

# Data & Web Mining

Domenico Sosta

Anno accademico: 2023/2024

## Contents

<b>0 Note al Lettore</b>	<b>5</b>
0.1 Warning . . . . .	5
0.2 Obiettivo . . . . .	5
<b>1 Concetti Base</b>	<b>6</b>
1.1 Supervised Learning Vs Unsupervised Learning . . . . .	6
<b>2 Supervised Learnig</b>	<b>7</b>
2.1 What is a Model? . . . . .	7
2.2 General Framework for Classification . . . . .	7
2.2.1 Cosa è la Classificazione? . . . . .	7
2.2.2 Il Modello . . . . .	7
2.2.3 Terminologia . . . . .	8
2.3 Stratified Sampling . . . . .	8
<b>3 Valutazione dei Modelli</b>	<b>9</b>
3.1 Misurare le Prestazioni dei Modelli . . . . .	9
3.1.1 L'Accuracy . . . . .	9
3.1.2 Altre Metriche di Valutazione . . . . .	9
3.1.3 La ROC Curve . . . . .	10
3.2 Tabella di contingenza . . . . .	10
3.2.1 Valutazione Multi-Classe & Confusion Matrix . . . . .	10
3.3 Il Set di Validazione . . . . .	11
3.4 Strategie di Valutazione del Modello . . . . .	11
3.4.1 K-Fold Cross Validation . . . . .	12
3.4.2 Nested K-Fold Cross Validation . . . . .	12
3.5 Model Selection and Overfitting . . . . .	13
<b>4 Classification and Regression Algorithm</b>	<b>14</b>
4.1 KNN - Nearest-Neighbour Classifier . . . . .	14
4.1.1 L'idea . . . . .	14
4.1.2 Considerazione sul valore di k . . . . .	14
4.1.3 La predizione . . . . .	14
4.1.4 Le metriche . . . . .	14
4.1.5 Altre Metriche . . . . .	15
4.2 Decision Tree Classifier . . . . .	15
4.2.1 L'idea . . . . .	15
4.2.2 Struttura di un albero di decisione . . . . .	15
4.2.3 Algoritmo - Stile Ricorsivo . . . . .	15
4.2.4 Scelta del test condition . . . . .	16
4.2.5 Algoritmo & Stopping Criterio . . . . .	16
4.2.6 Pruning to avoid Overfitting . . . . .	17
4.2.7 Scelta del modello . . . . .	17
4.2.8 Small Tree & Fully Grown Tree . . . . .	18

4.3	Linear Regression . . . . .	19
4.4	Logistic Regression . . . . .	19
4.5	SVM - Support Vector Machines . . . . .	20
4.5.1	L'idea . . . . .	20
4.5.2	Problemi Multi-Classe . . . . .	20
4.5.3	Obiettivo . . . . .	21
4.5.4	Problemi non linearmente separabili - Kernel Trick . . . . .	22
<b>5</b>	<b>Ensemble Methods</b>	<b>23</b>
5.1	Metodi per costruire classificatori ensemble . . . . .	23
5.1.1	Bias-Variances Decomposition . . . . .	23
5.2	Bagging . . . . .	23
5.2.1	Algoritmo . . . . .	23
5.2.2	Predizione . . . . .	24
5.2.3	Conclusione . . . . .	24
5.3	Boosting . . . . .	24
5.3.1	Algoritmo . . . . .	24
5.3.2	Predizione . . . . .	25
5.3.3	Conclusione . . . . .	25
5.4	Random Forest . . . . .	26
5.4.1	Algoritmo . . . . .	26
5.4.2	Predizione . . . . .	26
5.4.3	Conclusione . . . . .	26
5.4.4	Random Forest Similarity . . . . .	27
<b>6</b>	<b>Feature Analysis</b>	<b>28</b>
6.1	Importanza delle Feature e Selezione Futura . . . . .	28
6.2	Feature Engineering . . . . .	28
6.3	Feature Categoriali . . . . .	28
6.3.1	One Hot-Encoding . . . . .	28
6.4	Feature "Unique" . . . . .	30
6.5	Missing Values . . . . .	30
<b>7</b>	<b>Text-Processing</b>	<b>31</b>
7.1	TD-IDF - Term Frequency - Inverse Document Frequency . . . . .	31
7.2	Stemming & Lemming . . . . .	31
7.2.1	Stemming . . . . .	32
7.2.2	Lemming . . . . .	32
7.3	Caso Applicativo . . . . .	32
7.4	Naive Bayes . . . . .	32
7.4.1	Come Calcolare: $P(x_j C_i)$ . . . . .	33
7.4.2	Zero-Frequency Problem . . . . .	33
7.4.3	In Conclusione . . . . .	33
<b>8</b>	<b>ANN - Artificial Neural Network</b>	<b>34</b>
8.1	Origini . . . . .	34
8.2	Struttura . . . . .	34
8.3	In Sintesi . . . . .	34
8.4	Perceptron . . . . .	35
8.4.1	Struttura di un Perceptron . . . . .	35
8.4.2	Apprendimento del Perceptron . . . . .	36
8.4.3	Limiti . . . . .	36
8.5	Multilayer Neural Network . . . . .	37
8.5.1	Struttura di una Multilayer Neural Network . . . . .	37
8.5.2	Il Processo di Apprendimento . . . . .	38
8.6	Ottimizzatore . . . . .	39
8.6.1	Learning Rate . . . . .	39

8.7	Loss Function . . . . .	39
8.8	Overfitting & Best Practice . . . . .	39
8.8.1	Dropout . . . . .	40
8.8.2	Transfer Learning . . . . .	40
<b>9</b>	<b>CNN - Convolutional Neural Network</b>	<b>41</b>
9.1	Calcolo Matrice Di Convoluzione . . . . .	41
9.2	Poolling . . . . .	42
9.3	Proprietà . . . . .	43
9.4	1-D Convolution . . . . .	43
9.5	Funnel or Information Compression . . . . .	43
<b>10</b>	<b>Data Mining Methodology</b>	<b>44</b>
10.1	Metodologia . . . . .	44
10.2	Fasi della Metodologia . . . . .	44
10.3	Struttura dell'Istanza . . . . .	44
10.4	Label . . . . .	45
10.5	Valutazione . . . . .	45
10.6	Algoritmi . . . . .	45
<b>11</b>	<b>Unsupervised Learning</b>	<b>46</b>
11.1	Cluster Analysis . . . . .	46
11.1.1	Approcci nel Clustering . . . . .	46
11.2	Associative Rules . . . . .	46
<b>12</b>	<b>Clustering Algorithm</b>	<b>47</b>
12.1	Cosa si intende con Algoritmo di Clustering . . . . .	47
12.1.1	Proprietà desiderate di un algoritmo di Clustering . . . . .	47
12.1.2	Classificazione Algoritmi di Clustering . . . . .	47
12.2	Partitioning Clustering . . . . .	48
12.2.1	K-Means . . . . .	48
12.2.2	K-Medoids . . . . .	49
12.2.3	Fuzzy C-Means & Partizionamento Parziale . . . . .	50
12.3	Hierarchical Clustering . . . . .	52
12.3.1	Stimare la Similarità tra Cluster - Misure di Linkage . . . . .	52
12.3.2	AHC - Agglomerative Hierarchical Clustering . . . . .	53
12.3.3	DHC - Divisive Hierarchical Clustering . . . . .	54
12.4	Clustering basato sulla Densità . . . . .	54
12.4.1	Core Point " $p$ " e Vicinato di un Punto " $N_\epsilon(p)$ " . . . . .	54
12.4.2	Condizione per la Densità Reachability e Density Connectivity . . . . .	54
12.4.3	DB-Scan . . . . .	55
12.5	Model-Based Clustering . . . . .	56
12.5.1	Self-Organizing Map (SOM) . . . . .	56
12.6	Clustering Evaluation . . . . .	57
12.6.1	Valutazione Intrinseca . . . . .	57
12.6.2	Valutazione Estrinseca . . . . .	58
<b>13</b>	<b>Associative Rules</b>	<b>59</b>
13.1	Definizioni . . . . .	59
13.2	Obiettivo delle Associative Rules . . . . .	59
13.2.1	Approccio Brute-Force . . . . .	59
13.3	Apriori . . . . .	60
13.3.1	Apriori Principle . . . . .	60
13.3.2	Strategie . . . . .	61
13.3.3	Algoritmo . . . . .	61
13.4	FP-Growth . . . . .	62
13.4.1	Algoritmo . . . . .	63

<b>14 Recommender System</b>	<b>64</b>
14.1 Utenti, items e modellazione . . . . .	64
14.2 Content-Based . . . . .	64
14.2.1 Algoritmo . . . . .	64
14.2.2 Misure utilizzate . . . . .	64
14.2.3 Valutazione . . . . .	65
14.3 User-Based . . . . .	65
14.3.1 Algoritmo . . . . .	65
14.3.2 Misure utilizzate . . . . .	65
14.3.3 Valutazione . . . . .	66
14.4 Item-Based . . . . .	66
14.4.1 Algoritmo . . . . .	66
14.4.2 Misure utilizzate . . . . .	66
14.4.3 Valutazione . . . . .	66
14.5 Tabella di Confronto . . . . .	67
<b>15 Crediti</b>	<b>68</b>

# **0 Note al Lettore**

Questo documento è una raccolta di dispense per il corso di Data & Web Mining, tenuto dal Prof. Claudio Lucchese. Questa raccolta rappresenta una trascrizione ordinata degli appunti e degli approfondimenti che ho raccolto durante lo studio per l'esame del corso. È importante sottolineare che queste dispense non sostituiscono in alcun modo le lezioni del docente. Gli studenti sono invitati ad utilizzare queste dispense come supporto aggiuntivo per comprendere meglio gli argomenti trattati durante il corso e per approfondire ulteriormente la materia.

## **0.1 Warning**

La presente versione 1.0.1, non è stata supervisionata dal Prof. Lucchese, pertanto non mi assumo alcuna responsabilità per eventuali errori contenuti nel testo.

## **0.2 Obiettivo**

L'obiettivo iniziale di questo elaborato era quello di ordinare gli appunti per un corso che mi ha dato tante soddisfazioni quanto conoscenze. Successivamente, si è evoluto nella volontà di aiutare tutti coloro che, come me, non sapevano da dove iniziare per affrontare questo corso, tanto interessante quanto impegnativo. Questo elaborato potrebbe rappresentare un punto di riferimento per chiunque si trovi ad affrontare lo stesso percorso.

# 1 Concetti Base

## 1.1 Supervised Learning Vs Unsupervised Learning

Gli algoritmi di apprendimento supervisionato utilizzano dati etichettati, ovvero dati che includono sia le caratteristiche (input) che le etichette (output) per addestrare il modello. Il modello quindi utilizza questi dati per "imparare" a fare previsioni (creare una funzione di mapping) su dati nuovi e non visti. Gli esempi di algoritmi di apprendimento supervisionato sono Regressione lineare, SVM, Random Forest, Naive Bayes, Neural Network e molti altri.

Al contrario, gli algoritmi di apprendimento non supervisionato utilizzano dati non etichettati, ovvero dati che includono solo le caratteristiche (input) e non le etichette (output). Il modello utilizza questi dati per "scoprire" relazioni o strutture nascoste all'interno dei dati. Gli esempi di algoritmi di apprendimento non supervisionato sono Clustering, PCA, Apriori, K-Means e molti altri. In generale, gli algoritmi di apprendimento supervisionato sono utilizzati per problemi di classificazione e regressione, mentre gli algoritmi di apprendimento non supervisionato sono utilizzati per problemi di clustering e riduzione della dimensionalità.

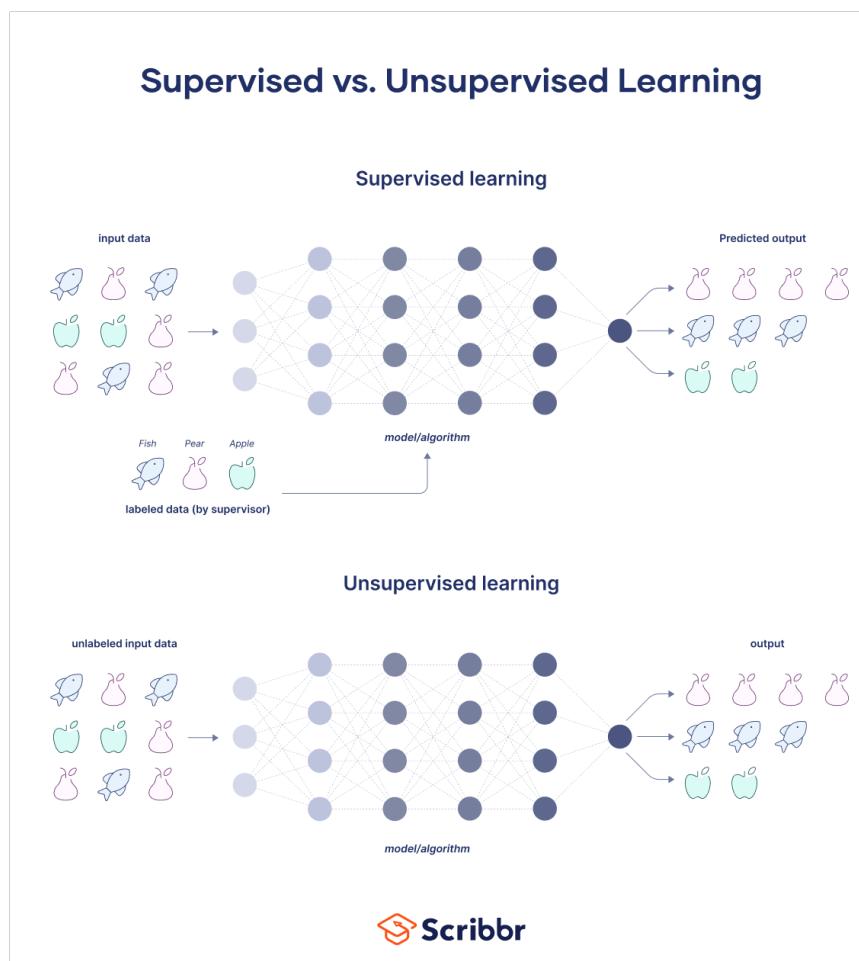


Figure 1: Supervised Learning Vs Unsupervised Learning, Crediti: Fonte

## 2 Supervised Learning

Gli esseri umani hanno la capacità innata di classificare le cose in categorie. L'idea generale dietro al compito di classificazione è quella di costruire un modello che possa categorizzare automaticamente i dati di input in classi o categorie predefinite.

I dati che utilizziamo per un compito di classificazione consistono in una collezione di istanze (record), ciascuna caratterizzata dalla tupla  $(\mathbf{x}, y)$  dove  $\mathbf{x}$  è l'insieme di attributi che descrivono l'istanza e  $y$  è l'etichetta di classe dell'istanza.

### 2.1 What is a Model?

Un modello di classificazione è una rappresentazione astratta della relazione tra l'insieme di attributi e l'etichetta di classe. Il modello può essere rappresentato in molte forme, ad esempio come un albero, una tabella di probabilità o un vettore di parametri reali, ma matematicamente possiamo rappresentare il modello come una **funzione** target  $f$ , che prende in input l'insieme di attributi  $x$  e produce un output corrispondente all'etichetta di classe predetta.

### 2.2 General Framework for Classification

#### 2.2.1 Cosa è la Classificazione?

La classificazione è il compito di assegnare etichette a istanze di dati prive di etichette, e un classificatore è utilizzato per svolgere tale compito. Un classificatore è tipicamente descritto in termini di un modello.

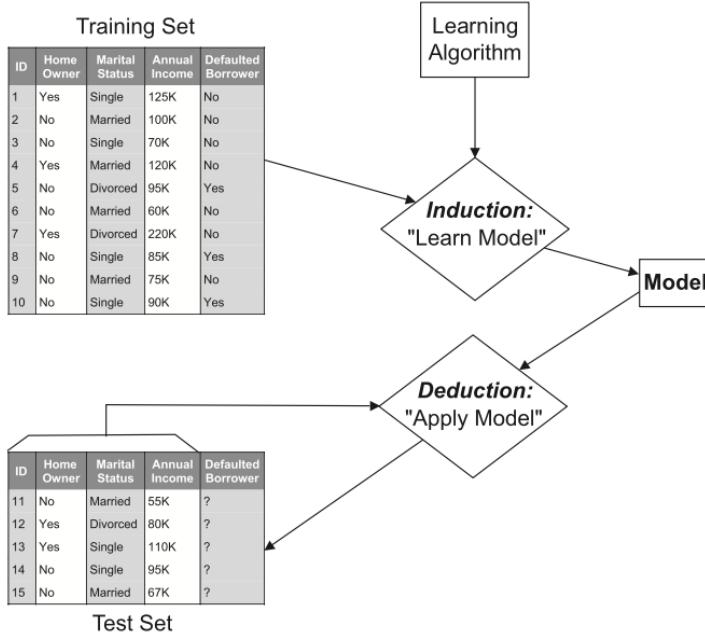
#### 2.2.2 Il Modello

Il modello viene creato utilizzando un dato insieme di istanze  $(\mathbf{x}, y)$ , noto come **set di addestramento**. Il processo di utilizzo di un **algoritmo di apprendimento** per costruire un modello di classificazione dal set di addestramento è noto come **induzione**, ma è spesso descritto anche come "apprendimento del modello" o "costruzione di un modello".

Il processo di applicare il modello su un **set di test**, che è un dato insieme di istanze non viste  $(\mathbf{x}, ?)$ , per prevedere le loro etichette di classe è noto come **deduzione**.

Il modello risultante è ciò che definisce il classificatore. Il classificatore è una manifestazione della capacità del modello di assegnare etichette di classe a punti dati in ingresso basandosi sui pattern e sulle relazioni apprese dai dati di addestramento. Il processo di classificazione coinvolge due passaggi:  
1) Applicare un algoritmo di apprendimento ai dati di addestramento per apprendere un modello  
2) Applicare il modello per assegnare etichette a istanze prive di etichetta.

**Una tecnica di classificazione** si riferisce a un approccio generale alla classificazione, ad esempio la tecnica dell'albero decisionale.



**Figure 3.3.** General framework for building a classification model.

Figure 2: General framework for building a classification model. Crediti: Tan, P.-N., Steinbach, M., & Kumar, V. (Year). Introduction to Data Mining (2nd ed.). Publisher.

### 2.2.3 Terminologia

I termini "classificatore" e "modello" sono spesso considerati sinonimi. Se una tecnica di classificazione costruisce un singolo modello globale, allora va bene. Tuttavia, ogni modello definisce un classificatore, ma non ogni classificatore è definito da un singolo modello. Alcuni classificatori, come i classificatori k-nn (k-nearest neighbor), non costruiscono un modello esplicito ma effettuano previsioni basate sulla somiglianza con i dati circostanti. Il termine "classificatore" è spesso usato in un senso più generale per fare riferimento a una tecnica di classificazione. Quindi, ad esempio, "classificazione a albero decisionale" può riferirsi alla tecnica di classificazione a albero decisionale o a un classificatore specifico costruito utilizzando tale tecnica.

