire la metrica di distanza: la distanza euclidea è una metrica di distanza molto comune, che misura la lunghezza del segmento avente per estremi i due punti.

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + \cdots + (p_d - q_d)^2}$$

dove d è la dimensione dello spazio in cui si trovano i punti. Per esempio, in uno spazio tridimensionale la formula sarà:

$$d(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2}$$

Bisogna fare attenzione nel caso non si fosse solo interessati a trovare i punti dentro un certo raggio ma si volesse dare più importanza ai punti più vicini a quello preso in questione: un fattore di peso comune è $w = \frac{1}{d^2}$.

Un altro modo per sensibilizzare la distanza è quello di riscalare i dati. I due metodi più comuni sono:

- **StandardScaler** : funziona bene se i dati seguono una distribuzione gaussiana/normale. Non rimappa ogni feature nell'intevallo $[0, 1]$ ma dipende dalla variamza: una feature con grande varianza potrebbe impattare fortemente sulla distanza complessiva;

- **MinMaxScaler** : riscala in un range (di default è $[0, 1]$, ma si può modificare). È solitamente più sensibile ai valori anomali, ma pesa le features in maniera più uniforme.

2.1.2 Scelta di K

Il valore k nell'algoritmo KNN definisce quanti vicini verranno controllati per determinare la classificazione di un punto specifico. Ad esempio, se $k = 1$, l'istanza verrà assegnata alla stessa classe del suo singolo neighbors più vicino.

La scelta di k è importante per le performance dell'algoritmo in termini di risultati e di costo computazionale: se k è troppo piccolo l'algoritmo sarà sensibile ai punti di rumore all'interno del dataset, mentre se k è troppo grande il costo di computazione potrebbe crescere di molto.

In generale, la scelta di k dipende in gran parte dal dataset poiché i set di dati con più valori anomali probabilmente necessiteranno di valori di k elevati. È consigliato di fare tuning del parametro per vedere quale valore porta al risultato migliore e si consiglia di avere un numero dispari per k per evitare pareggi nella classificazione.

2.1.3 Pigrizia e Memoria

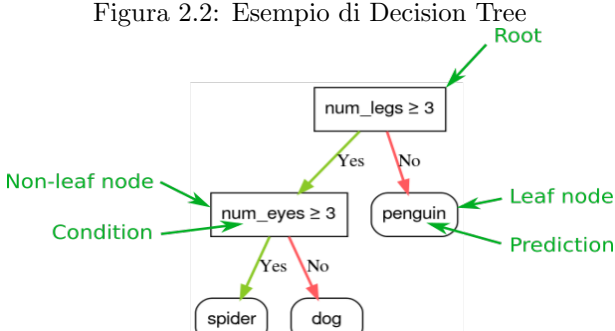
L'algoritmo KNN fa parte della categoria di algoritmi "lazy", ovvero che il calcolo non viene effettuato durante la sessione di train ma avviene quando viene effettuata una predizione. Inoltre fa largo uso della memoria per archiviare i dati di train, il che rende molto costoso in termini di spazio se la dimensionalità dei dati in input è elevata.

2.2 Decision Trees

Il **Decision Tree Learning** è un approccio di apprendimento supervisionato che utilizza un albero decisionale di classificazione o regressione che verrà utilizzato come modello predittivo per trarre conclusioni su un insieme di osservazioni.

I modelli ad albero in cui la variabile target può assumere un insieme discreto di valori sono chiamati *alberi di classificazione*; in queste strutture ad albero, le foglie rappresentano etichette di classe e i rami rappresentano congiunzioni di caratteristiche che portano a quelle etichette di classe. Gli alberi decisionali in cui la variabile target può assumere valori continui (tipicamente numeri reali) sono chiamati *alberi di regressione*.

Figura 2.2: Esempio di Decision Tree

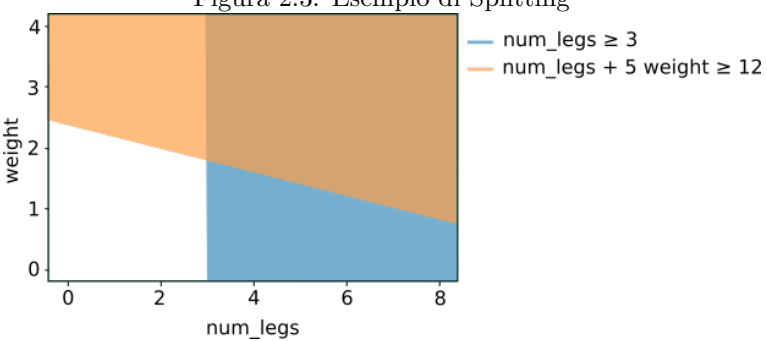


2.2.1 Condizioni Verticali vs Oblique

Una condizione verticale (o anche detta "allineata agli assi", "split verticale") coinvolge una sola feature mentre una condizione obliqua coinvolge molteplici features. Per esempio, $num\_legs \geq 2$ è una condizione verticale mentre $num\_legs \geq num\_eyes$ è una condizione obliqua.

Spesso i Decision Trees sono allenati su condizioni verticali. Tuttavia, lo splitting obliquo è molto più potente perché può esprimere pattern più complessi e può produrre risultati migliori, ad un elevato costo computazionale per il train e l'inferenza.

Figura 2.3: Esempio di Splitting



2.2.2 Condizioni Binarie vs Non-Binarie

Condizioni con due possibili risultati (ad esempio, vero o falso) sono chiamate condizioni binarie. Alberi decisionali contenenti solo condizioni binari sono chiamati *Binary Decision Trees*.

Le condizioni non binarie hanno più di due possibili esiti, dunque hanno un potere discriminante maggiore rispetto a quelle binarie. Gli alberi che contengono decisioni una o più condizioni non binarie sono chiamati *Non-Binary Decision Trees*.

Attenzione: Le condizioni non binarie hanno maggiore probabilità di overfit

2.2.3 Elenco delle Condizioni più Usate

Giusto per essere completi, ecco un elenco dei tipi condizioni che vengono comunemente utilizzate:

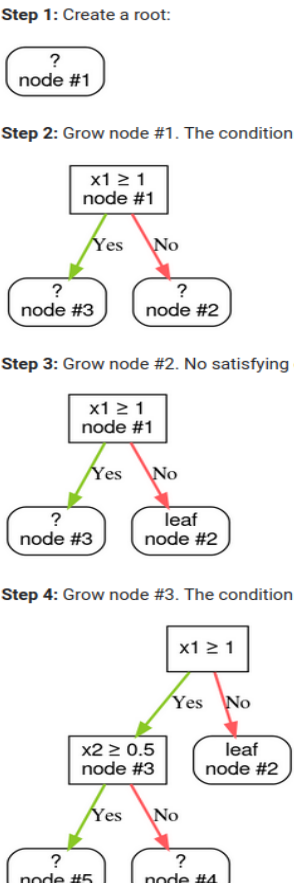
Name	Condition	Example
threshold condition	$\text{feature}_i \geq \text{threshold}$	$\text{num\_legs} \geq 2$
equality condition	$\text{feature}_i = \text{value}$	$\text{species} = \text{“}cat\text{”}$
in-set condition	$\text{feature}_i \in \text{collection}$	$\text{species} \in \{\text{“}cat\text{”}, \text{“}dog\text{”}, \text{“}bird\text{”}\}$
oblique condition	$\sum_i \text{weight}_i \text{feature}_i \geq \text{threshold}$	$5\ \text{num\_legs} + 2\ \text{num\_eyes} \geq 10$
feature is missing	$\text{feature}_i \text{isMissing}$	num_legsisMissing

2.2.4 Costruire un Decision Tree

La costruzione del migliore albero decisionale è un problema NP-Hard, dunque viene generalmente utilizzata un'euristica per costruire un albero ottimale. La maggior parte degli algoritmi utilizzati per allenare gli alberi decisionali funziona con una strategia greedy in stile *divide et impera*: l'algoritmo inizia creando un singolo nodo (la radice) e cresce ricorsivamente, e in maniera greedy, l'albero decisionale. Ad ogni nodo, tutte le possibili condizioni vengono valutate e viene assegnato a loro un punteggio; l'algoritmo seleziona la condizione "migliore", ovvero la condizione con il punteggio più alto.
Dopodiché viene eseguito lo split sulla base della condizione precedentemente scelta.

L'algoritmo, quindi, ripete ricorsivamente e indipendentemente su tutti i nodi figli, valutando condizioni che non sono ancora state scelte come condizioni di split. Quando non vengono trovate condizioni soddisfacenti, il nodo diventa una foglia. La predizione nella foglia è scelta come il valore dell'etichetta più significativo.

Figura 2.4: Esempio di Creazione



2.2.5 Misura dell'Impurità del Singolo Nodo

L'impurità di un nodo misura quanto sono differenti le etichette delle classi per le istanze di dati appartenenti a un nodo comune.

Alcune misure che possono essere utilizzate per misurare l'impurità di un certo nodo t sono:

- **Entropy**: $-\sum_{i=0}^{m-1} p_i(t) \cdot \log_2 (p_i(t))$

  - Massimo: $\log m$, dove m è il numero di classi, quando i records sono equamente distribuiti su tutte le classi, implica poca informazione;

  - Minimo: 0.0 quando tutti i recordi appartengono a una sola classe, implica maggiore informazione (assumiamo che $0 \cdot \log(0) = 0$) (**"Foglia Pura"**).

- **Gini Index**: $1 - \sum_{i=0}^{m-1} p_i(t)^2$

  - Massimo: $(1 - \frac{1}{m})$ quando i records sono equamente distribuiti su tutte le classi, implica poca informazione;

  - Minimo: 0.0 quando tutti i records appartengono a una sola classe, implica maggiore informazione.

- **Classification Error**: $1 - \max [p_i(t)], \ \forall i$

  - Massimo: $(1 - \frac{1}{m})$, dove m è il numero di classi, quando i

records sono equamente distribuiti su tutte le classi, implica poca informazione;

- Minimo: 0.0 quando tutti i records appartengono a una classe, implica maggiore informazione.