Nel quadro generale mostrato nella Figura 3.3, la fase di induzione e deduzione dovrebbe essere eseguita separatamente. Il set di addestramento e il set di test dovrebbero essere indipendenti.

## 2.3 Stratified Sampling

Quando dividiamo in train e test, abbiamo la necessità di effettuare un campionamento casuale sul dataset. Se il dataset è sufficientemente grande, un campionamento casuale può andar bene, ma prendendo in esame un dataset con tre tipi di label: A, B, C, dove A compare con frequenza 90 %, B compare con frequenza 8% e C con frequenza 2%; in questo caso effettuare un campionamento casuale non è la scelta più intelligente, in quanto si rischia di introdurre una distorsione causata dall'errore di campionamento.

La stratificazione è la strategia utilizzata per ovviare al problema, ovvero:

- Divido il dataset in più strati, tipicamente tanti quante sono le classi da predire. - Per arrivare al 60% di istanze di training seleziono la stessa percentuale di istanze da ogni strato.

In questo modo mi assicuro di avere all'interno del mio training una buona rappresentazione per ogni classe.

Di default *Sklearn* utilizza stratified sampling per il *train\_test\_split* e per fare la cross-validation.

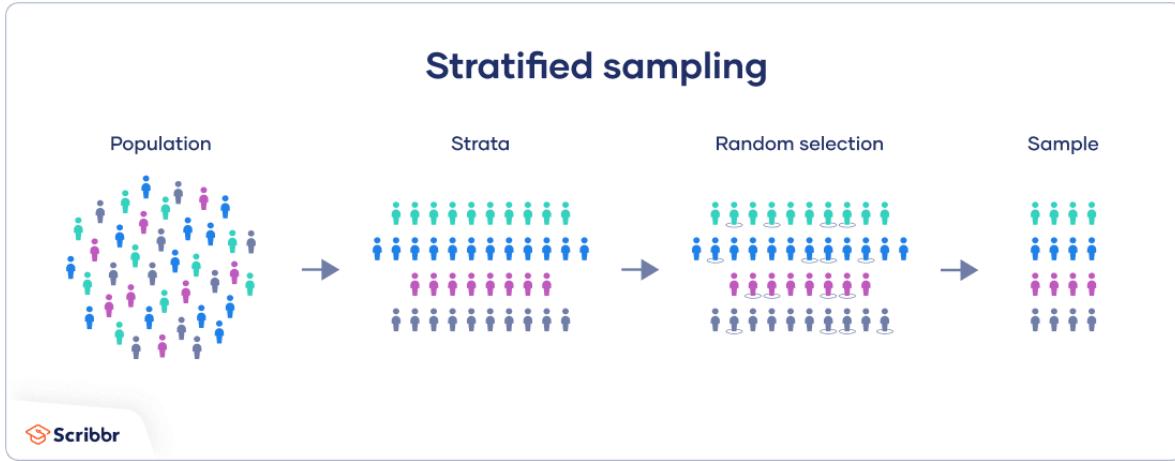


Figure 3: Stratified Sampling. Crediti: Fonte

### 3 Valutazione dei Modelli

Nel contesto della valutazione dei modelli di machine learning, diversi criteri e metriche sono impiegati per misurare le prestazioni. In questa sezione, esploreremo alcune delle principali misure di valutazione e strategie di validazione.

#### 3.1 Misurare le Prestazioni dei Modelli

##### 3.1.1 L'Accuracy

L'Accuracy è una misura comune utilizzata per valutare le prestazioni di un modello. Rappresenta la percentuale di previsioni corrette rispetto al totale delle previsioni.

$$\text{Accuracy} = \frac{\# \text{Correct Predictions}}{\# \text{Total Predictions}}$$

##### 3.1.2 Altre Metriche di Valutazione

Pensandoci bene, con dati sbilanciati l'accuracy non è la soluzione migliore. Oltre all'Accuracy, esistono infatti diverse altre metriche di valutazione, come la precisione, il richiamo, e l'F1-score, ciascuna focalizzata su aspetti specifici delle prestazioni del modello. Introduciamo le seguenti definizioni:

True Positives (TP) = Numero di osservazioni correttamente classificate come positive

True Negatives (TN) = Numero di osservazioni correttamente classificate come negative

False Positives (FP) = Numero di osservazioni erroneamente classificate come positive

False Negatives (FN) = Numero di osservazioni erroneamente classificate come negative

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

La precisione di una classe definisce quanto un modello è accurato nel predire una classe positiva. Più è alto, maggiore è la capacità del modello di evitare falsi positivi.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Il richiamo misura la capacità del modello di catturare o "richiamare" correttamente gli esempi positivi, evitando di trascurarne alcuni. Un valore di recall più alto indica che il modello ha una maggiore capacità di identificare gli esempi positivi, ma potrebbe portare a un aumento dei falsi positivi.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

In generale possiamo dire che Precision e Recall sono due misure quasi "contrastanti", se voglio aumentare uno devo essere disposto a sacrificare l'altro. In genere, in base al contesto applicativo specifico si cerca un trade-off.

### 3.1.3 La ROC Curve

La curva ROC è uno strumento utilizzato per visualizzare le prestazioni di un modello di classificazione binaria. Rappresenta il tasso di veri positivi (TPR) in funzione del tasso di falsi positivi (FPR) a vari livelli di soglia. Più la curva si avvicina all'angolo in alto a sinistra, migliore è il modello. L'area sotto la curva (AUC) è anche una metrica utile per confrontare modelli diversi. In generale: più alta è l'AUC, migliore è il modello.

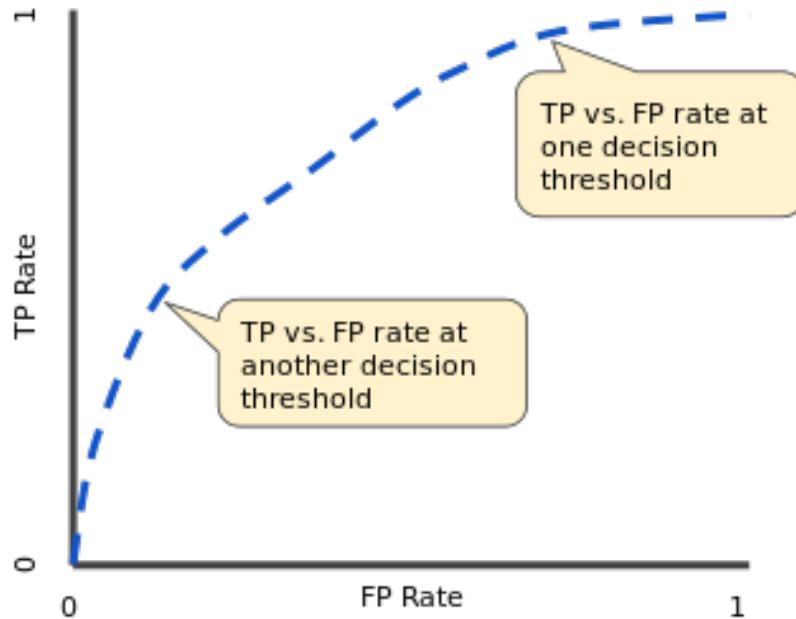


Figure 4: Illustrazione di una curva ROC. Crediti: Fonte

## 3.2 Tabella di contingenza

Una tabella di contingenza è uno strumento utile per rappresentare le relazioni tra due variabili categoriche. Di seguito è riportato un esempio di una tabella di contingenza:

	Predizione Positiva	Predizione Negativa
Effettiva Positiva	True Positive (TP)	False Negative (FN)
Effettiva Negativa	False Positive (FP)	True Negative (TN)

Table 1: Esempio di tabella di contingenza

La Tabella 1 illustra una classica tabella di contingenza utilizzata in problemi di classificazione binaria, dove le condizioni positive e negative della variabile di risposta sono confrontate con le predizioni del modello.

### 3.2.1 Valutazione Multi-Classe & Confusion Matrix

Per modelli che affrontano problemi di classificazione multi-classe, è importante "aggregare" le metriche viste, e sfruttare le matrici di confusione.

Per aggregare utilizziamo solitamente due strategie:

- **Macro:** Calcolo precision, recall o f-measure per ogni singola classe e poi faccio la media.
  - Do la stessa importanza ad ogni classe.

- **Weighted:** Calcolo precision, recall o f-measure per ogni singola classe e poi faccio la media pesata. Dove il peso potrebbe essere il numero di istanze presenti nel dataset per una determinata classe.

– Le classi con più istanze devono dare un impatto maggiore sulla misura finale.

### Matrice di Confusione

Una matrice di confusione è un modo efficace per visualizzare le prestazioni di un modello di classificazione. Di seguito è riportato un esempio di una matrice di confusione con quattro etichette:

	Etichetta 1	Etichetta 2	Etichetta 3	Etichetta 4
Etichetta 1	TP	FP	-	-
Etichetta 2	FN	TP	FP	-
Etichetta 3	-	FN	TP	FN
Etichetta 4	-	-	FN	TP

Table 2: Esempio di matrice di confusione con quattro etichette

La Tabella 2 rappresenta una matrice di confusione con quattro etichette, dove le righe indicano le etichette reali e le colonne indicano le etichette previste dal modello.

### 3.3 Il Set di Validazione

Il set di validazione rappresenta un sottoinsieme cruciale del dataset utilizzato per valutare le prestazioni del modello durante il processo di addestramento, svolgendo un ruolo fondamentale nel rilevare eventuali fenomeni di overfitting.

La sua utilità si estende oltre la semplice valutazione delle prestazioni; infatti, il set di validazione è spesso impiegato per la scelta della configurazione ottimale dei parametri del modello. L'approccio comune consiste nell'identificare i migliori parametri utilizzando il set di training e di validazione, per poi procedere al retraining del modello sull'unione di questi due insiemi, ottenendo così un modello più robusto e in grado di generalizzare su un insieme più ampio di dati.

La suddivisione tipica dei dati tra set di training, validation e test è solitamente del 60% per il training, il 20% per la validation e il 20% per il test.

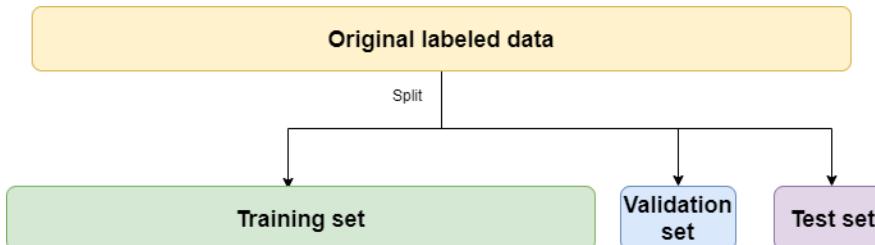


Figure 5: Split del Dataset. Credits: Fonte

### 3.4 Strategie di Valutazione del Modello

La scelta di adeguate strategie di valutazione del modello è fondamentale nel contesto del machine learning. Tali strategie sono utilizzate per valutare le prestazioni di un modello al fine di comprendere come si comporterà su dati non ancora visti e generalizzare al meglio. Le ragioni principali per l'utilizzo di queste strategie includono:

- **Valutazione dell'Affidabilità:** Le strategie di valutazione forniscono una misura dell'affidabilità del modello, indicando quanto bene il modello sarà in grado di generalizzare su nuovi dati.

- **Prevenzione dell'Overfitting:** Una corretta valutazione del modello aiuta a identificare fenomeni come l'overfitting, in cui il modello memorizza i dati di addestramento anziché apprenderne i pattern generali.
- **Comparazione tra Modelli:** Le strategie di valutazione consentono di confrontare le prestazioni di diversi modelli, aiutando a selezionare quello che si adatta meglio al problema specifico.
- **Ottimizzazione dei Parametri:** Durante la fase di valutazione, è possibile ottimizzare i parametri del modello per migliorarne le prestazioni.
- **Assicurazione della Qualità dei Modelli:** Le strategie di valutazione contribuiscono a garantire la qualità del modello, riducendo il rischio di costruire modelli che non generalizzano bene su nuovi dati.

In definitiva, l'uso di strategie di valutazione del modello è cruciale per garantire la validità e l'efficacia dei modelli di machine learning in diversi contesti applicativi.

### 3.4.1 K-Fold Cross Validation

Per rendere il modello più generico, si utilizza una procedura di ricampionamento dei dati chiamata Cross-Validation, ampiamente impiegata anche per valutare le prestazioni del modello.

L'idea fondamentale è quella di dividere i dati in k-fold (da cui il nome k-fold cross-validation). Il parametro k indica in quanti pezzi si suddivide il dataset, e di conseguenza, quante volte si riapplica il processo di training e di validazione.

Una volta addestrati i k modelli, l'accuratezza o qualsiasi altra metrica di valutazione utilizzata viene calcolata separatamente per ciascun modello. Successivamente, l'accuratezza complessiva del modello è ottenuta calcolando la media delle accuratezze ottenute durante le diverse iterazioni di training e validazione, fornendo così una stima più affidabile delle prestazioni del modello su dati non osservati.

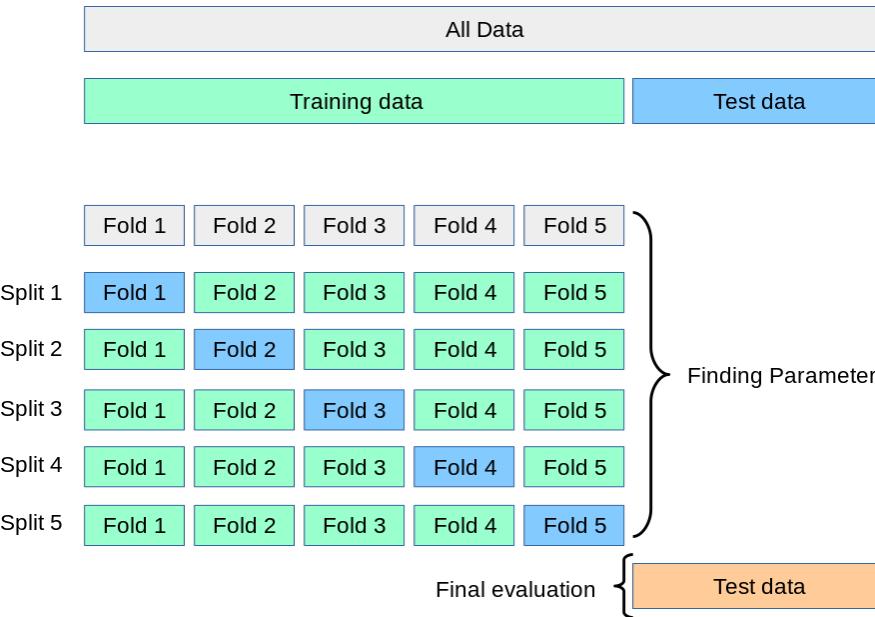


Figure 6: Cross-Validation. Crediti: Fonte

### 3.4.2 Nested K-Fold Cross Validation

Ogni k-fold potrebbe implementare a sua volta un processo k-fold, suddividendo il set di training in training e validation. Questa tecnica è nota come Nested-Cross-Validation. Tipicamente, si seleziona un singolo fold per il train-test principale e, successivamente, si suddivide il set di training in k-fold per l'iterazione interna.

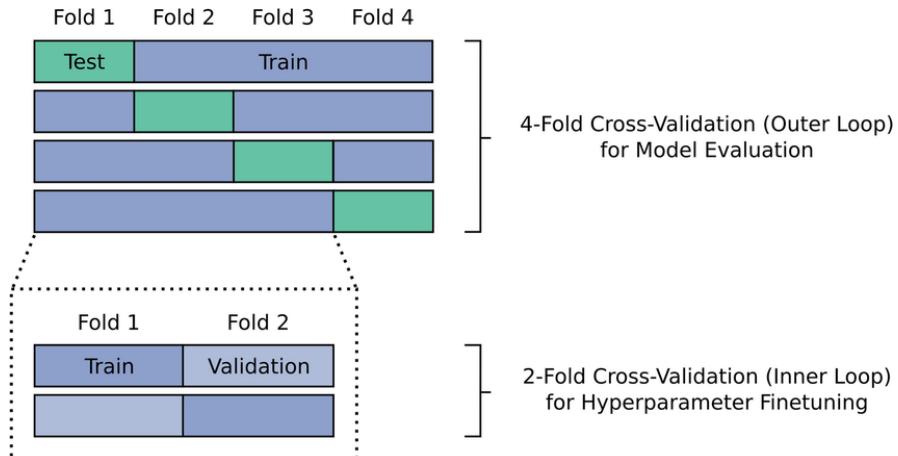


Figure 7: Nested Cross-Validation. Crediti: Fonte

**Scelta del K:** In generale, non esiste un valore di k ottimale che si applichi uniformemente a tutte le situazioni. La scelta di k dipende dalle specifiche caratteristiche del caso. Se si dispone di un ampio volume di dati, è possibile utilizzare valori elevati di k senza preoccuparsi eccessivamente dei costi computazionali. Al contrario, in presenza di un set di dati più limitato, è preferibile utilizzare k più bassi per garantire una suddivisione più statistica e rappresentativa dei dati.  
La strategia Leave-One-Out-Cross-Validation, ad esempio, utilizza k = n, dove n rappresenta il numero totale di istanze nel dataset.

### 3.5 Model Selection and Overfitting

L'**Overfitting** è il fenomeno per il quale il modello, pur continuando a crescere di complessità, non riesce a crescere di espressività. Questo accade perché, quando il modello viene allenato eccessivamente, esso non impara a generalizzare sui dati, bensì memorizza i pattern del set di training. L'Overfitting può essere causato anche da una scarsa qualità dei dati.

L'**Underfitting** è il fenomeno per il quale il modello non è espressivo poiché sotto allenato.

Per scegliere il modello migliore viene calcolato il generalization error dei vari modelli sul test set.

Poiché quest'ultimo tiene conto della complessità del modello, di norma, a parità di errore, si sceglie il modello più semplice.

## 4 Classification and Regression Algorithm

### 4.1 KNN - Nearest-Neighbour Classifier

Il classificatore KNN è un *lazy learner*, il che significa che ritarda il processo di modellazione dei dati di addestramento fino a quando non è necessario classificare le istanze di test.

#### 4.1.1 L'idea

I classificatori Nearest Neighbour si basano sull'apprendimento per analogia. Utilizzano la distanza euclidea per determinare quanto le tuple siano simili.