- **Gain Ratio**: $\frac{\Delta_{\text{info}}}{\text{SplitInfo}(\mathcal{D})}$ (vedi sotto per Δ_{info})

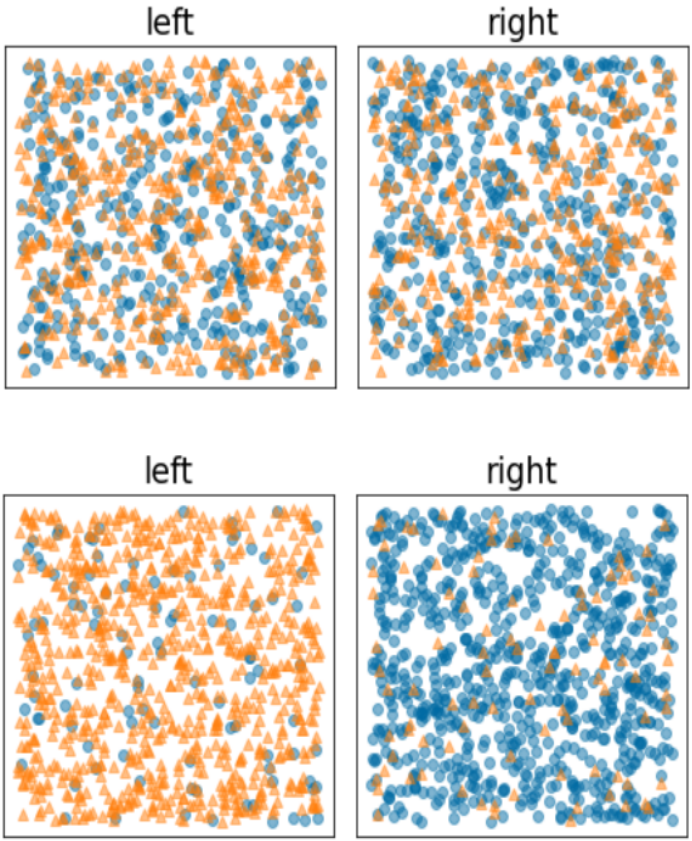
  - dove $\text{SplitInfo}(\mathcal{D}) = -\sum_{i=0}^{k} \left[\frac{N(v_i)}{N} \cdot \log_2(\frac{N(v_i)}{N}) \right]$ con $N(v_i) =$ numero di istanze di training associate al nodo v_i e $N =$ numero di istante di training;

  - Gain Ratio normalizza l'Information Gain per lo SplitInfo di uno split k-ario. Per Decision Trees k-ari (invece che binari), l'Information Gain favorisce gli splits con molte piccole partizioni (è più probabile che siano pure);

  - È usato per compensare attributi che producono un gran numero di nodi figlio.

dove $p_i(t)$ è la frequenza delle istanze di training che appartengono alla classe i al nodo t e m è il numero totale di classi.

Figura 2.5: Cattivo Split vs Buono Split



2.2.6 Impurità Collettiva dei Nodi Figlio

Possiamo calcolare l'impurità dei nodi figlio come:

$$I(children) = \sum_{i=0}^{k} \left[\frac{N(v_i)}{N} \cdot I(v_i) \right]$$

dove:

- k è il numero di nodi figlio;
- N è il numero delle istanze di training;
- $N(v_i)$ è il numero di istanze di training associate con il nodo v_i;
- $I(v_i)$ è l'impurità del nodo v_i calcolata con le formule di errore viste prima.

Esempio: Consideriamo i candidati (a) e (b) per il test della condizione del problema di classificazione dei mutuatari, mostrati nella figura sottostante.

Calcoliamo la **Weighted Entropy** di Home Owner come segue:

$$I(\text{Home Owner} = \text{Yes}) = -\frac{0}{3} \cdot \log_2(\frac{0}{3}) - \frac{3}{3} \cdot \log_2(\frac{3}{3}) = 0$$

$$I(\text{Home Owner} = \text{No}) = -\frac{3}{7} \cdot \log_2(\frac{3}{7}) - \frac{4}{7} \cdot \log_2(\frac{4}{7}) = 0.985$$

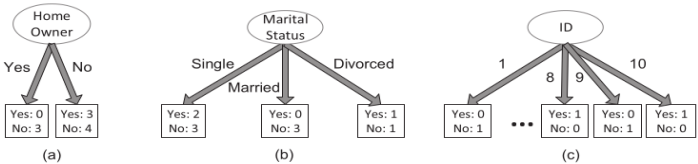


Figure 3.12. Examples of candidate attribute test conditions.

$$I(\text{Home Owner}) = \frac{3}{10} \cdot 0 + \frac{7}{10} \cdot 0.985 = 0.690$$

Ora la calcoliamo anche per Marital Status:

$$I(\text{Marital Status} = \text{Single}) = -\frac{2}{5} \cdot \log_2(\frac{2}{5}) - \frac{3}{5} \cdot \log_2(\frac{3}{5}) = 0.971$$

$$I(\text{Marital Status} = \text{Married}) = -\frac{0}{3} \cdot \log_2(\frac{0}{3}) - \frac{3}{3} \cdot \log_2(\frac{3}{3}) = 0$$

$$I(\text{Marital Status} = \text{Divorced}) = -\frac{1}{2} \cdot \log_2(\frac{1}{2}) - \frac{1}{2} \cdot \log_2(\frac{1}{2}) = 1.000$$

$$I(\text{Marital Status}) = \frac{5}{10} \cdot 0.971 + \frac{3}{10} \cdot 0 + \frac{2}{10} \cdot 1.0 = 0.686$$

Come possiamo vedere, l'impurità di Marital Status è leggermente più alta rispetto a Home Owner.

2.2.7 Trovare la migliore condizione per il test degli attributi

Per determinare la bontà di una condizione di test, dobbiamo confrontare il grado di impurità del nodo genitore (prima della divisione) con il grado pesato di impurità dei nodi figli (dopo la divisione).
Maggiore è la loro differenza, migliore è la condizione del test. Questa differenza, Δ, definita anche come il guadagno (**Gain**) in purezza di una condizione di test, può essere definita come segue:

$$\Delta = I(\text{parent}) - I(\text{children})$$

Maggiore è il guadagno, più pure sono le classi nei nodi figli rispetto al nodo genitore. Il criterio di suddivisione nell'algoritmo di Decision Tree seleziona la condizione di test che mostra il guadagno massimo.

Infine, quando l'entropia è usata come misura di impurità, la differenza di entropia tra genitore e figli è comunemente nota come **Information Gain**, Δ_{info}.

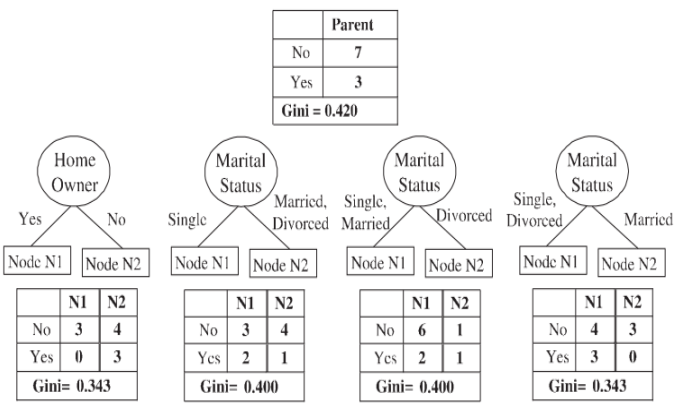


Figure 3.13. Splitting criteria for the loan borrower classification problem using Gini index.

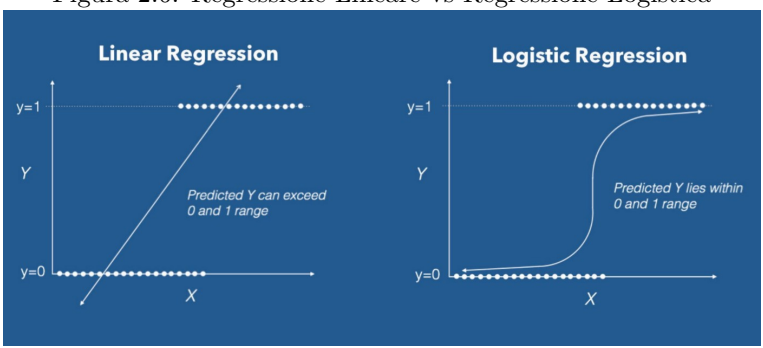
2.3 Logistic Regression

La **Logistic Regression** è una funzione definita come

$$sig(t) = \frac{1}{1 + e^{-t}}$$

e stima la probabilità che si verifichi un certo evento sulla base di un determinato set di dati di variabili indipendenti.

Figura 2.6: Regressione Lineare vs Regressione Logistica



Questo classificatore calcola la retta che separa una classe dall'altra, in uno spazio a due o più dimensioni. Tra le infinite rette che possono dividere due classi viene scelta, generalmente, quella che le divide a distanza equa dalla retta. È possibile, però, scegliere una retta diversa nel caso si voglia essere più restrittivi verso una classe rispetto a un'altra.

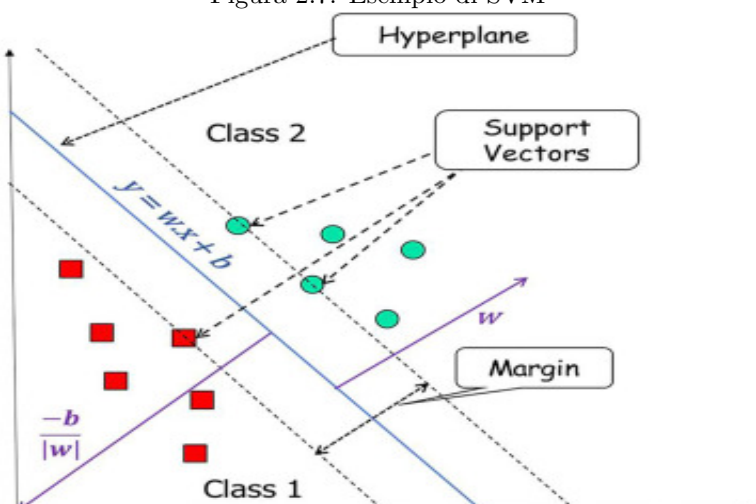
NOTA: Poiché il risultato è una probabilità, la variabile dipendente (Y) è limitata nell'intervallo $[0, 1]$.

2.4 SVM: Support Vector Machine

La **SVM** è un modello di Supervised Learning che utilizza algoritmi di classificazione per problemi di classificazione tra due gruppi.

Rispetto agli algoritmi più recenti (come le reti neurali), presenta due vantaggi principali: maggiore velocità e migliori prestazioni con un numero piccolo di campioni.

Figura 2.7: Esempio di SVM



Una SVM prende i punti del dataset e calcola l'iperpiano che separa meglio i due gruppi.
Questa linea è la **Decision Boundary** (nell'esempio sopra, tutto ciò che cade da un lato sarà classificato come rosso mentre tutto ciò che cade nell'altro sarà verde).