## K Nearest Neighbors

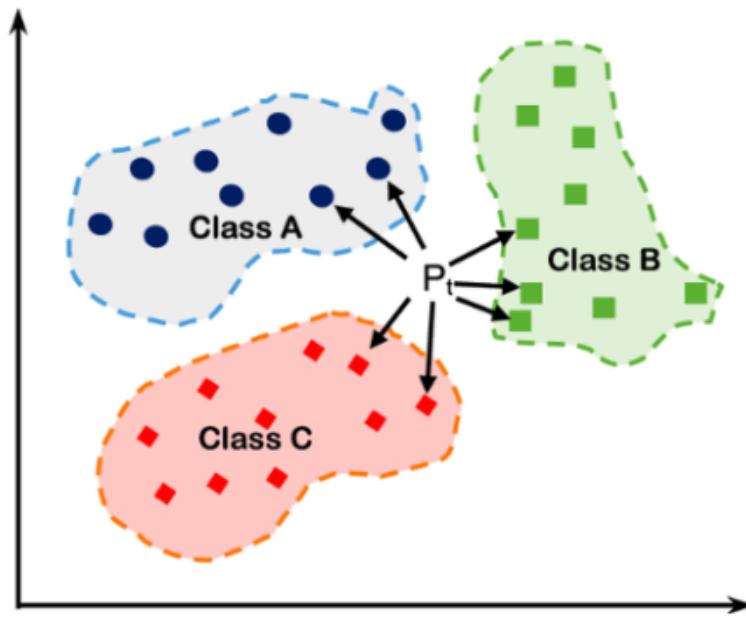


Figure 8: KNN - Cerco la classe per  $P_t$ . Crediti: Fonte

#### 4.1.2 Considerazione sul valore di k

Se  $k$  è troppo piccolo, il classificatore dei vicini potrebbe essere suscettibile all'overfitting dovuto al rumore; d'altro canto, se  $k$  è troppo grande, il classificatore dei vicini più prossimi potrebbe classificare erroneamente le istanze di test.

In particolare, se  $k = 1$ , il modello potrebbe risultare poco affidabile. Se ( $k \rightarrow \infty$ , il modello seguirebbe la moda del dataset.

#### 4.1.3 La predizione

Una volta scelte le  $k$ -tuple più "vicine" alla tupla del set di test, nel caso di classificazione si utilizza la moda della label da predire, nel caso della regressione si utilizza la media sulla label da predire.

#### 4.1.4 Le metriche

L'utilizzo di alcune metriche di distanza, come la distanza euclidea, presenta dei limiti. Questo tipo di distanza assegna lo stesso peso a ogni feature, rendendo possibile che il valore di una specifica feature influenzi in modo eccessivo la decisione di classificazione del modello. Per mitigare questo

effetto, è consigliabile considerare l'utilizzo di tecniche di normalizzazione come il MinMaxScaler, che rimappa ogni feature in un intervallo generalmente compreso tra 0 e 1. Tuttavia, è importante notare che il MinMaxScaler è particolarmente sensibile agli outliers. Un'alternativa potrebbe essere l'impiego dello StandardScaler, che standardizza le feature portandole a una distribuzione normale con media 0 e varianza 1. Questo approccio aiuta a ridurre l'effetto delle feature con varianza elevata sulla varianza complessiva del modello.

#### 4.1.5 Altre Metriche

Nel caso in cui si desideri evitare l'utilizzo della distanza euclidea per calcolare la distanza tra le tuple, è possibile passare all'oggetto k-nn una funzione personalizzata, ossia un callable, che calcoli la distanza. Questa flessibilità consente di adottare metriche di distanza più adatte al contesto specifico del problema.

## 4.2 Decision Tree Classifier

L'albero decisionale rappresenta una delle tecniche di classificazione più intuitive e visivamente accessibili nel campo del machine learning. Il suo approccio organizzato e gerarchico offre una chiara rappresentazione del processo decisionale, rendendolo un valido strumento in diverse applicazioni.

### 4.2.1 L'idea

Possiamo risolvere un problema di classificazione ponendo una serie di domande attentamente formulate sulle caratteristiche dell'istanza di test. La serie di domande e le loro possibili risposte possono essere organizzate in una struttura gerarchica chiamata albero decisionale.

### 4.2.2 Struttura di un albero di decisione

La struttura di un Albero Decisionale è la seguente:

1. Un nodo radice, che non ha collegamenti in ingresso e zero o più collegamenti in uscita.
2. Nodi interni, ciascuno dei quali ha esattamente un collegamento in ingresso e due o più collegamenti in uscita. Ogni nodo non terminale contiene condizioni di test delle caratteristiche (predicati) generalmente definite utilizzando un'unica attributo.
3. Nodi foglia o terminali, ciascuno dei quali ha esattamente un collegamento in ingresso e nessun collegamento in uscita. Ogni foglia è associata a un'etichetta di classe.

Ogni albero decisionale può essere espresso come un albero decisionale binario equamente espressivo.

### 4.2.3 Algoritmo - Stile Ricorsivo

```

1 def BuildTree(D):
2 BestSplit, BestGain = None
3 for each feature f
4     for each threshold t
5         Gain <- goodness of the split (f<=t)
6         if Gain>=BestGain:
7             BestGain <- Gain
8             BestSplit* <- (f<=t)
9 if BestGain = 0 or other stopping criterion is met:
10    u <- the best prediction for D # Media o Moda del Dataset (
11        Regressione o Classificazione)
12    return Leaf(u)
13 Let f and t be those of BestSplit = (f<=t)
14 Dl = {x in D | xf <= t}(Left Partition)
15 L <- BuildTree(Dl) (Left Child)
```

```

15 Dr = {x in D | xf > t}(Right Partition)
16 R <- BuildTree(Dr) (Right Child)
17 return Node(L,R)

```

#### 4.2.4 Scelta del test condition

Esistono molte misure che possono essere utilizzate per determinare la bontà di una condizione di test di un attributo. Queste misure cercano di dare preferenza allo split  $(f, t)$  che partiziona le istanze di addestramento in sottoinsiemi \*più puri\* nei nodi figli, che hanno principalmente le stesse etichette di classe.

Avere un nodo puro è utile perché un nodo con tutte le istanze di addestramento appartenenti a una singola classe non richiede ulteriore espansione. D'altra parte, quando un nodo contiene istanze da classi multiple, viene considerato \*impuro\* ed è probabile che richieda diversi livelli di espansione dei nodi, aumentando la profondità dell'albero decisionale.

Gli alberi più grandi sono più suscettibili all'**Overfitting** del modello.

Dato uno split, il Gain viene calcolato nel seguente modo:

$$Gain(f, t|D) = Error(D) - \left( \frac{|D_l|}{|D|} \cdot Error(D_l) + \frac{|D_r|}{|D|} \cdot Error(D_r) \right)$$

L'impurità di un nodo può essere calcolata usando:

1. Classification Error:  $Error(\mathcal{D}) = 1 - \max_i p_i$
2. Gini Index:  $Error(\mathcal{D}) = Gini(\mathcal{D}) = 1 - \sum_i p_i^2$
3. Gain Info (Entropy):  $Error(\mathcal{D}) = Info(\mathcal{D}) = - \sum_i p_i \log_2(p_i)$
4. Gain Ratio:  $Error(\mathcal{D}) = GainRatio(\mathcal{D}) = \frac{Info(\mathcal{D})}{SplitInfo(\mathcal{D})}$  dove  
 $SplitInfo(\mathcal{D}) = - \sum_{i=1}^k \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \log \left( \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \right)$

$p_i$  rappresenta la frequenza di una classe  $i$  all'interno del dataset. Tutte queste misure hanno due cose in comune:

1. Assegnano un valore di impurità pari a zero se un nodo contiene istanze di una singola classe.
2. Impurità massima se il nodo contiene una proporzione equa di istanze da classi multiple.

#### 4.2.5 Algoritmo & Stopping Criterio

```

1 def BuildTree(D):
2     Tree = []
3     # Find the best split of D
4     (f, t) = best_split(D)
5     gain = goodness_of_split(D, f, t)
6
7     # Initialize the queue with the gain, split condition, and dataset
8     Queue = PriorityQueue()
9     Queue.push((gain, (f, t), D))
10
11    while Queue is not empty and no_other_stopping_criterion_is_met:
12        # Pop the element with the minimum gain from the queue
13        (gain, (f, t), D_star) = Queue.pop()
14
15        # Add the node to the tree at the leaf corresponding to D_star
16        add_node_to_tree(Tree, (f, t), D_star)

```

```

17
18     # Left partition
19     Dl = {x in D_star | x[f] <= t}
20     (f_left, t_left) = best_split(Dl)
21     gain_left = goodness_of_split(Dl, f_left, t_left)
22     Queue.push((gain_left, (f_left, t_left), Dl))
23
24     # Right partition
25     Dr = {x in D_star | x[f] > t}
26     (f_right, t_right) = best_split(Dr)
27     gain_right = goodness_of_split(Dr, f_right, t_right)
28     Queue.push((gain_right, (f_right, t_right), Dr))
29
30     return Tree

```

Il criterio di stop può essere il numero di foglie o la profondità di un albero. Così da evitare l'overfitting, inoltre questo algoritmo **non è recursive**, ciò significa che il criterio di stopping non creerà un albero sbilanciato.

#### 4.2.6 Pruning to avoid Overfitting

Il processo di potatura è una tecnica fondamentale nell'addestramento degli alberi decisionali. L'obiettivo principale della potatura è prevenire l'eccessivo adattamento del modello ai dati di addestramento, noto come overfitting, rendendoli più generalizzabili e meno suscettibili all'overfitting.

1. Pre-Pruning:
  - (a) Diamo una condizione di stopping per la crescita del albero
2. Post-Pruning:
  - (a) Una volta cresciuto completamente l'albero, per ogni nodo guardo quanto "mi serve" il sotto albero, ordino i nodi in base a questo ed elimino i sottoalberi che "non mi servono".
  - (b) Bisogna trovare un bilanciamento tra espressività del modello e complessità del modello. Un modello troppo semplice rischia di essere poco espressivo.
  - (c) Processo maggiormente costoso

#### 4.2.7 Scelta del modello

L'idea è che, più un modello è grande, più esso è complesso, più è overfittato. Preferiamo un albero con un numero di nodi contenuto.

Quindi usiamo l'errore generalizzato per valutare un modello, all'interno del calcolo dell'errore includiamo anche la complessità del modello moltiplicata per un alpha.

$$Error(D, M) = \frac{1}{|D|} \sum_{x,y \in D} E(y, (M(x))) \quad (1)$$

$$Error_{gen}(D, M) = Error(D, M) + a \cdot Complexity(M) \quad (2)$$

$$Complexity(M) = \frac{\#NumberOfLeaves}{|D|} \quad (3)$$

**Sk-Learn Implementation**  $Complexity(M) = L(M)$

**Conclusione** L'alpha ci dice quante istanze devo giustificare per aggiungere una nuova foglia. Ogni nodo avrà un valore di alpha, che determinerà l'importanza del sotto-albero.

Algoritmamente: Ordino i nodi in ordine crescente e poto tutte le sotto-strutture che hanno una soglia inferiore ad un valore che sceglie l'utente.

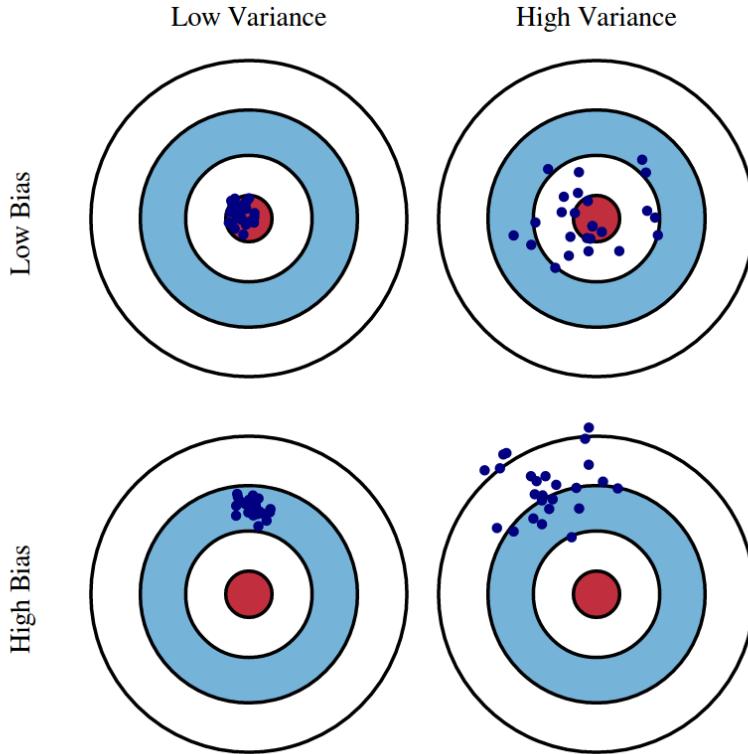


Figure 9: Bias e Varianza. Crediti: Fonte

#### 4.2.8 Small Tree & Fully Grown Tree

Nella costruzione di alberi decisionali, è comune incontrare concetti come "Small Tree" e "Fully Grown Tree". Questi concetti si riferiscono alla profondità e alla complessità dell'albero risultante dall'addestramento dell'algoritmo. Vediamo le differenze.

**Small Tree:** Un piccolo albero è un albero decisionale con una profondità limitata o un numero inferiore di nodi rispetto a un albero completamente sviluppato. La costruzione di un piccolo albero può essere ottenuta utilizzando tecniche di potatura o limitando il numero massimo di divisioni e nodi durante l'addestramento. Esso ha:

1. **Alto Bias**, poiché non fitta bene il training input
2. **Bassa Varianza**, poiché le predizioni non variano significativamente al susseguirsi di input diversi

**Fully Grown Tree:** Un albero completamente sviluppato è un albero decisionale che è stato espanso fino a quando ciascun nodo contiene un numero minimo di istanze o fino a quando viene raggiunto un criterio di arresto specificato. Gli alberi completamente sviluppati tendono ad adattarsi molto bene ai dati di addestramento, ma possono essere suscettibili all'overfitting, specialmente su set di dati di dimensioni ridotte. Esso ha:

1. **Basso Bias**, poiché fitta bene il training input
2. **Alta Varianza**, poiché le predizioni variano significativamente al susseguirsi di input diversi

L'equilibrio tra un piccolo albero e un albero completamente sviluppato dipende dalle caratteristiche del set di dati e dagli obiettivi specifici del problema in questione.

I concetti di Bias e Varianza verranno ripresi nei successivi capitoli quando si parlerà di "Bias-Variances Decomposition".

### 4.3 Linear Regression

La regressione lineare è un modello molto comune che viene usato su task di regressione, è facilmente interpretabile e si adatta facilmente ai dati.

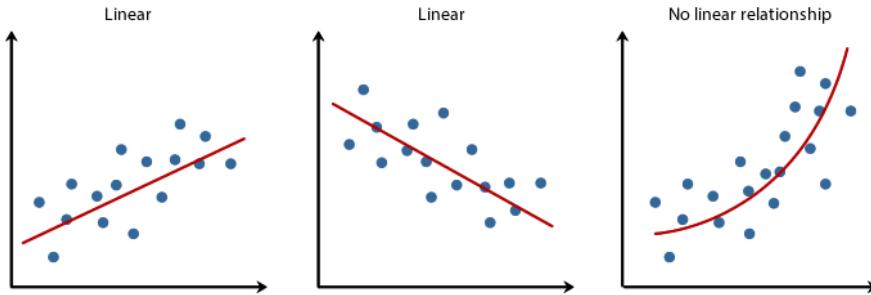


Figure 10: Regressione Lineare. Crediti: Fonte

Utilizzare la regressione lineare equivale a trovare dei coefficienti  $\beta_0, \beta_1 \in \mathbb{R}$  per la retta  $y = \beta_0 + \beta_1 x$  che minimizzano la misura dell'errore, ad esempio la Mean Squared Error (MSE).

$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$  In certi casi il modello lineare non è abbastanza potente, e quindi si può optare di usare un grado polinomiale più elevato (parabola, funzione  $x^3$ , eccetera).

**Attenzione:** se il grado del polinomio diventa troppo elevato (o uguale al numero di punti) c'è alto rischio di overfitting.

```

1 from sklearn.linear_model import LinearRegression
2
3 model = LinearRegression(fit_intercept=True)
4 model.fit(X_train, y_train)
5
6 y_pred = model.predict(X) # note: predicting on both train and
    test
7
8 fig, ax = plt.subplots(figsize=(6,5))
9 ax.plot(X_train, y_train, 'o:', label='train')
10 ax.plot(X_test, y_test, 's:', label='test')
11 ax.plot(X, y_pred, '+:', label='predicted')
12 ax.set_title("Bitcoin Exchange Rate")
13 ax.set_xlabel("days")
14 ax.set_ylabel("USD")
15 ax.legend();

```

### 4.4 Logistic Regression

La regressione logistica è un algoritmo di apprendimento automatico per task di classificazione binaria.

La regressione logistica impiega una funzione di regressione per generare previsioni, ma, poiché si occupa di problemi di classificazione, fa uso di una funzione di attivazione nota come sigmoide  $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$  per ottenere previsioni di probabilità.

Queste probabilità possono essere utilizzate per assegnare etichette di classe. L'addestramento del classificatore avviene attraverso l'utilizzo di un set di dati di addestramento per apprendere i pesi della funzione di regressione. Questo processo coinvolge l'impiego di una funzione di costo denominata log-loss, che misura la discrepanza tra le previsioni del modello e le etichette dei dati di addestramento.

$\text{sig}(\sum_{i=1}^{|D|} x_i \cdot w_i)$  L'algoritmo mira a trovare i pesi  $w_i$  che minimizzano la log loss, attraverso un metodo di ottimizzazione, comunemente la discesa del gradiente.

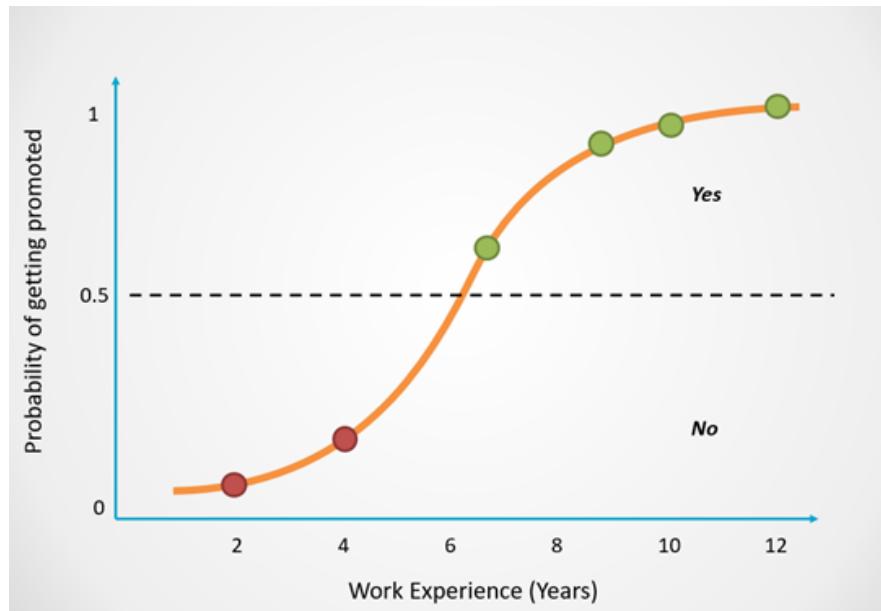


Figure 11: Funzione Logistica,  $[\infty, -\infty] \rightarrow [0, 1]$ . Crediti: Fonte

Una volta addestrato, il modello diventa in grado di effettuare previsioni su nuovi dati inserendo i valori delle caratteristiche in una funzione sigmoide. La previsione del modello viene considerata positiva quando il valore restituito dalla sigmoide supera una soglia prefissata (di solito 0,5), altrimenti è considerata negativa.