Per la SVM, il miglior iperpiano è quello che massimizza i margini dei vettori di support di ciascuna classe. In altre parole, trova l'iperpiano la cui distanza dell'elemento più vicino (possono essere anche più di uno se sono equamente vicini all'iperpiano) di ciascuna classe è la più grande.

2.4.1 Margini

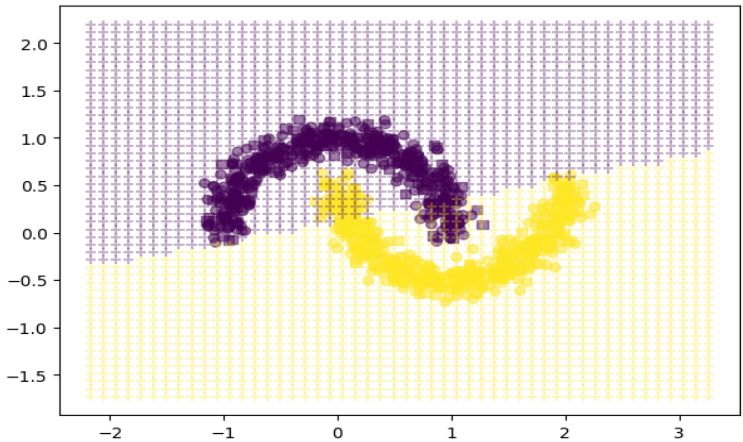
Il **margine** è la distanza che abbiamo tra i punti più vicini al nostro Decision Boundary, ovvero la retta.
Esistono due tipi di margini:

- **Hard Margin** : è il margine che possiamo usare quando riusciamo a separare tutti i punti da una parte e dall'altra, senza mischiarli;
- **Soft Margin** : rilassa i vincoli dell'hard margin, permettendo che alcuni punti che dovrebbero appartenere a una classe di "sorpassare" la retta. Questo permette di avere meno restrizioni nel caso non esista una retta che separe bene i due cluster.

2.4.2 Problemi di separazione non lineare

Immaginiamo di avere un problema del genere:

Figura 2.8: Esempio di problema non lineare



Per questo problema non esiste un iperpiano di due dimensioni che separa i due cluster di dati.
La soluzione a tale problema è quella di aumentare la dimensione dello spazio su cui si sta lavorando per vedere se esiste un iperpiano che separa i due clusters. Dopo aver trasformato lo spazio, si può usare il regressore nel nuovo spazio creato.

Figura 2.9: Aumento di una dimensione dello spazio

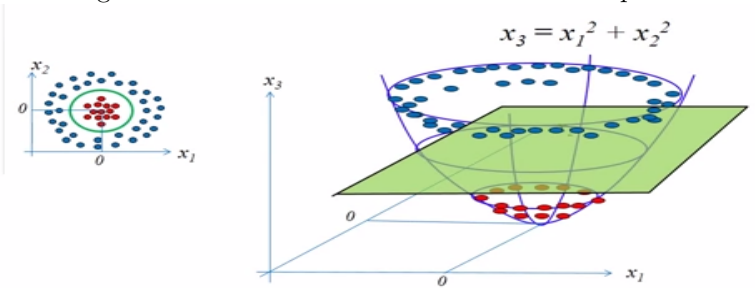
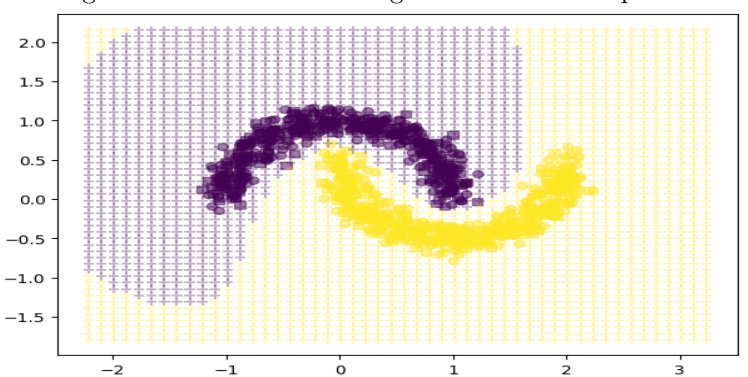


Figura 2.10: Risultato del regressore nel nuovo spazio



2.5 Linear Regression

La **Regressione Lineare** è un modello molto semplice basato sull'equazione della retta e viene utilizzata per studiare la relazione tra due o più variabili.

Nella Regressione Lineare viene utilizzato tipicamente l'errore quadratico medio (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{i=0}^{n}(y_i - \hat{y_i})^2$$

dove n è la dimensione del campione, y_i sono le osservazioni e $\hat{y_i}$ sono i valori stimati dal modello.

Il MSE è una misura di qualità del modello: più bassa è, migliore sarà il nostro modello!

La Regressione Lineare si divide in *semplice* quando viene stiamo un modello della forma $y = \beta_0 + \beta_1 x$ e *multipla* quando stimiamo un modello che utilizza più di un predittore, per esempio $y = \beta_0 + \beta_1 \log(x) + \beta_2 x^2$.

Capitolo 3

Ensemble Methods

3.1 Introduzione

I Ensembler sono algoritmi che costruiscono un insieme di classificatori di base (**Weak Learners**) e li utilizzano per classificare nuovi dati sulla base di un voto (pesato) delle loro previsioni. I metodi di ensemble di solito producono soluzioni più accurate rispetto a un singolo modello.

L'obiettivo degli Ensembler è combinare le previsioni di diversi stimatori di base costruiti con un dato algoritmo di apprendi-

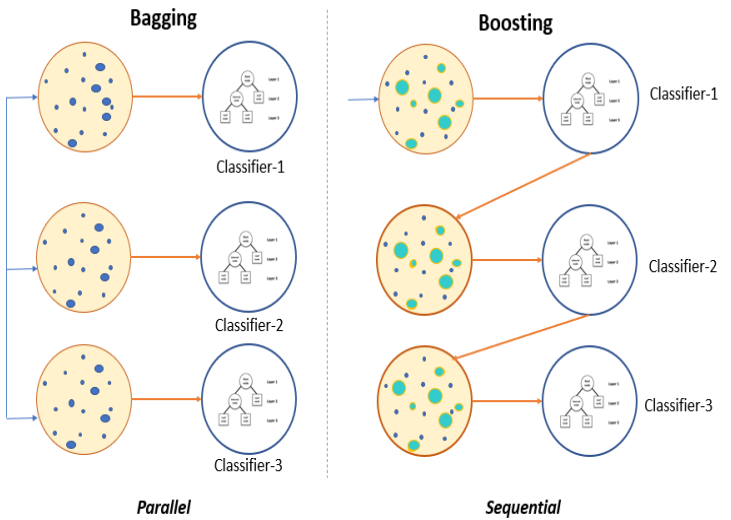
mento (es: Decision Trees) al fine di migliorare la generalizzabilità/robustezza su un singolo stimatore.

Generalmente si distinguono due famiglie di Ensemble Methods:

- **Averaging Methods** (o anche **Bagging Methods**) : l'obiettivo principale è costruire diversi stimatori in modo indipendente e quindi calcolare la media delle loro previsioni. In media, lo stimatore combinato è generalmente migliore di qualsiasi stimatore di base perché la sua varianza è ridotta;

- **Boosting Methods** : stimatori di base vengono costruiti in sequenza e si cerca di ridurre il bias dello stimatore di base successivo, aggiustando i pesi (sulla base dell'errore/precisione) per ogni campione di training in modo che gli stimatori successivi si possano concentrare su istanze erroneamente predette da quelli precedenti.

Figura 3.1: Esempio di Bagging e Boosting



3.2 Bagging vs Boosting

3.2.1 Bagging

Il BAGGing (Boostrap AGGregation) è un Ensemble Method utilizzato per ridurre la varianza all'interno di un dataset con molto rumore.

Nel Bagging, un campione casuale di dati di training viene selezionato con il metodo della sostituzione, ovvero che all'interno di quel campione casuale è possibile selezionare lo stesso dato più di una volta (la probabilità è uniforme). Dopo aver generato diversi campioni, questi Weak Learners vengono allenati parallelamente e indipendentemente l'uno dall'altro per poi utilizzare, a seconda del problema, una media o il voto di maggioranza come stima.

3.2.2 Boosting

Il Boosting è un Ensemble Method utilizzato per ridurre al minimo gli errori durante la fase di training.

Nel Boosting, un campione casuale di dati di training viene selezionato con ripetizione (come nel bagging) e utilizzato per addestrare in sequenza ogni modello: ogni modello cerca di compensare i punti deboli del proprio predecessore. I pesi dei dati vengono ripesati: i dati classificati in modo errato ottengono un peso maggiore mentre quelli classificati correttamente perdono peso. Pertanto,

i Weak Learners successivi si concentreranno maggiormente sui dati che i precedenti hanno classificato erroneamente.

La ricalibrazione del peso di ogni dato indica una ricalibrazione della probabilità con la quale quel dato può essere pescato come campione dal modello successivo: più è alta e più probabilità avrà di essere scelto (solo all'inizio i pesi sono tutti uniformi),

3.2.3 Differenze

Le principali differenze tra questi due metodi sono:

- Il modo in cui vengono allenati: nel bagging, i Weak Learners vengono addestrati in parallelo mentre nel boosting imparano in sequenza;
- Il campionamento dei dati per la fase di addestramento: il bagging prende i dati con la stessa probabilità mentre il boosting li prende in base al peso dato dai modelli precedenti;
- Gli scenari in cui vengono utilizzati: il Bagging funziona bene per Weak Learners con alta varianza e bassa distorsione, mentre il Boosting quando si osserva una bassa varianza e un'elevata distorsione.

3.3 Random Forest

Random Forest (o **Random Decision Forest**) è un Ensemble Method per la classificazione, la regressione e altre attività (es: Stima di Similarità o Feature Selection). Opera costruendo una moltitudine di alberi decisionali durante la fase di training, seguendo l'idea degli algoritmi di Bagging. L'output della foresta dipende dal tipo di attività: se di classificazione sarà la classe selezionata dalla maggioranza degli alberi, se di regressione sarà la predizione media degli alberi.

Le Random Forest correggono l'abitudine degli albero decisionali di adattarsi in modo eccessivo (**overfitting**) al loro set di allenamento.

In particolare, gli alberi che crescono molto in profondità tendono ad apprendere schemi molto irregolari: fanno overfitting sul loro train set, cioè hanno una bassa distorsione ma una varianza molto alta. Le Random Forest sono un modo per utilizzare le predizioni di più alberi decisionali, addestrati su parti diverse dello stesso train set, con l'obiettivo di ridurre la varianza.
Ciò avviene a scapito di un piccolo aumento della distorsione e una certa perdita di interpretabilità, ma, generalmente, aumenta notevolemte le prestazioni del modello finale.

3.3.1 R.F. come Stimatore di Similarità

In definitiva, un modello raggruppa istanze simili e fornisce la stessa previsione per esse.

Un albero decisionale utilizza i predicati dei nodi per identificare un sottoinsieme di istanze per le quali viene fornita la stessa previsione: questo raggruppamento per somiglianza si basa sull'etichetta target piuttosto che sulle features.
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 allenamento.

Possiamo costruire una misura di similarità sulla base della seguente ipotesi: due istanze sono simili se attraversano la foresta casuale lungo percorsi simili, ovvero se cadono sulle stesse foglie.

Misuriamo quindi la somiglianza tra due istanze (o_i e o_j) come il rapporto tra il numero di foglie comuni raggiunte durante l'attraversamento della foresta e il numero di alberi (n) in cui i due dati appaiono nella stessa foglia:

$$RF_{sim}(o_i, o_j) = \frac{1}{n} \cdot \sum_{k=0}^{n} [\text{leaf}_k(o_i) == \text{leaf}_k(o_j)]$$

In linea di principio, questo potrebbe essere applicato a qualsiasi foresta di alberi decisionali, inclusi il bagging e il boosting.
Random Forest è preferibile per i seguenti motivi:

- La R.F. è migliore del Boosting poiché la R.F. dà lo stesso peso agli alberi;
- La R.F. è migliore di Bagging in quanto i suoi alberi sono più diversi.

3.3.2 Identificare gli Outliers con le R.F.

Con un ragionamento simile al concetto di similarità, possiamo usare le Random Forest per identificare gli outliers nel dataset.
Definiamo l'**Outlying Score** come l'inverso della somma dei quadrati delle similarità tra il nodo o_i preso in questione e ogni altro nodo nel dataset:

$$out(o_i) = \left[\sum_{o_j \in \mathcal{D}} RF_{sim}(o_i, o_j)^2 \right]^{-1} ; \qquad \forall o_j \neq o_i$$

3.3.3 Feature Importance & Feature Selection

Importanza: ogni albero della foresta può calcolare l'importanza di una feature in base alla capacità della feature stessa di aumentare la purezza delle foglie.

Maggiore è l'incremento della purezza delle foglie, maggiore è l'importanza della caratteristica. Questo viene fatto per ogni albero, quindi viene calcolata la media tra tutti gli alberi e, infine, normalizzata nell'intervallo $[0, 1]$.

Selezione: Una volta individuata l'importanza di ogni feature, esegue la selezione delle features utilizzando una procedura chiamata **R.F.E** (Recursive Feature Elimination). È comune combinare questa tecnica con la Cross-Validation.

L'idea è quella di rimuovere la feature meno rilevate dopo il fitting del modello, calcolare le performance usando la [k-fold] Cross-Validatio. Questa sequenza di operazioni vengono iterate fino all'esaurimento di tutte le features. Il set di features con il più alto punteggio di performance calcolato durante tutto il procedimento è il set di features ritenuto migliore dalla R.F.

3.4 Naive Bayes

Naive Bayes è una tecnica semplice per costruire classificatori. Non esiste un unico algoritmo per addestrare tali classificatori, ma una famiglia di algoritmi basati su un principio comune: tutti i classificatori *ingenui* (naive) presuppongono che il valore di una particolare feature sia **indipendente** dal valore di qualsiasi altra feature.

Prende il nome di Naive Bayes dal **Teorema di Bayes**:

$$P\left[X \mid Y\right] = \frac{P\left[Y \mid X\right] \cdot P\left[X\right]}{P\left[Y\right]}$$

da cui si può implicare che:

$$P\left[Y \mid X\right] \cdot P\left[X\right] = P\left[X \mid Y\right] \cdot P\left[Y\right] = P\left[X,\ Y\right]$$

In Machine Learning, il problema si traduce in trovare $P\left[y \mid X\right]$ o, nel caso di molteplici classi $C_1, \ldots, C_m$, trovare

$$\arg\max_{C_i} \left[P\left(C_i \mid X\right)\right]$$

Assumendo che ogni feature è condizionalmente indipendente dalle altre:

$$P(C_i|X) = P(x_1|C_i) \cdot P(x_2|C_i) \cdot \ldots P(x_f|C_i)$$

la predizione del modello sarà:

$$\arg\max_{C_i} [P(C_i \mid X)] = P(x_1|C_i) \cdot P(x_2|C_i) \cdot \ldots P(x_f|C_i) \cdot \frac{P(C_i)}{P(X)}$$

Siccome $P(X)$ non cambia il ranking delle classi C_i possiamo anche rimuoverlo dall'equazione, se per scopi di classificazione:

$$\arg\max_{C_i} [P(C_i \mid X)] = P(x_1|C_i) \cdot P(x_2|C_i) \cdot \ldots P(x_f|C_i) \cdot P(C_i)$$

Figura 3.2: Esempio di classificazione

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Outlook	Temp	Humidity	Windy	Play
Rainy	Cool	High	True	?

$$P(Yes \mid X) = P(Rainy \mid Yes) \times P(Cool \mid Yes) \times P(High \mid Yes) \times P(True \mid Yes) \times P(Yes)$$

$$P(Yes \mid X) = 2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14 = 0.00529 \rightarrow 0.2 = \frac{0.00529}{0.02057 + 0.00529}$$

$$P(No \mid X) = P(Rainy \mid No) \times P(Cool \mid No) \times P(High \mid No) \times P(True \mid No) \times P(No)$$

$$P(No \mid X) = 3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = 0.02057 \rightarrow 0.8 = \frac{0.02057}{0.02057 + 0.00529}$$

3.4.1 Zero-Frequency Problem

Nel caso una certa probabilità $P(x_j \mid C_i) = 0$, l'intero prodotto andrebbe anch'esso a zero, pure se i segnali delle altre features sono molto forti!
Per ovviare a questo problema si utilizza la **Correzione di Laplace**

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

dove:

- N_{ij} è il numero di istanze in classe C_i con la j-esima feature uguale a x_j;
- N_i è il numero di istanze in classe C_i;
- v è il numero totale di valori unici della j-esima feature nel dataset $\mathcal{D}$.

Nota: questa correzione deve essere applicata a ogni feature ordinale e non solo a quella per cui accade il problema della frequenza a zero!

Capitolo 4

Feature Engineering

Durante il corso della creazione di un modello, è molto probabile (quasi certo) di incontrare problemi con i dati. Alcune problematiche che potrebbero sorgere sono, per esempio, la gestione delle features binarie, gestione delle features categoriali o anche dei valori mancanti!

Di seguito verranno illustrate alcune strategie per ovviare a questi problemi:

- **Features Categoriche Binarie** : fare un remapping a 0-1 dei valori (es: Alto $\longrightarrow$ 1; Basso $\longrightarrow$ 0);
- **Features Categoriche K-arie** : One-Hot Encoding (discusso in seguito);
- **Etichette di Classe Categoriche K-arie** : fare un remapping a degli ID numerici;
  Attebzione: Questa soluzione va bene per i Decision Trees ma non per kNN e Linear Regression!
- **Features con Valori Singoli** : essendo insignificative alla risoluzione di qualsiasi tipo di problema, le si può tranquillamente togliere;
- **Valori Mancanti** : rimpiazzare con la media se è un valore numerico o con la moda se categoriale.

4.0.1 Problemi con i Dati Categoriali

Alcuni algoritmi possono funzionare direttamente con i dati categoriali (es: Alberi Decisionali), ma molti altri algoritmi non riescono a processarle e richiedono una trasformazione numerica di queste variabili.

Codifica a Numerici Ordinali

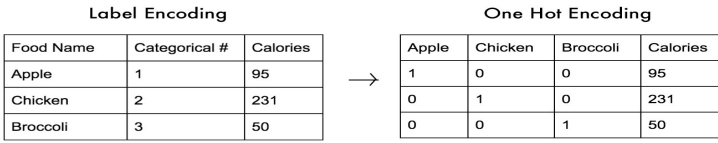
A ogni valori categorico viene assegnato un valori intero (es: "rosso" $\longrightarrow$ 1, "verde" $\longrightarrow$ 2, "blu" $\longrightarrow$ 3).

Bisogna fare attenzione nell'utilizzo di questo metodo perché si va a creare una relazione di ordinamento. Questo può essere d'aiuto se si vuole effettivamente rappresentare un ordinamente, ma nelle variabili categoriali in cui non esiste una relazione d'ordine ciò può portare a risultati imprevisti!
Prendiamo d'esempio la codifica in valori ordinali del colore degli occhi: dire che "rosso" (1) è tre volte più piccolo di "blu" (3) non ha senso in questo caso.

One-Hot Encoding

Questo metodo di codifica consiste nella sostituzione della variabile categoriale con n variabili binarie, una per ogni valore che poteva assumere la precedente feature categoriale.

Figura 4.1: Esempio di One-Hot Encoding



Capitolo 5

Text Processing

Di seguito alcuni metodi per processare dei testi:

- **TF-IDF** (Term Frequency - Inverse Document Frequency): è un algoritmo che utilizza la frequenza delle parole per determinare quanto tali parole siano rilevanti per un dato documento.
  È un approccio relativamente semplice ma intuitivo per dare peso alle parole.

  Per riassumere l'intuizione chiave, l'importanza di un termi-

ne è inversamente correlata alla sua frequenza nei documenti.

TF ci fornisce informazioni sulla frequenza con cui un termine t appare in un documento d e IDF ci fornisce informazioni sulla relativa rarità di un termine t nella raccolta dei documenti D. Moltiplicando questi valori insieme possiamo ottenere il nostro valore TF-IDF finale.

$$tf\ idf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

dove $idf(t, D)$ è definito come:

$$idf(t, D) = \ln\left(\frac{N_{\text{docs}}}{df(t)}\right)$$

in cui N_{docs} rappresenta il numero di documenti nella collezione D mentre $df(t)$ è il numero di documenti contenenti il termine t;

- **Stemming** : viene usato per rimuovere il prefisso/suffisso (es: "being" $\longrightarrow$ "be", "was" $\longrightarrow$ "was");

- **Lemming** : si riferisce all'identificazione dell'origine della parola (es: "being" $\longrightarrow$ "be", "was" $\longrightarrow$ "be").