Nota: Non funziona se i dati non sono linearmente separabili.

## 4.5 SVM - Support Vector Machines

La SVM è un classificatore inizialmente ideato per la risoluzione di problemi di natura dicotomica.

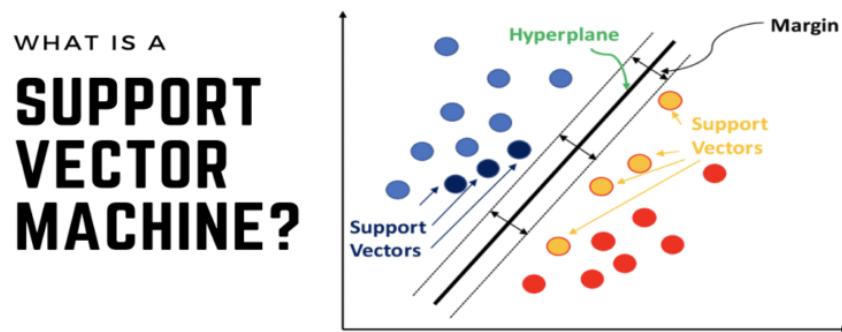


Figure 12: SVM. Crediti: Fonte

### 4.5.1 L'idea

L'algoritmo prova a risolvere il seguente problema: Dato un insieme di punti che appartengono a due classi, qual è la retta che separa meglio le due classi?

### 4.5.2 Problemi Multi-Classe

Per affrontare con successo problemi multi-classe con l'algoritmo, è possibile adottare diverse strategie. Una di queste è la strategia "one-vs-rest". In questo approccio, viene allenato un

classificatore per ogni classe, considerando una specifica classe come positiva e tutte le altre come negative.

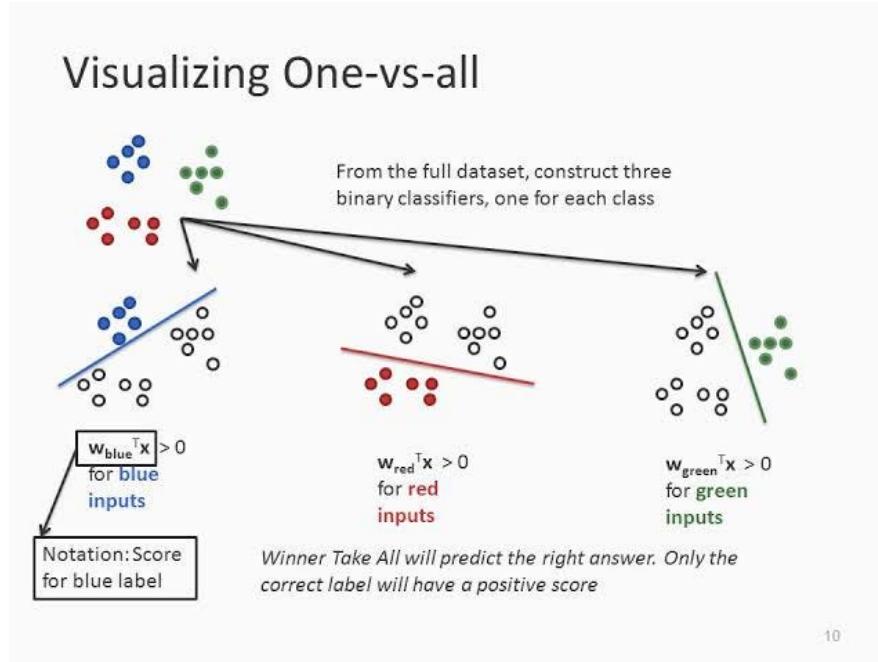


Figure 13: One-Vs-All. Crediti: Fonte

Ogni classificatore restituirà un valore che rappresenta la confidenza che l'istanza appartenga alla classe associata. La classe assegnata è quella con la confidenza più alta tra tutti i classificatori. Altre strategie comunemente impiegate includono la "one-vs-one" (uno-contro-uno), in cui vengono addestrati classificatori per distinguere ogni possibile coppia di classi, e l'approccio "gerarchico". Nell'approccio gerarchico, le classi vengono organizzate in una struttura ad albero, e vengono addestrati classificatori su nodi specifici dell'albero, semplificando così la complessità delle decisioni multi-classe.

#### 4.5.3 Obiettivo

La retta è rappresentata da  $w^t + b = 0$ , dove:

1.  $w$  è il vettore dei pesi associati alle feature. (L'obiettivo della SVM è trovare il vettore  $w$  che massimizza la larghezza del margine tra le due classi )
2.  $x$  è il vettore delle features
3.  $b$  è un bias

Idealmente quello che vorremo fare è massimizzare il margine, dato da :  $\frac{2}{\|w\|}$ . SVM si riduce al seguente problema di ottimizzazione:

1. Minimizzare:  $\frac{1}{2}\|w\|^2$
2. Soggetto ai vincoli:  $y_i(w^T x_i + b) \geq 1 \quad \forall i \in \{1, \dots, n\}$

Soltanamente è possibile avere un soft-margin o un hard-margin, ovvero possiamo permetterci di fare alcuni errori di misclassificazione lasciando che alcune istanze oltrepassino la retta, oppure forzare che ci siano errori nelle predizioni del training set. Questo si può decidere grazie ai due parametri  $\epsilon$  e  $C$ . Il problema di ottimizzazione con l'aggiunta della possibilità di avere un soft-margin si può riscrivere nel seguente modo:

1. Minimizzare:  $\frac{1}{2}\|w\|^2 + C \cdot \sum_i^n \epsilon_i$

$$2. \text{ Soggetto ai vincoli: } y_i(w^T x_i + b) \geq 1 - \epsilon_i \quad \forall i \in \{1, \dots, n\}$$

$C$  è un parametro di regolarizzazione (costo), che controlla il trade-off tra la minimizzazione della norma di  $w$  e la penalità per gli errori di classificazione.

#### 4.5.4 Problemi non linearmente separabili - Kernel Trick

Esistono problemi non linearmente separabili, ciò significa che non esiste un iper-piano che separa le classi. Ma quindi: Come approcciare un problema non linearmente separabile?

**L'idea** principale è mappare i dati in uno spazio delle feature di dimensioni superiori attraverso una funzione kernel.

Questa mappatura consente di trattare il problema non linearmente separabile come se fosse linearmente separabile nello spazio delle feature di dimensioni superiori. In questo nuovo spazio, si può cercare un iperpiano lineare che separi efficacemente le classi.

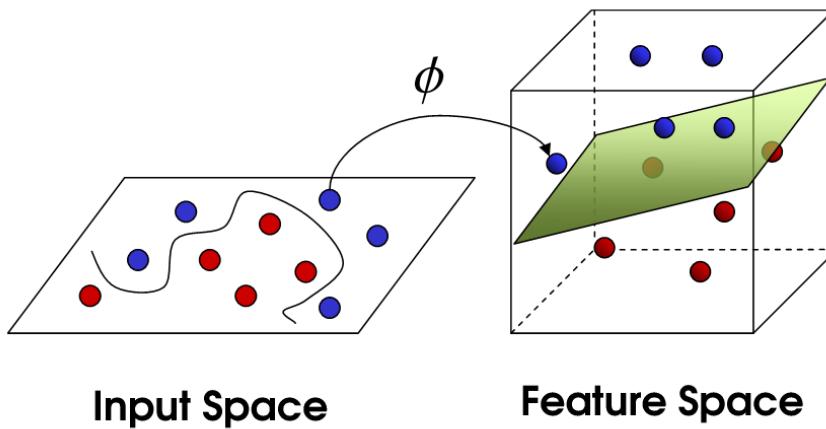


Figure 14: Kernel Trick. Crediti: Fonte

Esempio, chiamo  $z = x^2$ , la nuova feature che trasforma la vecchia feature  $x$ , i valori negativi e positivi sono sempre positivi, ma più piccolo è un numero meno questo viene rappresentato al quadrato, più il valore è grande più viene rappresentato nella spazio 2D. Concludendo: questo tipo di classificatore presenta una buona velocità e migliori prestazioni con un numero piccolo di campioni.

## 5 Ensemble Methods

Un Ensemble Method è una tecnica utilizzata per migliorare l'accuratezza di classificazione mediante l'aggregazione delle predizioni di più classificatori.

### 5.1 Metodi per costruire classificatori ensemble

#### 1) Manipolando l'insieme di addestramento:

In questo approccio vengono creati più insiemi di addestramento mediante il campionamento ripetuto dei dati originali, secondo una specifica distribuzione di campionamento. **Bagging** e **Boosting** sono due esempi di metodi di ensemble che manipolano l'insieme di dati di addestramento.

#### 2) Manipolando le caratteristiche di input:

In questo approccio, ogni insieme di addestramento è creato utilizzando un sottoinsieme casuale delle caratteristiche. Alcuni studi hanno dimostrato che questo tipo di approccio funziona molto bene con set di dati che contengono caratteristiche altamente ridondanti, ad esempio: \*Random Forest\*

#### 3) Manipolando le etichette di classe:

#### 4) Manipolando l'algoritmo di apprendimento:

I primi tre approcci sono metodi generici applicabili a qualsiasi classificatore, mentre il quarto approccio dipende dal tipo di classificatore utilizzato.

#### 5.1.1 Bias-Variances Decomposition

La *bias-variance decomposition* è un concetto chiave nell'analisi degli errori di previsione nei modelli di apprendimento automatico. Questa decomposizione fornisce una comprensione dettagliata delle fonti di errore totale di un modello e suggerisce come migliorarne le prestazioni.

L'errore complessivo di un modello  $M$  può essere scomposto nella somma di tre componenti:

$$\text{Error}(M) = \text{Bias}^2 + \text{Variance} + \text{Noise}$$

Il *Bias* rappresenta la discrepanza tra la previsione attesa del modello e il valore corretto che si sta cercando di prevedere. In pratica, questo misura quanto, in generale, le previsioni dei modelli si discostano dal valore corretto. La scomposizione del *Bias* e della *Variance* consente di analizzare in dettaglio queste discrepanze e di individuare come migliorare la precisione complessiva del modello. L'errore dovuto alla *variance* misura la variabilità delle previsioni del modello per un determinato punto dati. Questo componente riflette quanto le previsioni per un dato punto variano tra diverse realizzazioni del modello. Minimizzare la *variance* è altrettanto importante quanto ridurre il *bias* per ottenere un modello che generalizzi bene su nuovi dati.

Il *Noise* rappresenta gli errori intrinseci nel set di dati, come etichette sbagliate o caratteristiche errate, che non possono essere eliminati durante il processo di modellazione.

In sintesi, un modello mostra una migliore capacità di generalizzazione se riesce a ridurre sia il *bias* che la *variance*. La comprensione di queste componenti aiuta a ottimizzare il modello per ottenere prestazioni migliori su nuovi dati.

## 5.2 Bagging

Il Bagging, noto anche come aggregazione bootstrap, è una *tecnica* che campiona ripetutamente (con rimpiazzo) da un insieme di dati secondo una distribuzione di probabilità uniforme. Un campione bootstrap ha la stessa dimensione dell'insieme di dati originale; poiché il campionamento è effettuato con rimpiazzo, alcune istanze possono comparire più volte nello stesso set di addestramento, mentre altre possono essere omesse dal set di addestramento.

In media, un campione bootstrap  $D_i$  contiene approssimativamente il 63% dei dati di addestramento originali, poiché ogni campione ha una probabilità di essere campionato pari a  $1 - (1 - \frac{1}{N})^N$ . Con un grande  $N$ , questo converge a  $1 - \frac{1}{e}$ , che è approssimativamente 0.632.

#### 5.2.1 Algoritmo

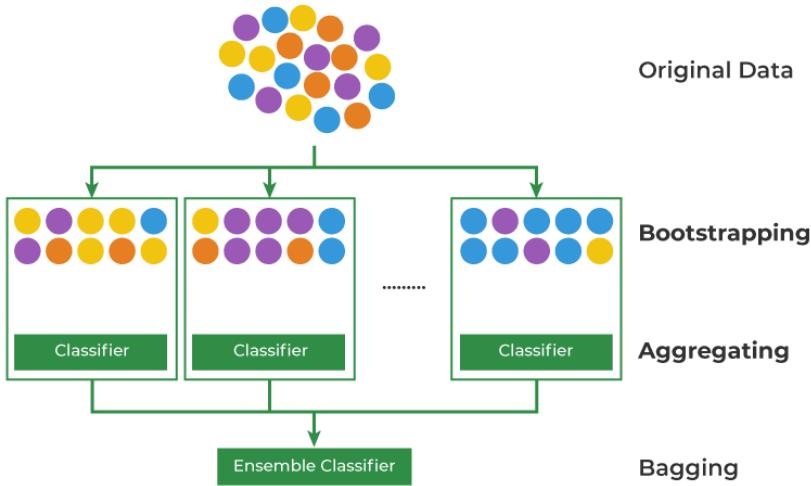


Figure 15: Bagging Algorithm. Crediti: Fonte

```

1 def Bagging Algorithm(D,k):
2   for i=1 to k: #let k be the number of bootstrap samples
3     Create a bootstrap sample D_i of D
4     Build a model M_i on D_i
5
6 Prediction for instance x is the combination of all predictions M_i(x)
7   (majority voting or average)

```

### 5.2.2 Predizione

La votazione a maggioranza viene utilizzata per la classificazione, mentre la media viene utilizzata per i problemi di regressione.

### 5.2.3 Conclusione

Sfruttare il parallelismo emerge come una strategia efficace per ridurre la varianza di un classificatore. Tuttavia, è importante notare che se i modelli non sono indipendenti tra di loro, questa strategia potrebbe non fornire i risultati desiderati. In generale possiamo applicare bagging a modelli **indipendenti** con bias basso e alta varianza

## 5.3 Boosting

Idealmente, vorremmo correggere quelle istanze per le quali la nostra previsione non è stata corretta. Un modo per affrontare questo problema è utilizzare gli **algoritmi di boosting**. Gli algoritmi di boosting sono tecniche di apprendimento automatico che allenano iterativamente modelli deboli e li combinano per creare un modello predittivo forte. Concentrandosi sulle istanze che sono state classificate erroneamente nelle iterazioni precedenti, gli algoritmi di boosting possono migliorare gradualmente l'accuratezza delle previsioni. Questo processo iterativo aiuta a correggere le istanze per le quali la nostra previsione non è stata corretta.

### 5.3.1 Algoritmo

Una possibile implementazione:

```

1 def AdaBoost(D,K):
2   1. initialize the weight w_j=1/n for each instance x_j

```

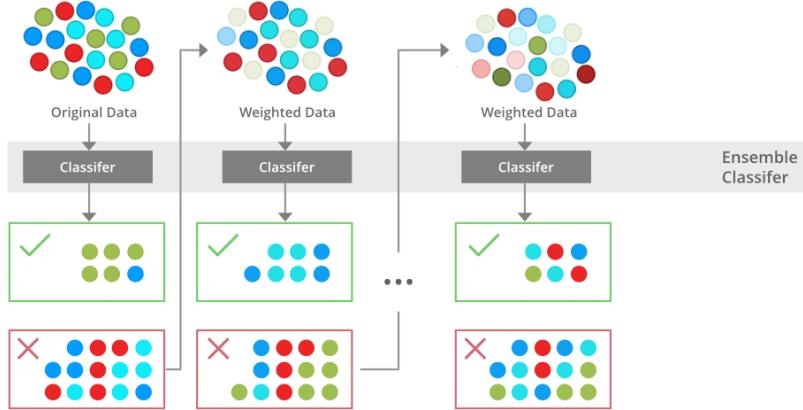


Figure 16: Boosting Algorithm. Crediti: Fonte

```

3 2. E = []
4 3. for i=1 to k:
5 4.     get D_i by sampling D with replacement according weights w_j
6 5.     train a model M_i on D_i
7 6.     compute error(M_i) on D #Attenzione che qui l'errore viene
    calcato su tutto il dataset
8 7.     if error(M_i) > 0.5 then
9 8.         reset weights wj = 1/n and go to step
    4
10 9.     compute the importance of M_i alpha_i
11 10.    update weights of instances in D
12 11.    add alpha M_i to the final ensemble E
13 12. return E

```

L'  $\alpha_i$  è un parametro che tiene conto di quanto il modello  $i$ -esimo è corretto:  $\alpha_i = \log\left(\frac{1 - \text{Error}(M_i)}{\text{Error}(M_i)}\right)$   
 $\text{Error}(M_i)$  è l'errore del modello calcolato sul dataset originale.

$$\text{Error}(M_i) = \frac{1}{N} \sum_{(x_j, y_j) \in D} w_j \cdot [M_i(x_j) \neq y_j] : 1 : 0;$$

L'idea è che ogni modello cerca di compensare i punti deboli del proprio predecessore, aggiornando i pesi nel seguente modo:

Se l'istanza  $x_j$  in  $D$  è correttamente classificata:  $w_j = w_j e^{-\alpha_i}$  Altrimenti:  $w_j = w_j e^{\alpha_i}$  Normalizzo i pesi alla fine:  $w_j = \frac{w_j}{\sum_k w_k}$

### 5.3.2 Predizione

Alla fine ogni modello avrà associato un peso alpha, la predizione è data da un majority voting ponderato.

### 5.3.3 Conclusione

Non è possibile sfruttare il parallelismo poiché ciascun modello dipende dal suo predecessore.  
Tuttavia, questa strategia contribuisce a ridurre il bias di un classificatore, poiché comporta il ri-addestramento su istanze particolarmente complesse. In generale possiamo applicare il boosting a modelli con alto bias e bassa varianza

## 5.4 Random Forest

Si utilizza la tecnica del bagging per migliorare l'accuratezza di un singolo decision tree, inoltre ad ogni split il sottoinsieme di feature da scegliere è random. Forzandomi ad avere alberi diversi ed indipendenti allenati sullo stesso dataset.

Scegliere ad ogni split il sottoinsieme di feature è vantaggioso poiché ci permette di allenare l'albero su potenzialmente tutto l'insieme delle feature, evitando di escludere feature importanti.

Ogni albero è \*\*\*FullyGrown\*\*\* fintantoché non si arriva a pure leaves. Questo perché siccome il bagging non riduce il \*Bias\*, abbiamo bisogno di alberi con \*Bias\* basso.

\*d\* è il numero di feature nel dataset D e solitamente F è scelto come un sottoinsieme stretto.

Solitamente:  $F = \sqrt{d}$  or  $F = \log_2(d)$

### 5.4.1 Algoritmo

```

1 def RF(D, k):
2     for i = 1 to k:
3         get a bootstrap sample Di from dataset D
4             train a full tree Mi on Di
5                 at each split use only F << d random
6                     feature
7     RF = None
8     for i = 1 to k:
9         RF.add(Mi)
10    return RF

```

### 5.4.2 Predizione

La votazione a maggioranza viene utilizzata per la classificazione, mentre la media viene utilizzata per i problemi di regressione.

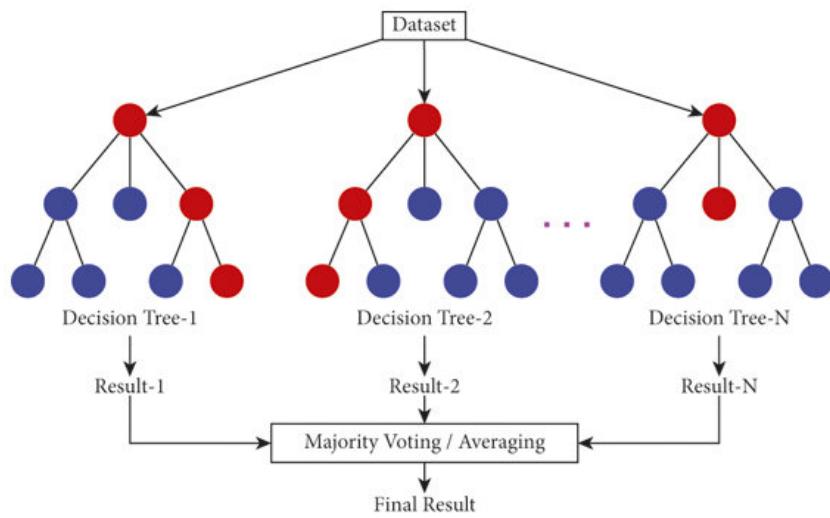


Figure 17: Random Forest Prediction. Crediti: Fonte

### 5.4.3 Conclusione

Sfruttare il parallelismo emerge come una strategia efficace per ridurre la varianza di un classificatore. La Random input selection ad ogni nodo forzerà che i modelli siano indipendenti tra di

loro. In generale la R.F è in grado di gestire grandi insiemi di dati con numerose caratteristiche, rendendola robusta e adatta a problemi complessi.

#### 5.4.4 Random Forest Similarity

L'idea è che due istanze sono simili, se attraversano la foresta casuale lungo percorsi simili, ovvero se cadono sulle stesse foglie. La similarità tra due istanze è:

$$RM_{sim}(o_i, o_j) = \frac{1}{n} \cdot \sum_{k=0}^n [leaf_k(o_i) == leaf_k(o_j)]$$

In linea di principio questa funzione può essere applicata a qualsiasi foresta di alberi decisionali, inclusi bagging e boosting. Ma la random forest è preferibile poiché:

1. Rispetto al Boosting essa da lo stesso peso ad ogni albero.
2. Rispetto al Bagging, la Random Forest contiene alberi diversi ed indipendenti.

**Outliers** Idealmente possiamo usare la random forest per capire quando un'istanza è dissimile dalle altre, scovando in questo modo gli outliers.

Definiamo così l'Outlying Score:

$$out(o_i) = \frac{1}{\sum_{o_j \in D} RFsim(o_i, o_j)^2}$$

**Missing Value** Il punteggio di somiglianza può essere sfruttato anche per l'imputazione del valore mancante, ovvero per sostituire un valore mancante con un'ipotesi ragionevole. Chiamiamo  $o_j \in D$  l'istanza dove manca la caratteristica  $f$

L'Algoritmo è il seguente:

1. Sostituisco il valore mancante con:

$$o'_j[f] = \frac{\sum_{o_i \in D, o_i[f] \neq NaN} o_i[f] \cdot RMsim(o_i, o_j)}{\sum_{o_i \in D, o_i[f] \neq NaN} RMsim(o_i, o_j)}$$

‘In italiano:’ Sto calcolando una valore medio in modo pesato, quindi le istanze più simili avranno maggiore importanza.

2. Addestro una nuova Random Forest sul Dataset modificato, e calcolo i nuovi punteggi di somiglianza. (la nuova RF potrebbe essere totalmente diversa)
3. Ripetere fino a convergenza.

## 6 Feature Analysis

### 6.1 Importanza delle Feature e Selezione Futura

Se un'informazione è veramente derivabile dalle altre feature presenti nel dataset, significa che non fornisce contributi unici o distintivi rispetto a quanto già catturato dalle altre feature. In questo caso, la feature è considerata ridondante perché aggiunge poco o nessun valore informativo aggiuntivo al modello.

D'altra parte, se un'informazione non è derivabile dalle altre feature, significa che la feature ha un contributo unico e non può essere completamente rappresentata o prevista dalle altre. Mantenere tale feature potrebbe comportare un errore leggermente più elevato, ma apporta informazioni preziose e non ridondanti al modello.

Eliminare una feature poco importante offre diversi vantaggi:

1. Migliora la comprensione del modello.
2. Riduce il rumore nei dati.
3. Riduce il tempo necessario per addestrare il set di addestramento e l'occupazione di spazio.

La scelta della feature da eliminare può essere effettuata considerando l'importanza di ciascuna feature nei nodi di un decision tree o di una Random Forest. Tuttavia, è essenziale considerare che l'importanza di un nodo dipende dalla sua capacità di discriminare, valutata durante il training (Delta Gain).

Ogni albero della Random Forest ha la capacità di calcolare l'importanza di una feature in base alla sua capacità di aumentare la purezza delle foglie (maggiore la purezza, più importante è la feature). Una volta calcolata l'importanza di una feature, la funzione di Recursive Feature Elimination (RFE) esegue la selezione delle feature.

Soltanamente, è consigliabile combinare questa strategia con la Cross-Validation. Un'idea poco aggressiva è eliminare la feature meno importante dopo il fitting del modello, calcolando le performance mediante k-fold Cross-Validation. Questo processo viene ripetuto ricorsivamente fino all'esaurimento di tutte le features. Il set di features con il punteggio di performance più alto, calcolato durante l'intero procedimento, è considerato il miglior set di features dalla Random Forest. La funzione RFECV è quella che consente di eseguire questo processo. .

### 6.2 Feature Engineering

Affrontare alcune sfide comuni nei dati è essenziale per garantire la robustezza e l'efficacia di un modello di machine learning. Alcune di queste sfide includono:

### 6.3 Feature Categoriali

Nel caso delle feature categoriali, è fondamentale trattarle correttamente per adattarle al processo di modellazione. I metodi comunemente utilizzati includono:

1. **Binarie:** Esegue un re-mapping tra 0 e 1.
2. **Nominali:** Sostituisce la variabile nominale con n variabili binarie che la codificano.
3. **Ordinali:** Utilizza il label encoding, sostituendo la variabile con un ID. È necessario prestare particolare attenzione in contesti di regressione. Per approfondire, puoi consultare il concetto di *One-Hot Encoding* nella sezione 6.3.1..

#### 6.3.1 One Hot-Encoding

**Features categoriche ordinali** l'idea è che ad ogni valore categorico è assegnato un numero

**Attenzione:** In regressione alto non è necessariamente 3 volte basso.

Valore Categorico	Numero Assegnato
Basso	1
Medio	2
Alto	3

Table 3: Le feature categoriche ordinali: Ad ogni valore categorico è assegnato un numero.

**Features categoriche nominali** Qui bisogna destare maggiore attenzione, poiché assegnare semplicemente un numero può indurre in modo implicito una gerarchia.

*Esempio:*

Blu→1, Rosso→2 ,Verde→3. Chi ci dice che il Verde è tre volte più importante del Blu?

Sia utilizzando una regressione lineare,  $w_i \cdot 3$ , che utilizzando un albero  $X < 3$ , stiamo implicitamente imponendo un ordine.

Valore Categorico	Codifica One-Hot	Codifica One-Hot	Codifica One-Hot
Blu	1	0	0
Rosso	0	1	0
Verde	0	0	1

Table 4: Come soluzione si utilizza un metodo di sostituzione della variabile categoriale con n variabili binarie (Una per ogni variabile da codificare).

**Conclusione** Il processo di One-Hot Encoding può portare ad un esplosione nel numero delle colonne del dataset, con un inevitabile impatto sul processo di addestramento dei modelli. Una strategia può essere quella di fare One-Hot Encoding solo per i k valori più frequenti, mappando gli altri meno frequenti con un valore speciale “other”.

## 6.4 Feature "Unique"

Le feature "unique", ossia quelle che non apportano espressività al modello, dovrebbero essere eliminate. La loro inclusione non contribuisce significativamente alla capacità predittiva e può influenzare negativamente le prestazioni del modello.

## 6.5 Missing Values

La gestione dei valori mancanti è cruciale per mantenere l'integrità dei dati. Alcune possibili strategie includono:

**Possibilità 1:** Utilizzo di Random Forest per predire i valori mancanti.

**Possibilità 2:** Rimpiazzare con la media (nel caso di regressione) o moda (nel caso di classificazione).

**Possibilità 3:** Aggiungere una feature binaria che indica se l'elemento è mancante o no. Questa soluzione può fornire informazioni aggiuntive sui dati in certi contesti.

**Possibilità 4:** Eliminare l'istanza con valori mancanti, con il rischio di perdere informazioni.

**Possibilità 5:** Eliminare l'intera colonna della feature con valori mancanti, con il rischio di perdere informazioni.

Affrontare queste sfide in modo oculato è fondamentale per garantire che il modello di machine learning sia in grado di effettuare previsioni accurate e affidabili.

## 7 Text-Processing

Il trattamento del testo è un aspetto cruciale nell'ambito dell'elaborazione delle informazioni, dove l'analisi e la manipolazione del linguaggio scritto rivestono un ruolo fondamentale. In questa sezione, esploreremo le tecniche e gli strumenti dedicati al processamento del testo.

### 7.1 TD-IDF - Term Frequency - Inverse Document Frequency

E' un algoritmo che utilizza la frequenza delle parole per determinare quanto tali parole siano rilevanti per un dato documento. L'idea è che più una parola appare, meno è importante.

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

Dove:

1.  $tf(t, d) = \frac{|\{x : x = t \text{ and } x \in d\}|}{|d|}$ , fornisce informazioni sulla frequenza con cui un termine appartiene in un documento  $d$ .
2.  $idf(t, D) = \ln(\frac{|D|}{df(t)})$ , fornisce informazioni sulla relativa rarità di un termine  $t$  nella raccolta dei documenti  $D$ .

$df(t)$  dice quanti documenti contengono il termine  $t+1$ .

**Sklearn implementation: TfIdfVectorizer.**

```
1 from sklearn.feature_extraction.text import TfIdfVectorizer
2
3 # Esempio di corpus di documenti
4 corpus = [
5     'Questo    il primo documento.',
6     'Questo documento    il secondo documento.',
7     'E ora c\'    un terzo documento.',
8     'Questo    l\' ultimo documento.',
9 ]
10
11 # Creazione di un oggetto TfIdfVectorizer
12 tfidf_vectorizer = TfIdfVectorizer()
13
14 # Adattamento del vettorizzatore al corpus e trasformazione dei
15 # documenti
16 tfidf_matrix = tfidf_vectorizer.fit_transform(corpus)
17
18 # Ottieni le feature (parole) identificate dal vettorizzatore
19 feature_names = tfidf_vectorizer.get_feature_names_out()
20
21 # Creazione di un DataFrame pandas per visualizzare la matrice TF-IDF
22 import pandas as pd
23 df_tfidf = pd.DataFrame(data=tfidf_matrix.toarray(), columns=
24                         feature_names)
25 print(df_tfidf)
```

Listing 1: Esempio di utilizzo della libreria TfIdfVectorizer.

### 7.2 Stemming & Lemming

In combinazione all' TF-IDF, si possono utilizzare delle tecniche che permettono di ridurre la dimensionalità del vocabolario, favorendo di conseguenza un aumento dell'efficienza computazionale.

L'idea è quella di raggruppare in un solo token, tutte le variabili che hanno la stessa semantica.

### 7.2.1 Stemming

Riduce una parola alla sua forma radice o "stemma", rimuovendo eventuali suffissi. Ad esempio, le parole "running" e "runner" possono essere ridotte entrambe a "run".

Questo tipo di tecnica lavora sulla sintassi.

### 7.2.2 Lemming

Riduce una parola alla sua forma di base o "lemma", tenendo conto della sua forma grammaticale. Ad esempio, il lemma di "running" sarà "run", mentre il lemma di "better" sarà "good".

Questo tipo di tecnica lavora sulla sintassi e sulla semantica.

## 7.3 Caso Applicativo

**Recupero dell'Informazione (Information Retrieval):** In sistemi di ricerca, motori di ricerca e archivi di documenti, TF-IDF viene utilizzata per valutare la rilevanza di un documento rispetto a una query. I documenti con punteggi TF-IDF più alti sono considerati più rilevanti.

## 7.4 Naive Bayes

Naive Bayes è una tecnica semplice per costruire classificatori. Esistono diversi algoritmi che si basano su un principio comune che permettono di costruire i classificatori.

**Il principio comune:**

Tutti i classificatori ingenui (naive) presuppongono che il valore di una particolare feature sia indipendente dal valore di qualsiasi altra feature.

Naive Bayes è un modello di machine learning ampiamente utilizzato nel processo di text processing. Questo classificatore si basa sul Teorema di Bayes e offre un'efficace soluzione per problemi di classificazione in ambienti di linguaggio naturale.

Il Teorema di Bayes viene formulato come segue:

$$P(Y|X) = \frac{P(X, Y)}{P(X)}$$

## Naive Bayes Classifier

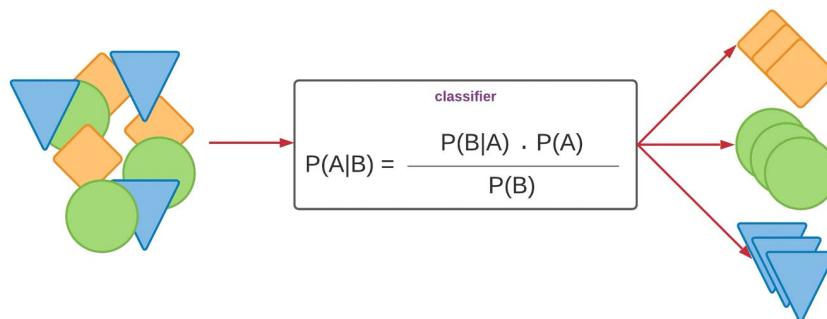


Figure 18: Naive Bayes Classifier. Crediti: Fonte

Nel contesto del machine learning, l'obiettivo diventa calcolare la probabilità di una particolare classe dato un'istanza:

$$P[y|X] = \frac{P(X|Y) \cdot P(Y)}{P(X)}$$

Considerando l'assunzione di indipendenza delle feature, la probabilità condizionata può essere approssimata come il prodotto delle probabilità condizionate delle singole feature:

$$P(X|C_i) = P(X_1|C_i) \cdot P(X_2|C_i) \cdot \dots \cdot P(X_f|C_i)$$

$$\arg \max_{c_i} P(C_i|X) = P(X|C_i)P(C_i)$$

La classe predetta è quella che massimizza la probabilità  $P(C_i|X)$ . Notiamo che quando si confrontano le probabilità condizionate per diverse classi, la costante di normalizzazione ha lo stesso effetto su tutte, e quindi  $P(X)$  può essere ignorata per semplificare i calcoli.

$$\arg \max_{c_i} P(C_i|X) = P(X|C_i)P(C_i)$$

#### 7.4.1 Come Calcolare: $P(x_j|C_i)$

Per la probabilità condizionata di  $x_j$  dato  $C_i$ :

Se la feature  $f_j$  è **categorica**:

$$P(x_j|C_i) = \frac{\#\text{Numero di istanze di classe } C_i \text{ con } x_j = x}{|C_i|}$$

Se la feature  $f_j$  è **numerica**, assumiamo una distribuzione gaussiana:

$$P(x_j|C_i) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_j - \mu)^2}{2\sigma^2}}$$

#### 7.4.2 Zero-Frequency Problem

Se  $P(x_j = x|C_i) = 0$ , si utilizza la correzione di Laplace:

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

$N_{ij}$  è il numero di istanze nella classe  $C_i$  con il valore della feature  $x_j = x$ ,  $N_i$  è il numero di istanze nella classe  $C_i$ , e  $v$  è il numero totale di valori unici della feature  $f_j$  nel dataset  $D$ .

#### Gestione di Missing Values

- **Tempo di Training:** Si può saltare l'istanza.
- **Tempo di Test:** Si può assumere che  $P(x_j|C_i)$  sia la stessa per ogni  $C_i$ .

**Complessità Computazionale** Osserviamo che la moltiplicazione tra le varie probabilità e la scarsa precisione dei numeri floating point possono portare a probabilità molto basse. Pertanto, si usa spesso la trasformazione in logaritmo e la somma dei logaritmi:

$$\log(P(X|C_i))$$

#### 7.4.3 In Conclusione

Il classificatore Naive Bayes è estremamente veloce sia da allenare che da utilizzare in pratica. È robusto al rumore grazie al calcolo di statistiche globali. Inoltre, non ha parametri.

## 8 ANN - Artificial Neural Network

Le reti neurali artificiali (ANN) sono potenti modelli di classificazione in grado di catturare relazioni complesse e non lineari tra le variabili indipendenti e dipendenti. Sono ben conosciute e ampiamente accettate in diversi ambiti, come la visione, il linguaggio e l'elaborazione del linguaggio. In questi settori, hanno costantemente dimostrato prestazioni superiori rispetto ad altri modelli di classificazione e, in alcuni casi, hanno persino superato le capacità umane.

### 8.1 Origini

Nel corso della storia, l'esplorazione delle reti neurali artificiali è stata alimentata dall'ambizione di replicare le complessità dei sistemi neurali biologici. Il cervello umano, composto da circa 100 miliardi di neuroni interconnessi che comunicano attraverso assoni e dendriti, forma una rete complessa essenziale per la nostra capacità di imparare nuovi compiti e svolgere attività quotidiane. Ogni neurone svolge una funzione modulare fondamentale, rispondendo agli stimoli nervosi e trasmettendoli. Tuttavia, è la combinazione di queste funzioni di base che permette l'espressione di comportamenti complessi. Questo concetto fondamentale serve come base per la costruzione delle reti neurali artificiali.