Capitolo 6

Dimensionality Reduction

6.1 Curse of Dimensionality

La Maledizione della Dimensionalità (**Curse of Dimensionality**) si riferisce a una serie di problemi che sorgono quando si lavora con dati ad alta dimensione.

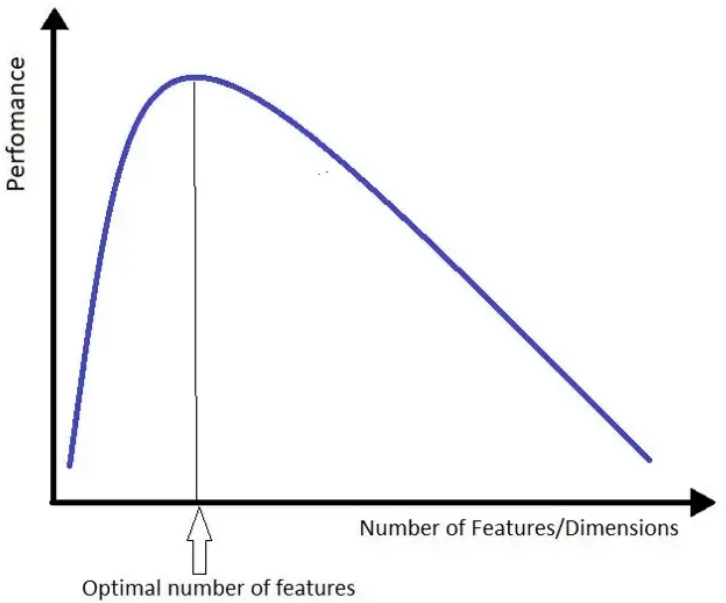
La dimensione di un set di dati corrisponde al numero di attribu-

ti/features che esistono in un set di dati. Un set di dati con un numero elevato di attributi, generalmente dell'ordine di un centinaio o più, viene definito dati ad alta dimensione.

Alcune delle difficoltà che derivano dai dati ad alta dimensione si manifestano durante l'analisi o la visualizzazione dei dati per identificare i modelli, e alcune si manifestano durante l'allenamento dei modelli stessi.

All'aumentare della dimensionalità, il numero di punti richiesti per una buona prestazione di qualsiasi algoritmo aumenta in modo esponenziale. Il motivo è che avremmo bisogno di un numero sempre più grande di punti per una data combinazione di features, affinché qualsiasi modello di apprendimento sia valido.

Figura 6.1: Relazione tra Performance e Dimensionalità



6.2 PCA: Principal Component Analysis

L'analisi delle componenti principali, o **PCA**, è un metodo per la riduzione della dimensionalità che viene spesso utilizzato per ridurre la dimensionalità di set di dati di grandi dimensioni, trasformando un set di variabili di grandi dimensioni in uno più piccolo che contiene ancora la maggior parte delle informazioni nel set di dati originale.

La riduzione del numero di variabili di un set di dati va naturalmente a scapito dell'accuratezza, ma il trucco nella riduzione della dimensionalità è scambiare un po' di accuratezza con la semplicità: proprio perché i set di dati più piccoli sono più facili da esplorare e visualizzare, rendono l'analisi dei dati molto più semplice e veloce per gli algoritmi.

Quindi, per riassumere, l'idea di PCA è semplice: ridurre il numero di variabili di un set di dati, preservando quante più informazioni possibili.

6.2.1 Funzionamento

Nota: non sono state elencate tutte a lezione, ma vengono lo stesso inserite (per completezza).

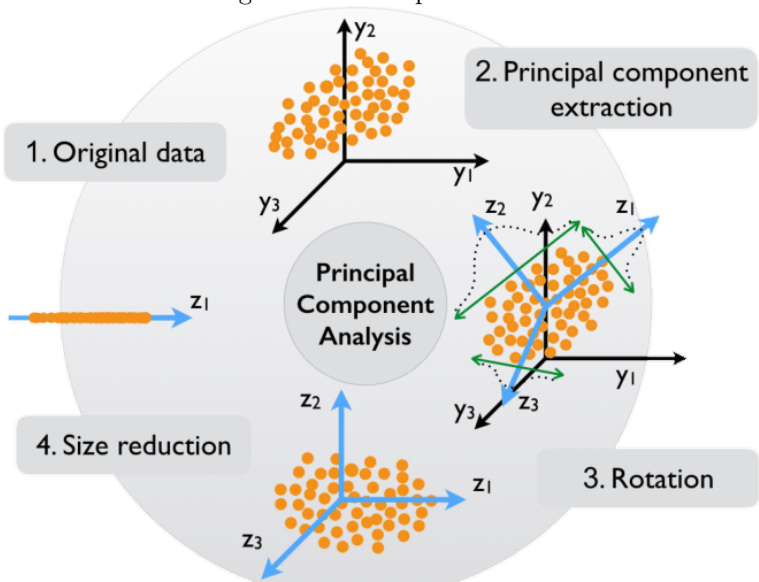
La PCA può essere suddivisa in cinque steps:

1. Standardizzare l'intervallo di variabili continue;
2. Calcolare la matrice di covarianza/correlazione per identificare le correlazioni;
3. Calcolare gli autovettori e gli autovalori della matrice di covarianza per identificare le componenti principali;
4. Crea un vettore di caratteristiche per decidere quali componenti principali mantenere;
5. Esegue un recasting dei dati lungo gli assi delle componenti principali.

PCA trova le direzioni ortogonali della varianza massima dei dati forniti. Può essere pensato come una trasformazione in un nuovo sistema di coordinate, dove la prima coordinata (la prima componente principale) identifica la proiezione di massima varianza, la seconda coordinata la seconda massima varianza, etc...

Matematicamente, i componenti principali sono gli autovettori della matrice di covarianza.

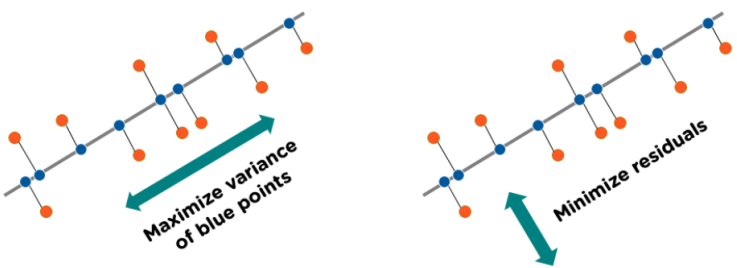
Figura 6.2: Esempio di PCA



6.2.2 Come trova le Componenti Principali?

Poiché ci sono tante componenti principali quante sono le variabili nei dati, le componenti principali sono costruite in modo tale che la prima componente principale rappresenti la massima varianza possibile nel set di dati.

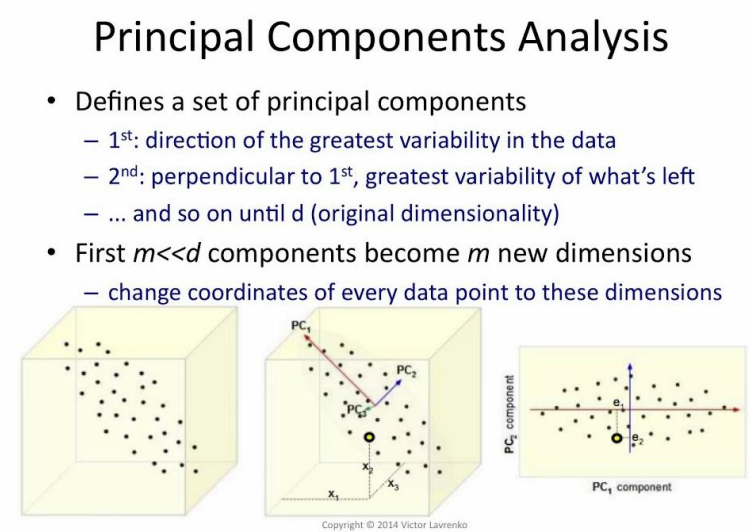
Nota: oltre alla massimizzazione della varianza (nell'esempio sottostante, la distanza fra i punti blu), si cerca anche di minimizzare la perdita di informazione (distanza tra la componente e i punti arancioni).



La seconda componente principale viene calcolata allo stesso modo, con la condizione che sia incorrelata con (cioè perpendicolare a) la prima componente principale e che tenga conto della successiva varianza più alta. Questo continua fino a quando non è stato calcolato un totale di d componenti principali, pari al numero ori-

ginale di variabili.

Alla fine del procedimento, i primi m componenti con $m \leq d$ (d è la dimensione iniziale del dataset) diventeranno le nuove dimensioni del nuovo set di dati.



Capitolo 7

Clustering

7.1 Introduzione

L'analisi dei Clusters si pone l'obiettivo di trovare le similarità tra i dati secondo le caratteristiche trovate nei dati stessi e raggruppare i dati simili tra loro in **clusters**, ovvero delle collezioni di dati.

Questo tipo di problemi fanno parte della seconda famiglia di apprendimento, ovvero **Unsupervised Learning**: non ci sono etichette di classe già predefinite, ma l'algoritmo deve imparare da

solo attraverso l'osservazione e l'analisi dei dati.

Il risultato di un'ottima procedura di clustering prevede come risultato dei clusters che hanno:

- **Alta Similarità "Intra-Class"** : ogni dato è molto simile a tutti gli altri dati del cluster;
- **Bassa Similarità "Inter-Classe"** : ogni dato è molto dissimile da tutti gli altri dati fuori dal cluster.

Tra i vari approcci per la risoluzione dei problemi di clustering, i tre più comuni sono:

- **Partitioning** : vengono costruite varie partizioni e valutate secondo alcuni criteri (es: minimizzazione del Sum of Square Errors);
- **Hierarchical** : viene creata una decomposizione gerarchica del dataset usando alcuni criteri;
- **Density-Based** : approccio basato su funzioni di densità e connettività.

7.2 Partitioning

Il Partitional Clustering scompone un set di dati in un insieme di cluster disgiunti.

Dato un set di dati di n punti, un metodo di partizionamento costruisce k ($k \leq n$) partizioni, con ciascuna partizione che rappresenta un cluster; classifica i dati in k gruppi soddisfacendo i seguenti requisiti:

1. ogni gruppo contiene almeno un punto;
2. ogni punto appartiene esattamente a un gruppo.

Nota : per il partizionamento Fuzzy (spiegato successivamente), un punto può appartenere a più di un gruppo.

Molti algoritmi di clustering partizionale tentano di minimizzare una funzione obiettivo.
Ad esempio, in K-Means e K-Medoidi la funzione (nota anche come **funzione di distorsione**) è:

$$\sum_{i=1}^{k} \sum_{j=1}^{|C_i|} [d(x_j, \text{center}(i))]$$

dove $|C_i|$ è il numero di punti nel cluster i, $d(x_j, \text{center}(i))$ è la distanza tra il punto x_j e il centro i, e k è il numero di clusters.

7.2.1 K-Means

Questo algoritmo di clustering opera in maniera molto semplice:

1. sceglie k centri a caso $c_1, \ldots, c_k$;
2. dati i nuovi centri $c_1, \ldots, c_k$, minimizza la distanza euclidea $\|p - c_i\|^2$, ponendo p nel cluster C_i del centro più vicino c_i;

   Reminder : $\|\overrightarrow{u}\|$ è la norma del vettore u, ovvero la funzione $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ definita come:

   $$\|\overrightarrow{u}\| = \sqrt{u_1^2 + \ldots + u_n^2}$$

3. dati i cluster $C_1, \ldots, C_k$, calcola il nuovo centro c_i di C_i come la media dei punti $p \in C_i$;
4. ripete la procedura dal punto (2) fino a convergenza.

Vantaggi

- **Efficiente** : algoritmo più veloce tra tutti gli algoritmi di clustering;
- Buono per cluster sferici, di dimensioni uguali e distribuiti in modo simile.

Svantaggi

- È necessario specificare il numero di cluster k;
- Non adatto a scoprire clusters con forme non sferiche;
- Sensibile a dati rumorosi e valori anomali: un punto lontano può spingere il centroide (baricentro);
- Applicabile solo agli oggetti in uno spazio n-dimensionale continuo (come calcolare la media di dati categorici?);
- Piccoli cluster potrebbero essere assorbiti da quelli più grandi.

K-Means ++

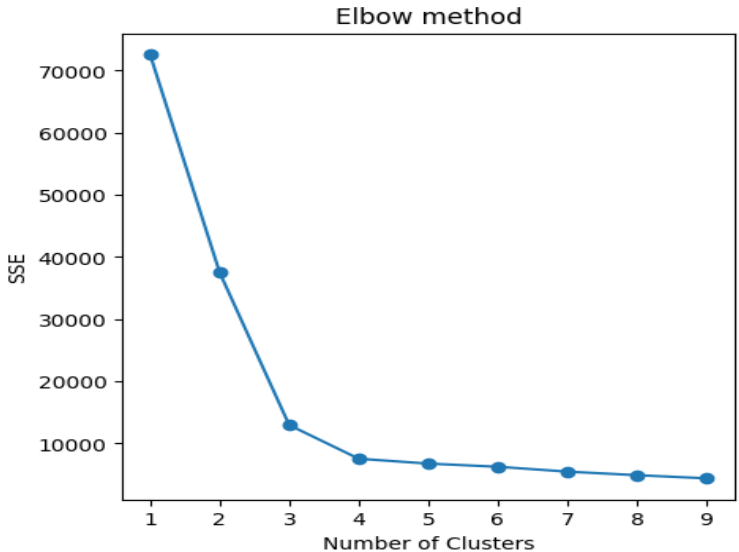
Siccome K-Means è sensibile alla scelta iniziali dei centroidi. L'idea è pensare che un centroide copra una certa area e che, quindi, gli altri centroidi dovranno essere pescati fuori da quell'area.
Il prossimo centroide verrà, dunque, preso lontano dagli altri.
Dopo la selezione iniziale dei centroidi, l'algoritmo procede come il classico K-Means.

Elbow Method

Il Sum of Square Errors (SSE) decresce all'aumentare di k, ma si vorrebbe tenere un k ragionevolmente piccolo.

Una pratica comune, conosciuta come **Elbow Method**, è di fermare l'incremento di k se il vantaggio che si ottiene è piccolo (nell'esempio sottostante, $k = 4$ è un ottimo candidato):

Figura 7.1: Grafico del "Gomito"



7.3 Hierarchical

Il clustering gerarchico è una delle tecniche di clustering popolari in quanto non richiede di sapere già il numero di clusters.

Si divide in due principali metodi di clustering:

- **Agglomerativo** : utilizza una strategia *bottom-up*: inizialmente ogni oggetto forma il proprio cluster, che viene unito iterativamente fino a quando un singolo cluster diventa la radice della gerarchia;
- **Divisivo** : utilizza una strategia *top-down*: inizialmente tutti gli oggetti formano un singolo cluster, che viene ricorsivamente suddiviso in cluster più piccoli.

7.3.1 Algoritmo A.H.C.

Agglomerative Hierarchical Clustering (A.H.C.) è un metodo di clustering il cui principio è semplice:

1. Il processo inizia calcolando la dissomiglianza tra gli n oggetti;

2. Due oggetti che, quando raggruppati insieme, minimizzano un dato criterio di agglomerazione, vengono raggruppati insieme creando così una classe che comprende questi due oggetti;

3. Quindi la dissomiglianza tra questa classe e gli altri $n-2$ oggetti viene calcolata utilizzando il criterio di agglomerazione.
   I due oggetti o classi di oggetti il cui raggruppamento minimizza il criterio di agglomerazione vengono quindi raggruppati insieme.

Questo processo continua finché tutti gli oggetti non sono stati raggruppati.

Queste operazioni di clustering producono un albero di clustering binario, chiamato **dendrogramma**, la cui radice è la classe che contiene tutte le osservazioni. Questo dendrogramma rappresenta una gerarchia di partizioni. È quindi possibile scegliere una partizione troncando l'albero a un dato livello, livello che dipende da

vincoli definiti dall'utente (l'utente sa quante classi devono essere ottenute) o da criteri più oggettivi.

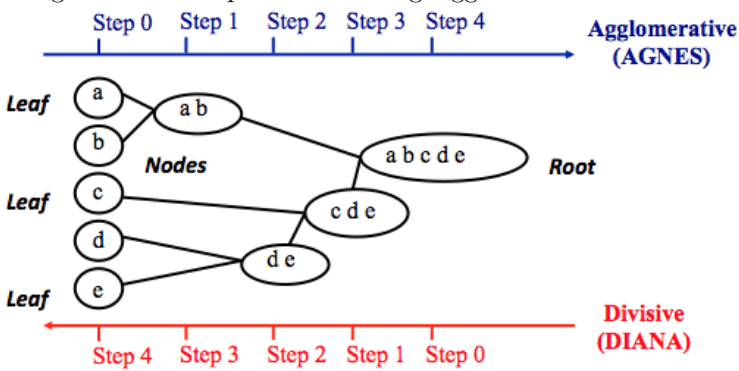
7.3.2 Algoritmo D.H.C.

Divisive Hierarchical Clustering è un approccio di clustering in cui inizialmente tutti i punti nel dataset appartengono a un singolo cluster e la divisione viene eseguita in modo ricorsivo man mano che si scende nella gerarchia.

Gli steps che segue l'algoritmo sono:

1. Inizialmente, tutti i punti nel dataset appartengono a un singolo cluster;
2. Partiziona il cluster in due cluster, il meno simili tra loro;
3. Ad ogni iterazione, il cluster più eterogeneo viene diviso in due cluster;
4. Procede con le iterazioni fino a che tutti gli oggetti sono nel loro cluster.

Figura 7.2: Esempio di Clustering Agglomerativo e Divisivo



7.4 Density-Based

Il Density-Based Clustering funziona rilevando le aree in cui i punti sono concentrati e dove sono separati da aree vuote o sparse. I punti che non fanno parte di un cluster vengono etichettati come rumore.

Funziona bene anche il il clustering temporale: i valori delle feature temporali dei punti possono essere utilizzate per trovare gruppi di punti che si raggruppano insieme nello spazio e nel tempo.

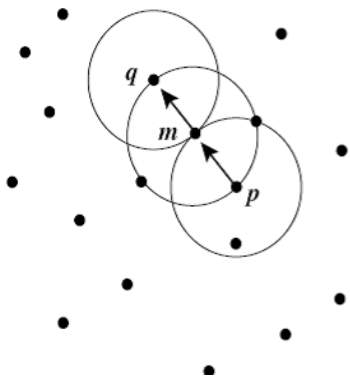
7.4.1 DBSCAN

Il **Density-Based Spatial Clustering of Applications with Noise** (DBSCAN) utilizza una distanza specificata (ϵ) per separare i cluster densi dal rumore più sparso.

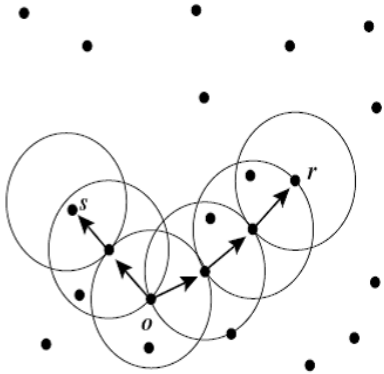
L'algoritmo DBSCAN è il metodo di clustering più veloce, ma è appropriato solo se viene già definita una distanza di ricerca e che funzioni bene per tutti i potenziali cluster. Ciò richiede che tutti i cluster significativi abbiano densità simili.

L'algoritmo utilizza il concetto di raggiungibilità della densità (**density reachability**) e connettività della densità (**density connectivity**):

- **Density Reachability** : si dice che un punto p sia "density reachable" da un punto q se il punto p si trova entro una distanza ϵ dal punto q e q ha un numero sufficiente di punti nei suoi vicini che sono a distanza ϵ.



- **Density Connectivity** : si dice che un punto p e q sono "density connected" se esiste un punto r che ha un numero sufficiente di punti nei suoi vicini ed entrambi i punti p e q sono all'interno della distanza ϵ.
  Per esempio, se q è vicino di r, r è vicino di s, s è vicino di t che a sua volta è vicino di p, allora anche q è vicino di p.



Algoritmo

Siano $X = x_1, x_2, \ldots, x_n$ l'insieme dei punti. DBSCAN richiede due parametri: ϵ e il numero minimo di punti richiesti per formare un cluster ($MinPts$).

Nota: I parametri ϵ e $MinPts$ sono degli iperparametri che devono essere decisi prima del lancio dell'algoritmo.

1. Inizia con un punto di partenza arbitrario che non è stato visitato;

2. Estrae l'intorno di questo punto usando ϵ (tutti i punti che si trovano all'interno della distanza ϵ sono i vicini);

3. Se ci sono abbastanza vicini attorno a questo punto, il processo di raggruppamento inizia e il punto viene contrassegnato come visitato, altrimenti questo punto viene etichettato come rumore;

   Nota: in seguito questo punto può diventare parte del cluster.