Un altro aspetto critico è la forza del punto di connessione, noto come sinapsi, tra una dendrite e un assone, che determina la connettività neuronale. Gli neuroscienziati hanno osservato che il cervello umano impara modificando la forza di connessione sinaptica tra neuroni attraverso la stimolazione ripetuta dello stesso impulso. Questo processo dinamico sottolinea ulteriormente la natura straordinaria dell'architettura all'interno della rete neurale del cervello umano.

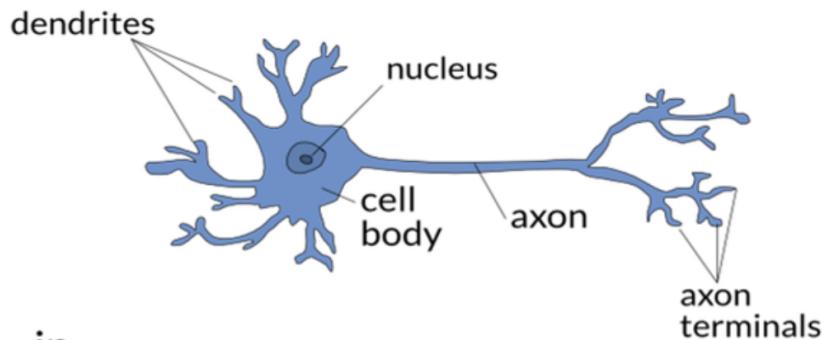


Figure 19: Immagine di un neurone biologico, Crediti: Fonte

### 8.2 Struttura

Una rete neurale artificiale riflette la complessa organizzazione del cervello umano, composta da unità di elaborazione chiamate nodi. Questi nodi, equivalenti ai neuroni, agiscono come unità di calcolo fondamentali. Le connessioni tra i nodi, rappresentate da collegamenti diretti, assomigliano alle connessioni interneuronali presenti nei sistemi biologici e sono composte da assoni e dendriti. Il peso assegnato a un collegamento diretto indica la forza della connessione sinaptica tra neuroni. Proprio come nei sistemi neurali biologici, l'obiettivo principale di una Rete Neurale Artificiale (ANN) risiede nell'adattare questi pesi. La rete subisce un processo di apprendimento, regolando i pesi dei collegamenti fino a che non si allineano con le relazioni input-output intrinseche nei dati sottostanti. Questo adattamento iterativo riflette i meccanismi di apprendimento osservati nei sistemi neurali biologici, contribuendo alla capacità della rete di discernere pattern e relazioni all'interno di dataset complessi.

### 8.3 In Sintesi

Le reti neurali sono potenti classificatori in grado di catturare pattern non lineari.

Traggono ispirazione dalle reti neurali biologiche, dove l'idea fondamentale è che i neuroni sono interconnessi attraverso sinapsi. Modificando la forza di tali connessioni sinaptiche, la rete neurale apprende i pattern.

Ogni neurone riceve in input l'output di altri neuroni, elaborandoli attraverso una funzione di attivazione e producendo un output che viene poi passato ai neuroni dello strato successivo. Infine, tramite propagazione, un risultato raggiunge i neuroni di output. La score function, o loss function, misura quanto la rete ha commesso errori. Sulla base di questo, grazie al processo chiamato "back propagation", i pesi delle varie sinapsi vengono aggiustati, concludendo così un'epoca di training. La rete neurale viene quindi allenata per diverse epoche di training, fino al completamento delle epoche prefissate o fino a quando un criterio di stop termina il processo di allenamento.

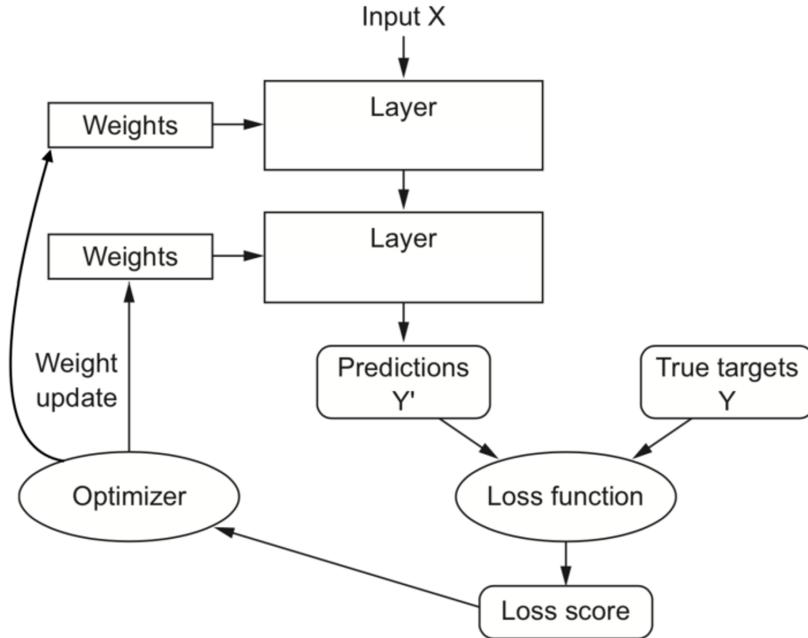


Figure 20: Crediti: Tan, P.-N., Steinbach, M., Kumar, V. (Year). Introduction to Data Mining (2nd ed.). Publisher.

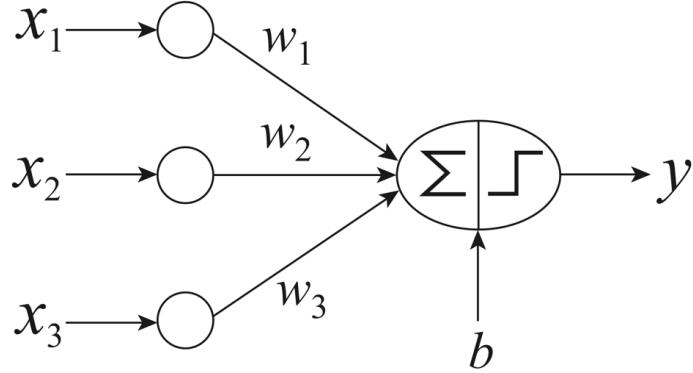
## 8.4 Perceptron

I percetroni non sono una nozione recente nelle reti neurali artificiali; infatti, risalgono agli anni '60 quando presero forma come perceptron, un tipo basilare di rete neurale artificiale. Questa architettura è composta da due tipi principali di nodi: nodi di input che rappresentano gli attributi di input e nodi di output che rappresentano l'output del modello. Col passare del tempo, il concetto di perceptron si è evoluto in perceptron multistrato, ora sinonimo delle moderne reti neurali artificiali. Nonostante la loro esistenza da decenni, l'attuale crescita di popolarità è plasmata da fattori come la limitata potenza di elaborazione dei computer e la scarsità di dati di addestramento. Il paesaggio attuale sta subendo cambiamenti significativi; i computer stanno diventando più veloci e potenti, e Internet funge da fonte abbondante di dati diversificati. Quest'era trasformativa significa l'adozione diffusa e il progresso delle reti neurali artificiali.

### 8.4.1 Struttura di un Perceptron

1. **Strato di Input:** Il percettrone riceve multiple caratteristiche di input binarie, spesso indicate come  $x_1, x_2, \dots, x_n$ .
2. **Pesi e Bias:** Ogni input viene moltiplicato per il suo peso corrispondente. I pesi rappresentano l'importanza degli input rispettivi nel processo decisionale del percettrone. Matematicamente, la somma pesata è calcolata come  $\sum_{i=1}^n w_i x_i$ . La somma pesata viene poi aggiunta a un termine di bias  $b$ , che rappresenta la tendenza del percettrone a fuoriuscire. Il bias sposta il limite decisionale del percettrone.

3. **Funzione di Attivazione:** La somma pesata  $z$  viene passata attraverso una funzione di attivazione. Lo scopo della funzione di attivazione è introdurre non linearità nel sistema.
4. **Output:** L'output della funzione di attivazione è l'output del perceptron, spesso indicato come  $y$ .



**Figure 6.20.** Basic architecture of a perceptron.

Figure 21: Crediti : Tan, P.-N., Steinbach, M., Kumar, V. (Year). Introduction to Data Mining (2nd ed.). Publisher.

#### 8.4.2 Apprendimento del Perceptron

I perceptron imparano attraverso un processo chiamato **apprendimento del perceptron** o **algoritmo di addestramento del perceptron**. L'obiettivo è regolare i pesi e il bias in modo che il perceptron possa classificare correttamente i pattern di input. Il ruolo della **loss function** è cruciale in questo processo, poiché misura la discrepanza tra le previsioni del perceptron e le etichette effettive. Questo guida gli aggiustamenti di pesi e bias per migliorare l'accuratezza della classificazione. L'obiettivo è minimizzare la loss function per migliorare le prestazioni del modello.

---

#### Algorithm 1 Algoritmo di Apprendimento del Perceptron

---

```

0: Sia  $D_{\text{train}} = \{(x_i^{\sim}, y_i) \mid i = 1, 2, \dots, n\}$  l'insieme di istanze di addestramento.
0: Imposta  $k \leftarrow 0$ .
0: Inizializza il vettore dei pesi  $\mathbf{w}^{\sim(0)}$  con valori casuali.
0: repeat
0:   for ogni istanza di addestramento  $(x_i^{\sim}, y_i)$  in  $D_{\text{train}}$  do
0:     Calcola l'output previsto  $\hat{y}^{(k)}$  usando  $\mathbf{w}^{\sim(k)}$ .
0:     for ogni componente del peso  $w_j$  do
0:       Aggiorna il peso:  $w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}^{(k)})x_{ij}$ .
0:     end for
0:   end for
0:   Aggiorna  $k \leftarrow k + 1$ .
0: until  $\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}^{(k)})$  è inferiore a una soglia  $\gamma = 0$ 

```

---

Fonte: Tan, P.-N., Steinbach, M., Kumar, V. (Year). Introduction to Data Mining (2nd ed.). Publisher.

#### 8.4.3 Limiti

Il Perceptron è un classificatore lineare, questo significa che è in grado di apprendere funzioni non linearmente separabili, come ad esempio la XOR function.

## 8.5 Multilayer Neural Network

I limiti del Perceptron ci spingono a cercare una soluzione per le funzioni non linearmente separabili. Per affrontare il problema, iniziamo ad aggiungere strati alla rete neurale, complicandola e rendendola in grado di catturare relazioni più complesse nei dati. Questo approccio, noto come Multilayer Neural Network, introduce uno strato intermedio o più strati intermedi tra lo strato di input e quello di output. Aggiungendo strati nascosti, la rete può apprendere pattern sempre più astratti e complessi permettendo di modellare anche problemi non linearmente separabili.

### 8.5.1 Struttura di una Multilayer Neural Network

Esaminando una rete neurale "tipica", possiamo concettualmente dividere la sua struttura in tre parti: Strato di Input, Strati Nascosti e Strato di Output.

#### 1. Strato di Input

Lo strato di input è il primo strato di una rete neurale e contiene un numero di neuroni pari al numero di features nel dataset. Ogni neurone nello strato di input rappresenta una specifica feature del dataset. È importante notare che lo strato di input non esegue alcun calcolo delle informazioni; piuttosto, la sua funzione principale è trasmettere i dati al primo strato nascosto.

Nel processo di trasmissione, i valori delle feature vengono moltiplicati dai rispettivi pesi associati alle connessioni tra i neuroni dello strato di input e quelli del primo strato nascosto. Questi pesi rappresentano l'importanza relativa di ciascuna feature nel contribuire alle attivazioni dei neuroni nel successivo strato nascosto.

#### 2. Strato Nascosto

Lo strato nascosto è il nucleo di una rete neurale. È composto da un numero arbitrario di neuroni, ognuno dei quali riceve valori di input dallo strato di input. Ogni neurone nello strato nascosto calcola una combinazione lineare dei valori delle feature e dei rispettivi pesi associati alle connessioni tra i neuroni dello strato di input e quelli dello strato nascosto.

Questa combinazione lineare viene quindi passata attraverso una funzione di attivazione, introducendo non linearità nel sistema. L'output della funzione di attivazione viene infine passato al neurone del successivo strato nascosto, e così via fino allo strato di output. Il numero di strati nascosti e il numero di neuroni in ciascuno strato nascosto sono due iperparametri che devono essere scelti durante la progettazione della rete neurale.

#### 3. Strato di Output

Lo strato di output è l'ultimo strato di una rete neurale. La sua composizione dipende dalla natura del problema da risolvere:

- (a) Per problemi di classificazione, lo strato di output consiste in un numero di neuroni pari al numero di classi nel dataset. Ogni neurone rappresenta una classe specifica. Lo strato di output calcola una combinazione lineare dei valori delle feature e dei rispettivi pesi associati alle connessioni tra i neuroni dell'ultimo strato nascosto e quelli dello strato di output. Questa combinazione lineare viene quindi passata attraverso una funzione di attivazione, introducendo non linearità nel sistema. L'output della funzione di attivazione rappresenta la probabilità che l'input appartenga alla classe rappresentata dal neurone dello strato di output. Il neurone dello strato di output con la probabilità più alta determina la classe predetta dall'algoritmo.
- (b) Per problemi di regressione, dove l'obiettivo è prevedere un valore continuo, lo strato di output consiste tipicamente in un singolo neurone. L'output è il risultato di una combinazione lineare dei valori delle feature e dei rispettivi pesi associati alle connessioni tra i neuroni dell'ultimo strato nascosto e il neurone di output. In questo caso, non viene applicata alcuna funzione di attivazione, e l'output rappresenta la previsione continua per il problema di regressione.

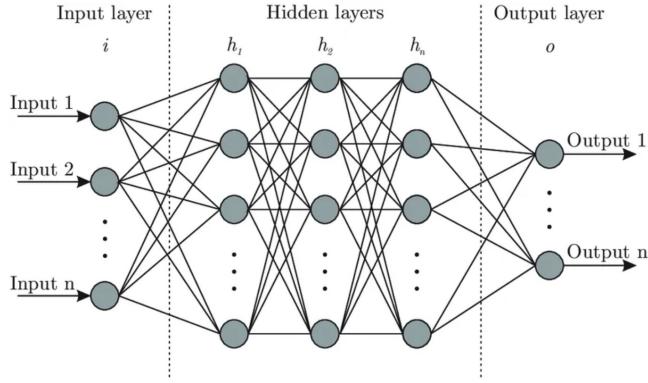


Figure 22: Crediti: Fonte

### 8.5.2 Il Processo di Apprendimento

La chiave dell'apprendimento profondo è fornire alla rete un insieme di istanze di addestramento ( $(X, y)$ ) e far predire alla rete una certa etichetta  $\hat{y}$ . Inizialmente, i pesi della rete saranno casuali e la rete si comporterà piuttosto male sui dati di addestramento. Pertanto, sarà necessario definire una funzione di costo, un modo per comunicare alla rete quanto si sta sbagliando.

Tuttavia, dire al computer quanto sia cattiva la sua previsione non aiuterà a migliorare. Infatti, per ottenere miglioramenti, sarà necessario modificare i pesi della rete. Possiamo immaginare la funzione di costo come una funzione che prende in input tutti i pesi e i bias della rete, e restituisce in output un numero che indica quanto si sta sbagliando. Il nostro obiettivo è minimizzare questa funzione, ma non è sempre possibile farlo solo attraverso metodi analitici. Quindi ci chiediamo: in quale direzione dovrei modificare i pesi per rendere la funzione di costo più piccola? Il gradiente è il vettore che ci indica la direzione in cui dovremmo modificare i pesi per rendere la funzione di costo più piccola.

L'algoritmo della discesa del gradiente è un algoritmo iterativo che ci permette di trovare il minimo di una funzione. Pensate alla discesa del gradiente come all'esplorazione di colline. La funzione di costo è come una superficie collinare, dove ogni punto rappresenta valori di parametri diversi. Parti da un punto, guardi intorno e fai un piccolo passo nella direzione più ripida verso il basso, avvicinandoti a una valle. Ripeti questo processo finché non raggiungi un minimo locale, il punto più basso in una valle.

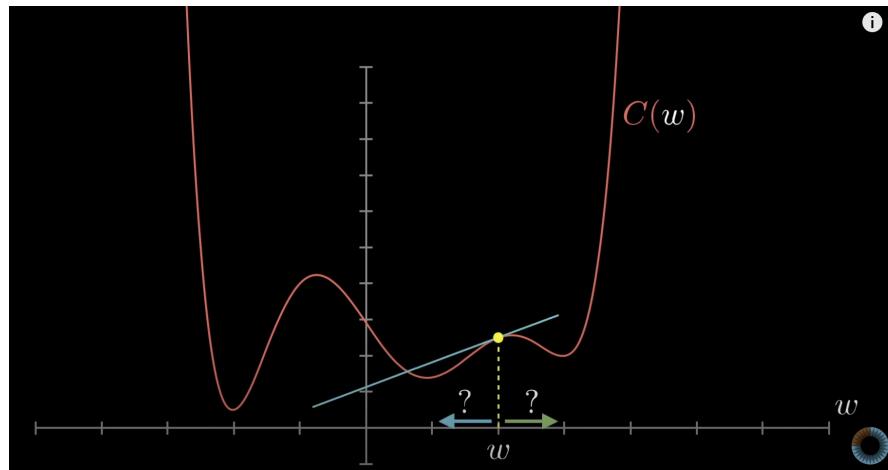


Figure 23: Illustrazione Discesa del Gradiente, Crediti: Fonte

È importante notare che alcune funzioni di costo possono avere più minimi locali. Durante

l'esecuzione della discesa del gradiente, l'algoritmo converge al minimo locale più vicino in base ai valori iniziali dei parametri. Se viene scelto un punto di partenza diverso, l'algoritmo può convergere a un diverso minimo locale. Di conseguenza, trovare un minimo assoluto non è garantito. Ecco come funziona il processo di apprendimento:

---

#### **Algorithm 2** Processo di Apprendimento

---

- 0: Inizializzazione casuale dei pesi della rete. Questi pesi rappresentano i parametri che la rete imparerà durante l'addestramento.
  - 0: Addestramento della rete per un numero predeterminato di epoches:
  - 0: **for** ogni istanza di addestramento  $(X, y)$  **do**
  - 0:   Calcola l'output predetto  $\hat{y}$  della rete per l'input  $X$ .
  - 0:   Calcola l'errore della rete confrontando  $\hat{y}$  con l'etichetta reale  $y$ .
  - 0:   Backpropagation: Calcola l'errore locale per ogni nodo di output e propaga l'errore all'indietro attraverso la rete, calcolando il gradiente della funzione di perdita.
  - 0:   Aggiornamento dei pesi: Utilizza un algoritmo di ottimizzazione (ad esempio, la discesa del gradiente) per aggiornare i pesi.
  - 0: **end for**=0
- 

La backpropagation è l'algoritmo che calcola il gradiente della funzione di perdita che indica la direzione in cui dovremmo modificare i pesi per rendere la funzione di costo più piccola. Forse pensare alla direzione in uno spazio di migliaia di dimensioni non è così intuitivo; possiamo invece pensare alla backpropagation come all'algoritmo per determinare come un singolo esempio di addestramento cambierebbe i pesi e i bias della rete.