4. Se scopre che un punto fa parte del cluster, allora anche il suo intorno ϵ è parte del cluster e gli steps dal punto (2) vengono ripetuti per tutti i punti dell'intorno ϵ.
   Questo viene ripetuto finché non vengono determinati tutti i punti nel cluster;

5. Un nuovo punto non ancora visitato viene preso ed elaborato, portando alla scoperta di un ulteriore cluster o etichettandolo come rumore;

6. Continuo questo processo finché tutti i punti non vengono contrassegnati come visitati.

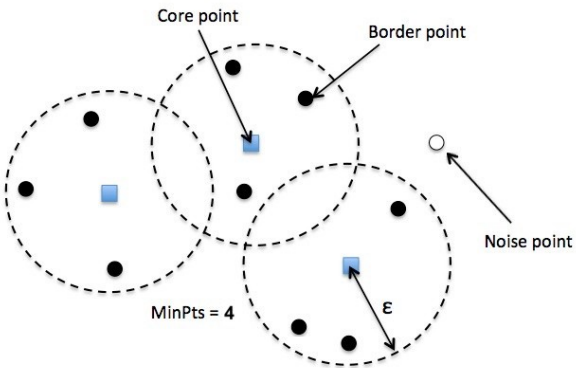
Vantaggi

- Non richiede la specificazione a priori del numero di cluster;
- È in grado di identificare rumore durante il clustering;
- È in grado di trovare cluster di dimensioni e forma arbitrarie.

Svantaggi

- Fallisce in caso di cluster a densità variabile;

Figura 7.3: Esempio di risultato del DBSCAN



Capitolo 8

Regole di Associazione

Association Rules è un metodo per l'identificazione di patterns frequenti, correlazioni, associazioni o strutture causali in set di dati.

Dato un insieme di transazioni, l'obiettivo dell'Association Rule Mining è trovare le regole che ci consentono di prevedere l'occorrenza di un elemento specifico in base alle occorrenze degli altri

elementi nella transazione.

Una regola di associazione è normalmente rappresentata nella forma $\{X\} \Rightarrow \{Y\}$ dove:

- $\{X\}$ è la parte *antecedente* (if), ovvero qualcosa che si trova nei dati;
- $\{Y\}$ è la parte *conseguente* (then), ovvero qualcosa che si trova in congiunzione con la parte antecedente.

Un esempio è la seguente frase: "Se un cliente acquista il pane, ha il 70% di probabilità di acquistare il latte".
Il pane è l'antecedente nella regola associativa, mentre il latte è il conseguente.

8.1 Definizioni

Utilizziamo questa tabella per capire meglio le varie definizioni che verranno a breve illustrate:

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

- **Supporto** : annotato come $S(X)$, è la frequenza di apparizione di X nella tabella. Se applicato a una regola, $S(X \Rightarrow Y)$ è la frequenza di apparizione di $X \cup Y$ e ci indica quante volte appare la regola $X \Rightarrow Y$ nella tabella.

  Per esempio, $S(\{\text{Milk, Bread}\} \Rightarrow \{\text{Diaper}\}) = \frac{\#(\{\text{Milk, Bread}\} \cup \{Diaper\})}{\#\text{transazioni}} = \frac{2}{5} = 40\%$.

- **Confidenza** : annotata come $C(X \Rightarrow Y)$, indica il rapporto tra il numero di volte che appare Y nelle transazioni che

includono anche X e il numero di volte che appare X.

Per esempio, $C(\{\text{Bread}\} \Rightarrow \{\text{Beer}\}) = \frac{2}{4} = 50\%$

8.2 Algoritmi

- **Apriori** : È utilizzato per la generazione degli itemset frequenti, per approssimazioni successive, a partire dagli itemset con un solo elemento.

  In sintesi, il presupposto teorico su cui si basa l'algoritmo parte dalla considerazione che se un insieme di oggetti (itemset) è frequente, allora anche tutti i suoi sottoinsiemi sono frequenti, ma se un itemset non è frequente, allora neanche gli insiemi che lo contengono sono frequenti.

- **FP-Growth** : (FP = Frequent Pattern) È un modo alternativo per trovare set di items frequenti senza utilizzare la generazione di candidati, migliorando così le prestazioni.

  Usa una strategia *divide et impera*: il fulcro di questo metodo è l'utilizzo di una speciale struttura di dati denominata Frequent-Pattern Tree (FP-tree), che conserva le informazioni sull'associazione del set di elementi.

  Usando questa strategia, FP-Growth riduce i costi di ricerca

cercando ricorsivamente pattern brevi e poi concatenandoli in dei pattern frequenti più lunghi.

Figura 8.1: Esempio di Apriori

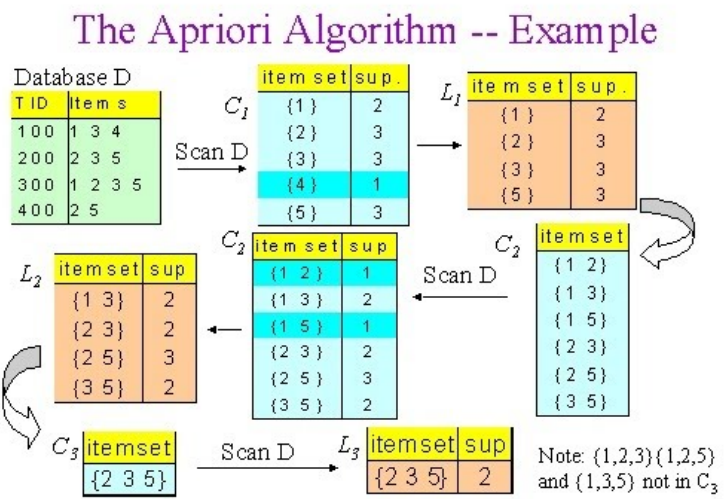
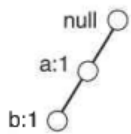
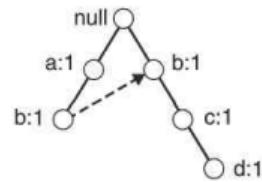


Figura 8.2: Esempio di FP-Growth

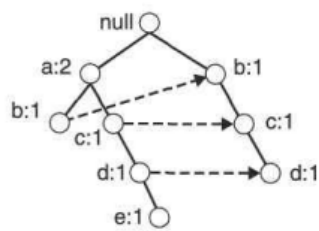
Transaction Data Set

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

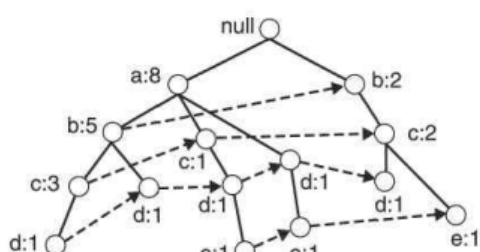




(i) After reading TID=1 (ii) After reading TID=2



(iii) After reading TID=3



(iv) After reading TID=10

Capitolo 9

Recommender Systems

Un **Recommender System** è un sistema di filtraggio delle informazioni che fornisce suggerimenti per gli elementi più pertinenti per un particolare utente (es: film o serie tv suggerite da Netflix).

9.1 Misure di Qualità

Prima di illustrare i sistemi di raccomandazioni, bisogna fare mente locale su cosa ci serve per capire i vantaggi e svantaggi dei vari sistemi.

Per capire ciò, ci si avvale delle *misure di qualità* di un sistema di raccomandazioni. Tali misure sono:

- Efficienza nella costruzione del modello;
- Efficienza nella generazione dei suggerimenti;
- "Serendipity" delle raccomandazioni, ovvero abilità nel suggerire cose non banali e interessanti;
- Adattamento al problema della "partenza a freddo" (Cold-Start Problem).

9.2 Sistemi

Possiamo identificare ben 3 tipi di recommender systems che utilizzano, ognuno, delle strategie diverse.
Prima di tutto, denotando l'insieme degli utenti come U e l'insieme degli items come I, formulare il profilo dell'utente $u \in U$ come:

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

9.2.1 Content-Based

Il filtraggio basato sul contenuto utilizza le features degli items per consigliare altri items simili a quelli che piacciono all'utente, in base alle azioni precedenti o al feedback esplicito.

La misura di similarità che questo sistema utilizza è il coseno tra due vettori:

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

dove:

- Ogni item $x \in I$ è un vettore di dimensione n, con n che rappresenta il numero di parole nel lessico (creatosi dopo lo stemming, lemming, etc...);
- $x[t] = tf(t, x) \cdot idf(t)$;

  Reminder: $\mathbf{tf}(t, x)$ è la frequenza del termine t nell'item x e $idf(t)$ è l'invero della frequenza di t nell'insieme di items I.

Qualità

- Facile da costruire in quanto la costruzione di un vettore per ciascuno item non è tanto costoso;

- Non è efficiente nella generazione dei suggerimenti siccome deve cercare i vicini più "vicini" tra tutti i documenti;
- Poca serendipity proprio perché è legato alla similarità del testo di un item;
- Cold-Start Problem "parziale" in quanto un utente dovrebbe prima "toccare" almeno un item.

9.2.2 User-Based

Il filtraggio basato sull'utente è una tecnica utilizzata per prevedere gli elementi che potrebbero piacere a un utente sulla base delle valutazioni assegnate a tale elemento dagli altri utenti che hanno gusti simili a quelli dell'utente target.

Utilizza una misura di similarità basata su un **ranking score** s_{ui} dove $u \in U$ e $i \in I$ (indica lo score dell'item i per l'utente u):

$$s_{ui} = \frac{\sum_{v \in N(u)} (v[i] - \overline{v}) \cdot \rho(u, v)}{\sum_{v \in N(u)} |\rho(u, v)|}$$

dove:

- $\rho(u, v)$ è il **coefficiente di correlazione di Pearson**, usato per pesare il rating di ogni utente $v \in N(u)$ per la similarità tra v e u;

- $N(u)$ sono i vicini dell'utente u, ovvero un set di utenti simili all'utente target u;

È possibile predire il rating da parte dell'utente u come: $r_{ui} = \overline{u} + s_{ui}$.

Qualità

- Costoso da costruire in quanto il calcolo della similarità per ogni utente è molto costosa ($O(|U|^2)$) e deve essere ricomputata per ogni nuovo rating di un utente;
- Efficiente nella generazione dei suggerimenti se la ricerca dei "vicini" viene precomputata offline;
- Ottima serendipity;
- Il Cold-Start Problem è molto problematico perché ha bisogno di un set di ratings sufficientemente grande per ogni nuovo utente/item.

9.2.3 Item-based

Il filtraggio basato sull'elemento cerca elementi simili in base agli elementi che gli utenti hanno precedentemente apprezzato o con cui hanno precedentemente interagito positivamente e li consiglia di conseguenza.

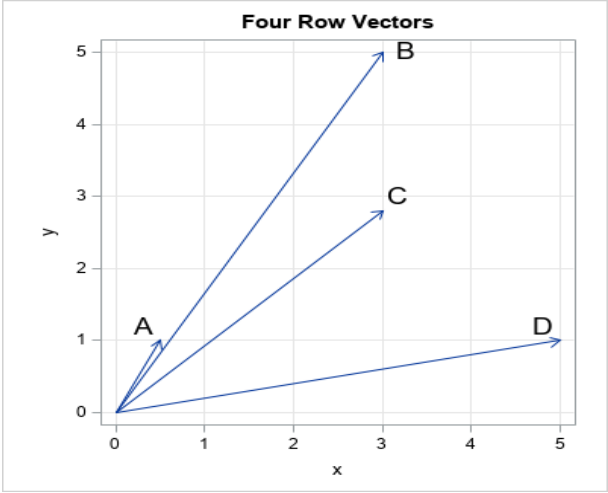
Utilizza la misura di similarità del coseno, ma aggiustata in modo

tale che la misura sia diversa se il set di ratings è diverso

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

Nota: il coseno "normale" non è influenzato dalla scala degli elementi/utenti, quindi è come se fossero tutti sulla stessa scala.

Figura 9.1: A e B sono molto simili ma A è valutato peggio!



Qualità

- Efficiente da costruire in quanto le computazioni avvengono offline e, generalmente, $|I|$ e più piccolo di $|U|$;
- Efficiente nella generazione dei suggerimenti visto che la ricerca dei vicini più vicini non è necessaria se precomputata offline;
- Meno di quella dello User-Based, siccome il ranking finale dipende dalle ratings dell'utente corrente;
- Il Cold-Start Problem è molto problematico perché ha bisogno di un set di ratings sufficientemente grande per ogni nuovo utente/item.

Capitolo 10

Shingling & LSH

Premessa

Usare il classico algoritmo di "string matching" per determinare la similarità fra 2 documenti e' decisamente costoso computazionalmente. Esistono quindi altri metodi di rappresentare un documento che permettono di calcolare la similarità in modi diversi.

10.1 k-shingles

E' un metodo per la rappresentazione di un documento come un insieme di k-grammi, ovvero sottostringhe di lunghezza k.

Costruiti i sottoset di k-grammi, si può calcolare la similarità tra due documenti tramite l'indice di Jaccard, ovvero la cardinalità dell'intersezione tra i due insiemi divisa per la cardinalità dell'unione.

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Tramite questa reinterpretazione, e' possibile avvalorale la similarità fra parole, in modo che pure parole che non sono perfettamente uguali (ad esempio "player" e "players") possano essere considerate simili.

10.1.1 Come scegliere k

La metrica k deve essere scelta in base al documento che si vuole rappresentare. Minore e' k e maggiore e' la probabilita' che un particolare k-gramma sia presente inoltre cresce la probabilita' che due documenti siano simili.

E' importante ricordare che il numero di k-grammi dipende da k e dalla lunghezza dell'alfabeto usato e cresce esponenzialmente.

Ad esempio, usando un alfabeto generico di 27 caratteri (26 + spazio), il numero di k-grammi e' 27^k.

10.2 Min-Hashing

Sebbene i k-grammi forniscano maggiore qualità rispetto al classico "string matching", si tratta di una metrica ancora troppo costosa.

Un metodo abbastanza potente per ridurre notevolmente la dimensione della rappresentazione di un documento è l'uso di un algoritmo di hashing.

Si consideri un documento A come l'insieme dei suoi k-grammi. Il **min-hashing** di A si definisce come l'elemento più piccolo di A dopo l'applicazione di una permutazione casuale π. Pertanto il min-hashing di A è dato da un elemento del set scelto casualmente da π.

Applicando la stessa permutazione ad ogni altro documento si ottengono i min-hashing di tutti i documenti.

Osservazione 10.2.1. *I min-hash dei documenti A e B si denotano come $min(\pi_A)$ e $min(\pi_B)$.*

Esempio 1.

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

π	A	B
c	0	0
d	1	1
b	0	1
e	0	1
a	1	0

La prima tabella rappresenta il documento A e il documento B come insiemi di k-grammi. La seconda tabella A e B dopo l'applicazione della permutazione π.

Per calcolare il min-hash dei due documenti è sufficiente scorrere la matrice dall'alto verso il basso seguendo l'ordine della permutazione π e prendere la prima riga che contiene un 1.

Si nota poi con x il numero di righe in cui entrambi i documenti presentano un 1, mentre con y il numero di righe in cui *esattamente uno dei due documenti* presenta un 1. Si scartano quindi le righe in cui entrambi i documenti presentano un 0.

Pertanto la probabilità che i due documenti siano simili e che pertanto il loro min-hash sia il medesimo è data dalla proba-

bilità che la prima riga incontrata sia una con 1 in entrambi i documenti.

$$p = \frac{x}{x+y}$$

Osservazione 10.2.2. *Non è strettamente necessario eseguire l'intera permutazione, bensì è sufficiente generare una scelta casuale per la prima riga, scegliendola da quelle contate da x o da y.*

10.2.1 Min-hash Signatures

Purtroppo il metodo di min-hash non è così affidabile. Infatti sebbene possa dire che due documenti abbiano una certa probabilità di essere simili, non è in grado di dire quanto siano simili.

Per superare questa limitazione, si possono usare più permutazioni, quindi calcolare più min-hashes. Supponendo di avere un set di permutazioni

$$\pi_1, \pi_2, \ldots, \pi_n$$

si può calcolare la similarità costruendo il vettore

$$\left[\min(\pi_A^1), \min(\pi_A^2), \ldots, \min(\pi_A^n)\right]$$

per ogni documento. Ogni vettore che rappresenta un documento è detto **signature**. Questo tipo di "firma" può essere utilizzato per fornire una stima della similarità tra due documenti.

Prendendo k il numero di min-hash uguali, calcolate da m permutazioni diverse per A e B, allora l'indice di similarità è dato da

$$\text{Jaccard}(A, B) \approx \frac{k}{m}$$

Ovviamente il tradeoff è che più permutazioni si usano e più la similarità è precisa, ma più è costoso il calcolo. In generale si vuole mantenere m inferiore alla media di termini contenuti in un documento.

10.3 Hashing con sensibilità locale (LSH)

Pure tramite il min-hashing il costo computazionale può risultare troppo elevato. È necessario un metodo che permetta di eseguire una ricerca più rapida, mantenendo tuttavia un'alta probabilità che se due documenti sono simili, allora il loro min-hash sia lo stesso.

Per farlo si può suddividere il set di min-hash m in b blocchi di $r = m/b$ elementi ciascuno. Ogni blocco è quindi un vettore di r elementi. Ognuno di essi viene poi concatenato e nuovamente hashato, in modo da ottenere un nuovo vettore detto "super-signature" o "super-shingle"

In questo modo si salvano i documenti in hash-tables basate sulle super-signatures. Inoltre si utilizzano b tabelle diverse, in modo da mantenere indipendentemente le b super-signatures.

Il funzionamento dell'algoritmo è il seguente:

1. Si calcolano le min-hash di un documento A
2. Si calcolano le b super-signatures di A
3. Si usano le super-signatures per accedere alle hash-tables
4. Si recuperano i documenti associati a tali hash-tables
5. Si ritorna l'unione di tutti i documenti recuperati

Addizionalmente si può in seguito calcolare la similarità fra A e i documenti con l'indice di Jaccard, il quale sebbene sia costoso, verrebbe applicato ad un sotto-set ristretto dei documenti, in modo da filtrare eventuali "falsi positivi".

10.3.1 Similarity Hashing

Si consideri la seguente immagine, nella quale si vedono due documenti A e B come un vettore bidimensionale, dove i due vettori sono separati da un certo angolo θ.

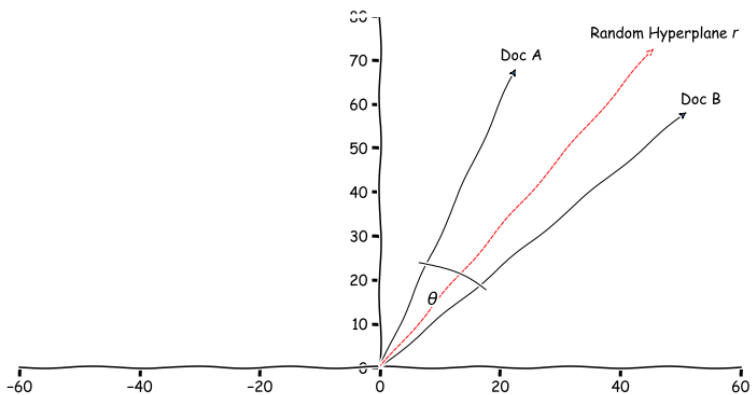


Figura 10.1: Similarity Hashing

Preso un vettore casuale r (un iperpiano se fossimo in uno spazio a più dimensioni), qual è la probabilità che A e B siano separati da r?

Ponendo che l'angolo di r può essere compreso fra $[0, 180]$ gradi, allora

- La probabilità che r separi A e B è data da $\frac{\theta}{180}$
- Al contrario, la probabilità che r non separi A e B è data da $\frac{180-\theta}{180}$

Osservazione 10.3.1. *In pratica l'iperpiano r rappresenta una firma composta da un singolo bit per un dato documento. Se il documento è al di sopra dall'iperpiano, allora il bit è* 1*, altrimenti è* 0*.*

Pertanto la probabilità che due documenti abbiano la stessa firma (quindi siano dallo stesso lato rispetto ad r) è data da $\frac{\theta}{180}$

Osservazione 10.3.2. *Dato un documento A si può usare la seguente strategia*

1. *Si genera r come una sequenza casuale di $+1$ e -1*
2. *Si computa il prodotto scalare $r \cdot A$*
3. *Se il risultato è positivo, allora si aggiunge un* 1 *alla firma, altrimenti un* 0
4. *Si ripete il processo m volte, ottenendo così una firma di m bit*
5. *Infine si può sfruttare la distanza di Hamming per calcolare la similarità fra due documenti*