## 8.6 Ottimizzatore

L'obiettivo principale dell'ottimizzatore è trovare i valori dei pesi che minimizzano la funzione di perdita, la quale quantifica l'errore tra le predizioni del modello e i valori attesi (ground truth).

### 8.6.1 Learning Rate

Il "learning rate" (tasso di apprendimento) è un parametro fondamentale nell'addestramento dei modelli di machine learning. È un valore numerico che determina quanto dovrebbero essere aggiornati i pesi del modello in base all'errore durante il processo di addestramento.

Un learning rate troppo basso può causare un addestramento lento del modello, mentre un learning rate troppo alto può far sì che il modello non converga o salti sopra il minimo globale. La scelta di un learning rate appropriato è fondamentale per garantire un addestramento efficiente e stabile del modello.

## 8.7 Loss Function

È il metodo di valutazione del modello. Misura l'errore tra la predizione della rete e l'output desiderato. Il tipo di loss function dipende dal task specifico. In regressione usiamo Mean Squared Error, mentre in classificazione usiamo la Categorical-CrossEntropy e in classificazione binaria la Binary Cross-entropy.

## 8.8 Overfitting & Best Practice

Delle buone performance sul training set non significano necessariamente buone performance con il test set. Il modello potrebbe aver "overfittato" sui dati di training imparando "a memoria" il training set.

Utilizziamo quindi il validation set, che viene utilizzato non per imparare i pesi, ma per misurare la qualità del modello.

Se le performance migliorano sul training ma peggiorano sul validation potrebbe essere che:

1. **Mismatch tra rete e dati:** Il modello è troppo complesso, e i dati non sono sufficienti per fare un buon tuning dei parametri.

2. **Mismatch tra training e validation:** il validation è troppo diverso dal training, quindi è come se ci stessimo allenando su qualcosa che non verrà “validato”.

Se non riusciamo a semplificare il modello, possiamo aumentare la quantità dei dati, attraverso un processo chiamato **data augmentation**.

L’idea del data augmentation è quella di generare dati **ragionevoli** in modo artificiale.

### 8.8.1 Dropout

Il dropout è una strategia utile per migliorare l’apprendimento di reti neurali complesse. Consiste nell’introdurre un tipo di rumore particolare nel processo di addestramento. Il dropout altera la trasmissione degli output di un layer al layer successivo, spegnendo alcuni neuroni in modo casuale. Ciò costringe la rete a trovare un modo per combinare i segnali dei neuroni rimanenti, migliorando così la generalizzazione del modello. Questa tecnica viene solitamente inserita prima dell’ultimo dense layer. È necessario specificare una probabilità per i neuroni da disattivare (ad esempio, 0.25 o 0.50), e non ha effetto durante il test. In generale, l’utilizzo del dropout richiede un numero maggiore di epoche di training per allenare la rete.

### 8.8.2 Transfer Learning

Il transfer learning è un processo utile quando l’allenamento di una rete neurale richiede una potenza computazionale elevata. In alcuni casi, non abbiamo la capacità computazionale necessaria per allenare una rete da zero. L’idea di base è che anziché partire da pesi casuali, possiamo utilizzare i pesi di una rete neurale già allenata su un task simile e poi aggiustarli per il nuovo task. Questo processo ci consente di risparmiare risorse computazionali e di velocizzare il processo di apprendimento. Possiamo anche utilizzare solo alcune componenti di un’altra rete per costruire la nostra rete, come ad esempio mantenere e sfruttare l’architettura per l’estrazione delle feature e personalizzare i layer densi successivi. Questo approccio è particolarmente utile in ambiti in cui i pattern riconosciuti dai primi layer della rete sono simili tra task diversi.

## 9 CNN - Convolutional Neural Network

Le reti neurali tradizionali non hanno percezione spaziale.

Le Convolution Neural Network provano a fare meglio delle reti neurali tradizionali, ma per fare ciò complicano l'architettura servendosi di componenti aggiuntive:

**Strato di Convoluzione** Coinvolgono l'applicazione di **filtri convoluzionali** ai dati di input.

Questi filtri sono ideati per estrarre caratteristiche locali dai dati bidimensionali. Questi strati sono fondamentali per riconoscere **pattern spaziali** nelle immagini.

**Strato di Pooling** Gli strati di Pooling **riducono la dimensione spaziale** dell'output dei filtri convoluzionali. Contribuendo nella riduzione della complessità computazionale, e migliorando la generalizzazione, mantenendo le caratteristiche più rilevanti.

**MaxPooling** e **AveragePooling** sono i tipi di pooling più comuni.

**Strato di Flattering** Una volta estratte le feature attraverso gli strati convoluzionali, l'output sarà del tipo: width x height x n° channels, esso dunque viene appiattito in un vettore unidimensionale grazie ad un flatten layer. I dati unidimensionali saranno dati in pasto ad una rete Fully Connected, che si occuperà del task di classificazione.

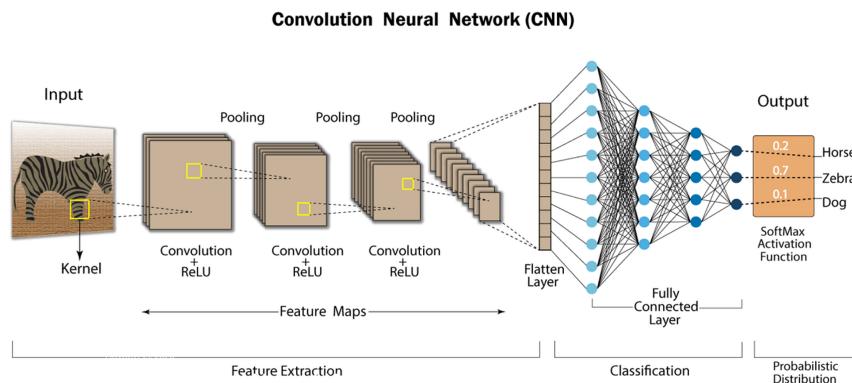


Figure 24: Struttura di una CNN, Crediti: Fonte

### 9.1 Calcolo Matrice Di Convoluzione

Per calcolare una matrice di convoluzione, segui questi passaggi:

1. Definisci la dimensione della matrice di convoluzione. Di solito è una matrice quadrata con dimensioni dispari come 3x3 o 5x5.
2. Assegna i valori dei pesi ai singoli elementi della matrice di convoluzione. Questi pesi rappresentano i filtri che verranno applicati ai dati di input.
3. Posiziona la matrice di convoluzione sul dato di input. Inizia dal primo elemento del dato di input e moltiplica i valori corrispondenti della matrice di convoluzione con i valori corrispondenti del dato di input.
4. Somma i risultati delle moltiplicazioni ottenute nel passaggio precedente. Questa somma rappresenta il valore che verrà assegnato al pixel corrispondente nella matrice di output.
5. Sposta la matrice di convoluzione al prossimo elemento del dato di input e ripeti i passaggi 3 e 4 fino a quando non hai coperto tutto il dato di input.
6. Ripeti i passaggi precedenti per ciascun canale di colore nel caso di immagini a colori.

È importante notare che il calcolo della matrice di convoluzione può variare in base ai parametri specifici utilizzati, come il passo della convoluzione e il riempimento dei bordi.

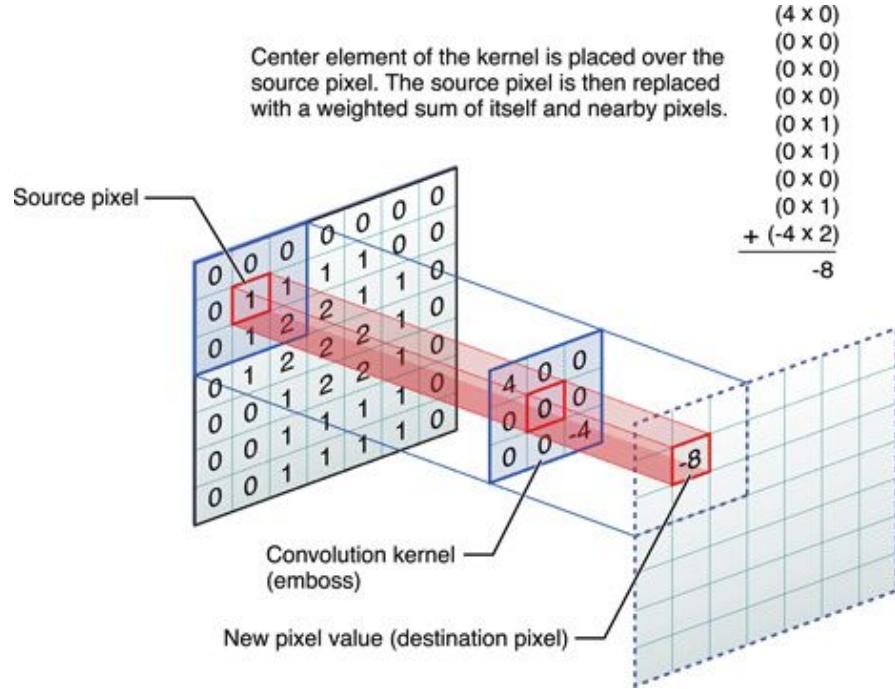


Figure 25: Crediti: Fonte

#### Alcune Note:

1. I filtri sono applicati pixel by pixel.
2. Data una matrice, l'applicazione del filtro ritorna una matrice della stessa grandezza.
3. I filtri sono generalmente più piccoli in confronto all'immagine.
4. L'output del filtro viene chiamato **risposta**.
5. I valori dei filtri, sono nuovi pesi/parametri all'interno della rete, aggiustati dal processo di apprendimento.
6. Ci sono diversi modi per processare punti fuori dai bordi, come ad esempio ripetere il valore dei bordi.

## 9.2 Pooling

Quello che fa il pooling è prendere una certa area (i.e: 2x2) di pixel del response e mapparla in un unico pixel. Il modo in cui viene mappata dipende dal tipo di pooling che usiamo. Ad esempio: Data una **response** 4x4, se applico MaxPooling 2x2, dimezzerò la dimensione della matrice, mappando 2x2 pixel in un unico pixel che in questo caso rappresenterà il massimo.

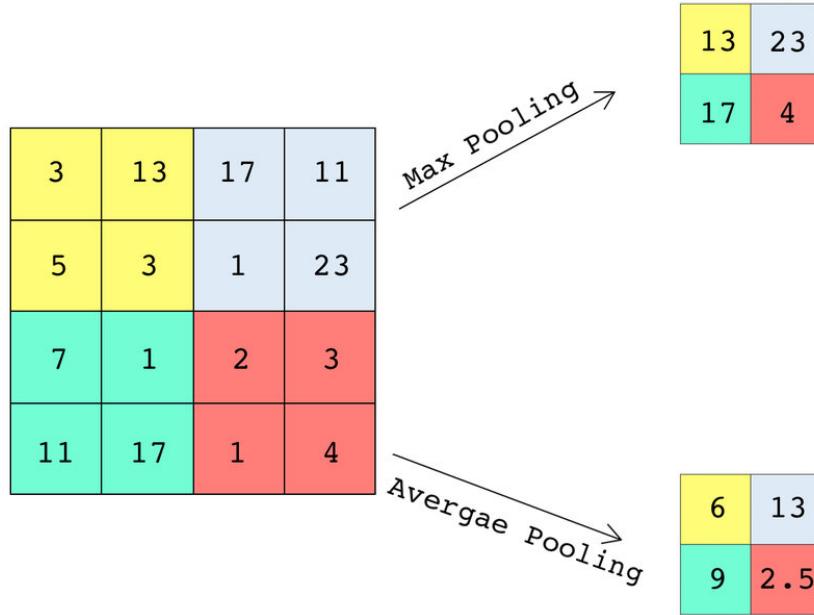


Figure 26: Confronto: Max Pooling e Average Pooling, Crediti: Fonte

### 9.3 Proprietà

1. I filtri convoluzionali sono *translation invariant*: Se il filtro trova un pixel con una risposta alta, troverà lo stesso pixel anche se l'immagine è traslata.
2. Le CNN imparano pattern spaziali gerarchici:
  - (a) I layer alti (vicino l'output) combinano i layer più bassi in possibili strutture spaziali.
  - (b) Possiamo pensare ai filtri come a delle funzioni che ritornano la presenza di pattern.

### 9.4 1-D Convolution

Una convoluzione 1D genera anch'essa features spazio—temporali. Questo tipo di applicazione può essere utile per analizzare le serie temporali.

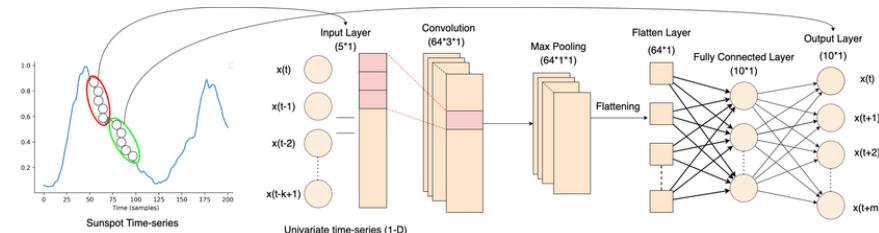


Figure 27: Crediti: Fonte

### 9.5 Funnel or Information Compression

Una cascata di layer con un numero decrescente di output crea il cosiddetto effetto a imbuto, in cui l'informazione proveniente dal layer precedente viene in qualche modo compressa in un numero più piccolo di output. I layer intermedi dovrebbero essere sufficientemente grandi da catturare tutte le informazioni significative, se ciò non dovesse accadere si parla di “information bottleneck”. In un problema di classificazione con N classi, avere n layer intermedi con  $n < N$  nodi porterà molto probabilmente ad un collo di bottiglia, poiché n nodi non includono sufficienti informazioni per distinguere tra le N classi.

## 10 Data Mining Methodology

Il Data Mining è una disciplina che si occupa dell'estrazione di conoscenza implicita, precedentemente sconosciuta e potenzialmente utile, dai dati. La metodologia del Data Mining è un processo strutturato che guida l'analisi dei dati per rivelare pattern, tendenze e relazioni significative. In questa sezione, esploreremo la metodologia del Data Mining e le sue fasi principali. Vedremo come vengono raccolti, preparati e analizzati i dati per estrarre informazioni preziose che possono essere utilizzate per prendere decisioni informate e guidare il processo decisionale aziendale.

### 10.1 Metodologia

Non si parte da una tabella con dei dati, ma solitamente si parte da un fenomeno da analizzare, provando a capire se metodi di machine learning possono essere utili.

- Valutazione, cosa significa? Avrò diverse soluzioni, e come faccio a valutare le qualità delle mie soluzioni?
- Quale algoritmo di machine learning? Cosa è più adatto per il problema che stiamo affrontando?
- I dati, abbiamo dei dati? i dati sono puliti?

### 10.2 Fasi della Metodologia

La metodologia del Data Mining comprende generalmente le seguenti fasi:

1. **Comprensione del Dominio:** In questa fase, è essenziale comprendere il contesto del problema e definire gli obiettivi del Data Mining.
2. **Raccolta dei Dati:** Vengono raccolti i dati rilevanti per l'analisi, che possono provenire da diverse fonti come database, file di testo, sensori, e così via.
3. **Preparazione dei Dati:** I dati vengono puliti, trasformati e ridimensionati per prepararli all'analisi. Questa fase è cruciale per garantire la qualità dei dati e la loro adattabilità agli algoritmi di Data Mining.
4. **Modellazione:** Sono applicati gli algoritmi di Data Mining per identificare pattern e relazioni nei dati. Questa fase coinvolge l'addestramento e la valutazione dei modelli.
5. **Valutazione:** I modelli sono valutati per determinare la loro accuratezza e affidabilità. Questo passaggio aiuta a garantire che i risultati del Data Mining siano validi e utili.
6. **Deploy e Monitoraggio:** I modelli validati vengono implementati in ambienti operativi e monitorati per garantire che continuino a fornire risultati accurati nel tempo.

### 10.3 Struttura dell'Istanza

La struttura di un'istanza potrebbe includere i seguenti elementi:

- Documento
- Query
- Termini in comune
- Data di upload
- Numero di slash nella URL (le sottopagine sono generalmente considerate meno importanti delle pagine principali)
- Numero di click su una coppia

Ad ogni query viene associato uno score che si assegna alla pagina. Il modello è la funzione di score stessa, che determina la bontà di un documento in funzione di una query.

L'istanza, intesa come la coppia query-documento, può essere affrontata sia come un problema di regressione dello score, sia come un problema di classificazione della rilevanza di una query per un documento (ad esempio, determinando se la rilevanza è buona o non buona).

Data la mole di documenti presenti in un database, analizzarli tutti può essere un processo dispendioso. Pertanto, si sfruttano gli indici dei database, utilizzando una parola presente nella query come filtro per l'indice.

Il processo può essere suddiviso in due fasi:

1. **Matching**: selezione dei documenti pertinenti
2. **Machine Learning**: applicazione di tecniche di apprendimento automatico per migliorare la precisione della ricerca

## 10.4 Label

Abbiamo parlato delle istanze di un training set, ma per allenare il modello servono anche le  $y$ . Quello che si fa è andare in cerca del feedback utente implicito, estraendo le etichette di rilevanza dal comportamento dell'utente.

## 10.5 Valutazione

Supponiamo di aver allenato un modello che prende una query e un documento, e il modello restituisce uno score.

Come faccio a dire se il mio modello è buono oppure no?

**Prima idea** Prendiamo un classificatore e misuriamone l'accuracy. Ha senso usare l'accuracy come misura di bontà? Può accadere che due modelli abbiano la stessa accuracy, ma comunque uno sia preferibile rispetto all'altro. In generale, la misura di bontà dovrebbe tener conto del contesto di applicazione.

Esempio, misura di qualità nel ranking, deve tener conto di:

- Posizione nella lista finale
- Label

Una tipica misura utilizzata è il Discounted Cumulative Gain.

## 10.6 Algoritmi

Cerchiamo un algoritmo che massimizzi l'NDCSG@K. In realtà si ottimizza un'approssimazione di NDCSG@K, e cerco un algoritmo che ottimizzi l'approssimazione. Nota bene che: Ottimizzare l'approssimazione non significa ottimizzare l'NDCSG@K e ottimizzare l'NDCSG@K non significa ottimizzare la qualità percepita dall'utente, quindi di fatto ci siamo allontanati dal problema reale. (Il classificatore non permette di ottimizzare l'NDCSG@K)

Perché è difficile ottimizzare? Questa formula ha un ordinamento nel mezzo e questo fa sì che la derivata prima non si possa calcolare.

# 11 Unsupervised Learning

L'apprendimento non supervisionato è un ambito dell'intelligenza artificiale che si concentra sull'analisi di dati non etichettati. In questo contesto, due approcci principali emergono come fondamentali: *Cluster Analysis* e *Associative Rules*.

## 11.1 Cluster Analysis

La *Cluster Analysis*, o analisi dei cluster, è una tecnica che mira a suddividere un insieme di dati in gruppi omogenei chiamati "cluster". L'obiettivo è far sì che gli elementi all'interno di ciascun cluster siano più simili tra loro rispetto a quelli in altri cluster. Questa tecnica è ampiamente utilizzata per esplorare la struttura intrinseca dei dati e identificare pattern nascosti. Alcuni degli algoritmi comuni utilizzati per la clusterizzazione includono il *K-Means*, il *Hierarchical Clustering* e il *DBSCAN*.

### 11.1.1 Approcci nel Clustering

Esistono diversi aspetti chiave che possono variare nei diversi approcci di clustering:

1. Partizionamento singolo o gerarchico, riguarda la struttura dell'output del clustering.
2. Separazione tra cluster: esclusiva o non-esclusiva, riguarda la natura dell'appartenenza dei dati ai cluster
3. Full Space o Subspace, riguarda il numero di dimensioni considerate durante il processo di clustering
4. Misura di similarità, riguarda il criterio che definisce la similarità dei punti dati

## 11.2 Associative Rules

Le *Associative Rules*, o regole associative, sono un altro aspetto dell'apprendimento non supervisionato. Questo approccio mira a identificare relazioni interessanti tra le variabili in un insieme di dati. Un esempio classico è l'algoritmo *Apriori*, spesso utilizzato per scoprire regole di associazione in dataset transazionali, come i modelli di acquisto nei negozi al dettaglio. Le regole associative possono fornire insights su collegamenti e comportamenti nei dati, aprendo la strada a decisioni informate.

## 12 Clustering Algorithm

### 12.1 Cosa si intende con Algoritmo di Clustering

**Definizione di Cluster:**

Un cluster è un insieme di oggetti molto simili tra di loro e dissimili da altri oggetti in altre classi.

**Definizione di Algoritmo di Clustering:**

Un algoritmo di clustering è un algoritmo utilizzato per assegnare un insieme di dati in gruppi omogenei detti cluster, in base a dei criteri di similarità o dissimilarità.

#### 12.1.1 Proprietà desiderate di un algoritmo di Clustering

Quali sono le proprietà che ricerchiamo in un algoritmo di Clustering?

1. Scalabilità
2. Abilità nel gestire tipi di attributi diversi
3. Capacità di gestione dei vincoli
4. Interpretabilità e usabilità
5. Scoperta di cluster di forma arbitraria
6. Clustering incrementale e insensibile all'ordine di input
7. Capacità di gestire il rumore dei dati

In generale è difficile trovare un algoritmo di clustering che soddisfi tutte le proprietà, perciò la scelta dell'algoritmo dipende molto dalla natura del problema da analizzare.

#### 12.1.2 Classificazione Algoritmi di Clustering

Questa sezione enumera vari algoritmi di clustering, più avanti metteremo in luce le loro caratteristiche e applicazioni specifiche. In generale, la classificazione degli algoritmi è un mezzo essenziale per differenziare tra approcci e strategie diverse utilizzate per raggruppare i dati in cluster omogenei. Questa suddivisione organizzativa agevola la scelta dell'algoritmo più idoneo per situazioni o problemi specifici, fornendone aiuto agli utenti nella scelta dell'opzione più adatta tra le molteplici opzioni disponibili. Classifichiamo gli algoritmi di clustering in:

1. Partitioning Clustering: Algoritmi che partizionano il dataset in sottoinsiemi diversi.
2. Hierarchical Clustering: Algoritmi che ci dicono se due cluster sono “vicini” e possono essere uniti in unico cluster.
3. Density-Based Clustering: Algoritmi che inducono cluster se una zona è “ricca di punti”.
4. Model-Based Clustering: L’idea è quella di ipotizzare un modello per ogni cluster, e si cerca di trovare il miglior adattamento di quel modello tra loro.
5. Grid-Based Clustering
6. Frequent Pattern-Based Clustering
7. Constraint-Based Clustering
8. Linked-Based Clustering

## 12.2 Partitioning Clustering

Il partitioning clustering è una categoria di algoritmi di clustering che cerca di soddisfare i seguenti requisiti:

1. Ogni cluster contiene almeno un punto
2. Ogni punto appartiene esattamente a un gruppo

Questo tipo di algoritmi solitamente cerca di minimizzare una funzione obiettivo.

$$\text{minimizzare: } E = \sum_{i=1}^k \sum_{p \in D} [distance(p, center(i))^2]$$

Alcuni algoritmi di partitioning clustering sono: k-means e k-medoids.

### 12.2.1 K-Means

Il K-Means è un algoritmo di clustering partitivo che punta a minimizzare:

$$\text{minimize } E = \sum_{i=1}^k \sum_{p \in C_i} \|p - center_i\|^2$$

da cui:

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

#### Algoritmo

```
1 def K_means(D, k):
2     seleziona casualmente k punti casuali come centroidi iniziali
3     while ( i Cluster continuano a cambiare o La funzione    maggiore di
4           una soglia t)
5         assegna ogni istanza al cluster con il centroide pi    vicino
6         Calcola i centroidi dei cluster della partizione corrente
```

Listing 2: Illustrazione Algoritmo K-Means, Complessità  $O(tkn)$

#### Vantaggi e Svantaggi

##### Vantaggi:

- Efficienza: È un algoritmo notevolmente veloce, superando molti altri algoritmi di clustering.
- Adattabilità: Ottimo per identificare cluster sferici, di dimensioni simili e distribuiti uniformemente.

##### Svantaggi:

- Specificità di  $k$ : Richiede la predefinizione del numero di cluster  $k$ , il che può essere un limite in alcune situazioni.
- Limitazioni sulla forma: Non è ideale per rilevare cluster con forme non sferiche.
- Sensibilità a rumori e anomalie: La presenza di dati anomali o punti lontani può influenzare significativamente i risultati.
- Spazialità: Applicabile solo a oggetti in uno spazio continuo  $n$ -dimensionale.
- Assorbimento di cluster: Cluster più piccoli potrebbero essere assimilati da quelli più grandi, influenzando la precisione.

## K-Means++

Abbiamo visto come K-Means sia sensibile alla scelta iniziale dei centroidi. Può accadere che durante la scelta dei centroidi, più di essi provengano dallo stesso cluster. K-Means++ suggerisce una strategia per la scelta iniziale dei centroidi. L'idea è che i centroidi iniziali siano ben separati e coprano bene i dati. Una volta selezionati i centroidi, l'algoritmo procede come K-Means.

## Elbow-Method

Il Sum of Square Errors (SSE) decresce all'aumentare di  $k$ , ma vorremmo tenere un  $k$  ragionevolmente piccolo. L'idea è di fermare il numero di  $k$  quando si ottiene un miglioramento piccolo del SSE.

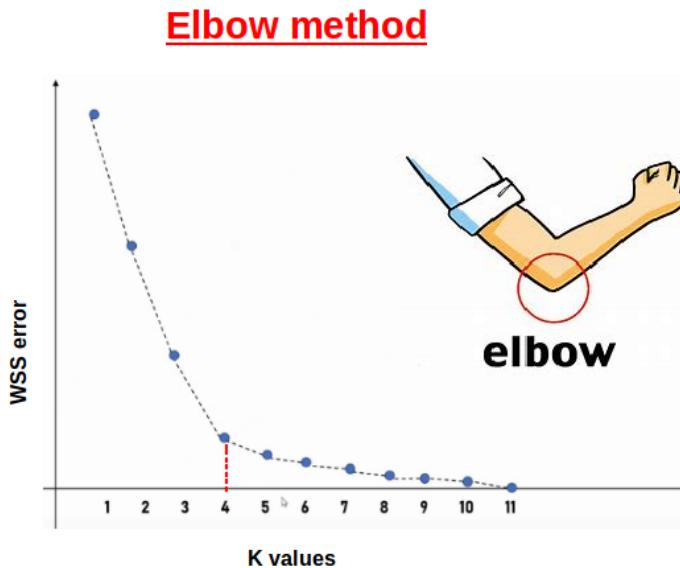


Figure 28: Elbow Method. Crediti: Fonte

### 12.2.2 K-Medoids

E' una variante del K-Means, invece che minimizzare la distanza tra un istanza  $p$  e un centroide, minimizza la distanza tra un istanza  $p$  e un medoide. Cioè, invece che computare centri, computiamo istanze del dataset come centri.

#### Algoritmo

```
1 def PAM(D, k):
2     seleziona casualmente k punti casuali come medoidi iniziali
3     while ( i Cluster continuano a cambiare o La funzione      minore di una
4             soglia t)
5         assegna ogni istanza al cluster con il medoide pi    vicino
6         for o_j, o_other in D:
7             if replacing o_j with o_other reduces E:
                    update medoids
```

#### Vantaggi e Svantaggi

##### Vantaggi:

- Rispetto al k-means è più robusto agli outliers

## Svantaggi:

- Computazionalmente più costoso, poiché calcolare la mediana in uno spazio n-dimensionale è più costoso.

### 12.2.3 Fuzzy C-Means & Partizionamento Parziale

#### Partizionamento Parziale

Delle volte l'assunzione di partizionamento è fin troppo forte, potremmo perciò avere un grado di appartenenza (**membership degree**) in un dato cluster.

Dato un dataset  $D$  con  $m$  istanze e un insieme di  $k$  cluster, diciamo che  $w_{ij}$  è il **grado di appartenenza di un'istanza**  $x_i$  al cluster  $C_j$ .

1. Per ogni istanza  $x_i$  nel cluster, la somma dei pesi  $w_{ij}$  è uguale a 1:  $\forall x_i : \sum_{j=1}^k w_{ij} = 1$
2.  $\forall C_j : 0 < \sum_{i=1}^m w_{ij} < m$ . Questo contribuisce a garantire che ogni cluster abbia almeno un punto assegnato e impedisce che tutti i punti siano assegnati a un singolo cluster.

Il termine  $w_{ij}$  definisce un **fuzzy clustering**.

#### Fuzzy C-Means

Richiamiamo la funzione da minimizzare nel K-Means:

$$E = \sum_{j=1}^k \sum_{x_i \in C_j} \text{dist}(x_i, c_j)^2$$

Introduciamo il concetto di *membership*:

$$E = \sum_{j=1}^k \sum_{i=1}^m w_{ij}^p \cdot \text{dist}(x_i, c_j)^2$$

Dove  $p$  controlla la “Fuzzy-ness”, più cresce, più abbiamo fuzzy-ness. Usiamo  $p = 2$ . La **nuova formula** per calcolare i centroidi:

$$c_j = \frac{\sum_{i=1}^m w_{ij}^2 x_i}{\sum_{i=1}^m w_{ij}^2}$$

Se  $w_{ij} = 0$  o  $w_{ij} = 1$ , è equivalente a K-Means.

La **formula** per calcolare  $w_{ij}$ :

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

Nota: Il denominatore serve a normalizzare, così che sommino 1.

#### Algoritmo

```

1 def c_means(D, k):
2     1. inizializza k centroidi nel dataset D
3     2. while (DeltaE < t or centrodi stabili)
4         3.     assegno gli oggetti ai centroidi pi vicini
5         4.     calcolo centroidi della partizione corrente, con la nuova
            formula

```

## Vantaggi e Svantaggi

### Vantaggi:

- Quelli del k-means
- Produce un raggruppamento che fornisce un' indicazione del grado di appartenenza di un qualsiasi punto a un qualsiasi cluster

### Svantaggi:

- Quelli del k-means
- E' più costoso del k-means

## 12.3 Hierarchical Clustering

L' Hierarchical clustering è una categoria di algoritmi di clustering che si divide in due metodi principali:

1. Agglomerativo Inizialmente ogni oggetto forma il proprio cluster, attraverso una strategia bottom-up i cluster vengono uniti iterativamente fino a quando un singolo cluster diventa la radice della gerarchia.
2. Divisivo Inizialmente tutti gli oggetti formano un singolo cluster, attraverso una strategia top-down i cluster vengono suddivisi in cluster più piccoli.

Questo metodo di clustering utilizza una matrice di distanza/similarità:

- La **Distance Matrix**  $D$  ha in  $D[i, j]$  la distanza tra l'oggetto  $i$  e l'oggetto  $j$ ;
- La **Similarity Matrix**  $S$  ha in  $S[i, j]$  la similarità tra l'oggetto  $i$  e l'oggetto  $j$ .

Le operazioni di clustering producono un albero di clustering binario chiamato **dendrogramma**. La cui radice è la classe che contiene tutte le osservazioni. Rappresenta una gerarchia di partizioni, è quindi possibile scegliere una partizione troncando l'albero ad un determinato livello.

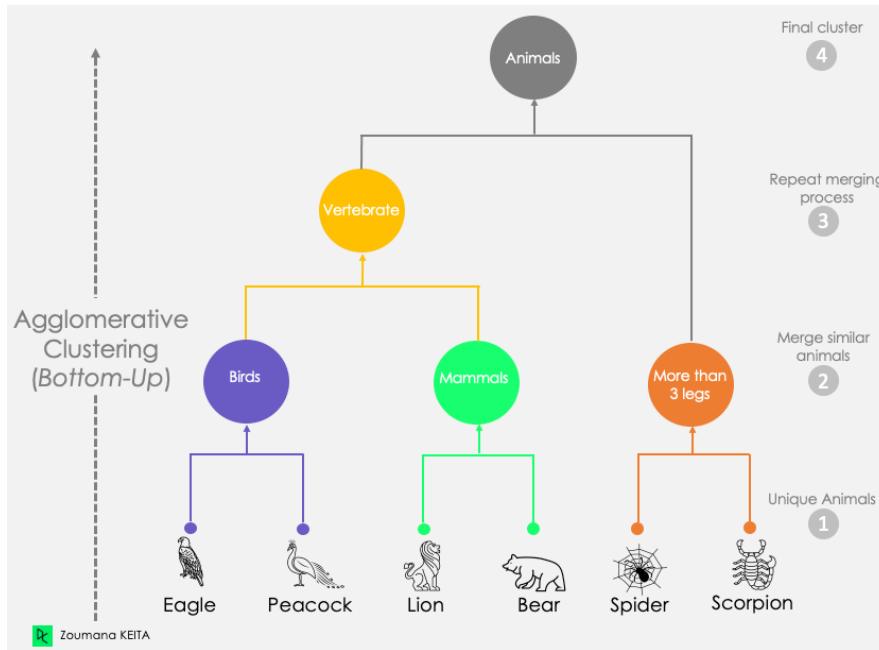


Figure 29: Dendrogramma. Crediti: Fonte

### 12.3.1 Stimare la Similarità tra Cluster - Misure di Linkage

Per effettuare una stima della similarità tra cluster, è essenziale scegliere una misura di linkage appropriata. Le principali misure di linkage includono:

1. **Single Linkage:** Questa misura è basata sulla distanza minima tra gli elementi dei due cluster. Tuttavia, può portare a sovrastimare la similarità e generare catene. La formula associata è  $dist(C_i, C_j) = \min dist(x, y)$ , con  $x \in C_i$  e  $y \in C_j$ .
2. **Complete Linkage:** In questo caso, si considera la distanza massima tra gli elementi dei due cluster. Questa misura tende a favorire la formazione di cluster globali. La formula è  $dist(C_i, C_j) = \max dist(x, y)$ , con  $x \in C_i$  e  $y \in C_j$ .

3. **Average Linkage:** Questa misura calcola la distanza media tra tutti gli elementi dei due cluster. La sua principale caratteristica è la robustezza ai valori anomali. La formula associata è  $dist(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum dist(x, y)$ , con  $x \in C_i$  e  $y \in C_j$ .
4. **Centroidi/Medoidi Linkage:** Questa misura è caratterizzata dalla velocità di calcolo. La distanza tra due cluster è definita come  $dist(C_i, C_j) = dist(o_i, o_j)$ , dove  $o_i$  e  $o_j$  rappresentano rispettivamente il centroide e il medoide dei cluster  $C_i$  e  $C_j$ .

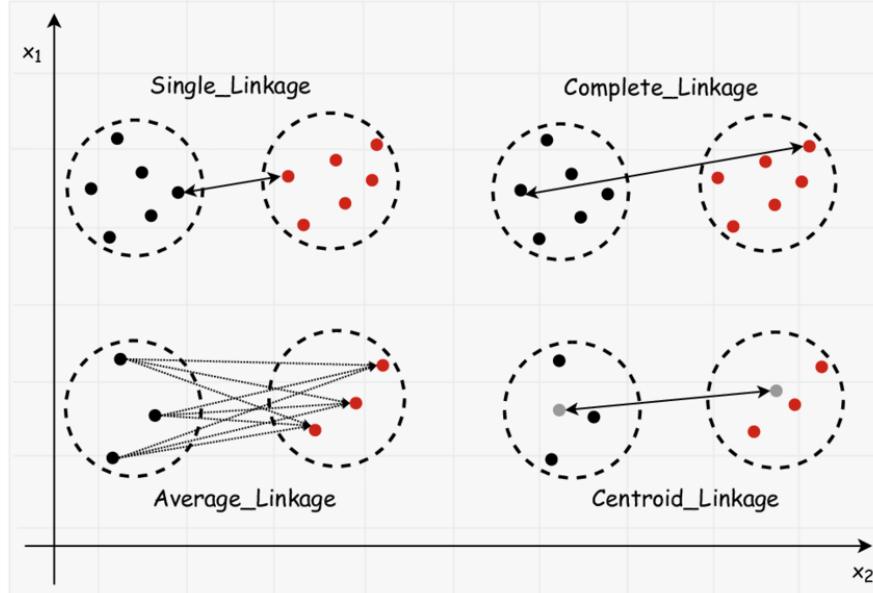


Figure 30: Linkage-Measure. Crediti: Fonte

Il single linkage è il più adattivo, potenzialmente un rischio, ma le altre misure tendono a cercare classi sferiche.

### 12.3.2 AHC - Agglomerative Hierarchical Clustering

#### Algoritmo

```

1 def AHC(Dt):
2     #initialize every single point as a cluster
3     for i = 1 to |Dt|
4         c_i = {o_i}
5         #set of Cluster
6         C = C U {c_i}
7     #aggregation loop
8     while |C| > 1:
9         #find closet cluster pair
10        c_i,c_j = arg_min_ci_cj(D[c_i,c_j])
11        c_new = c_i U c_j
12        C = { C \ {c_i} } \ {c_j} U {c_new}

```

#### Matrice di distanze "D"

Soltamente viene implementato attraverso una matrice di distanze D di grandezza  $|Dt| \times |Dt|$ :

1. Inizialmente:  $D[i, j] = dist(o_i, o_j) \quad \forall i, j$
2. Quando creiamo  $C_{new}$ :

- (a) La colonna e la riga che corrispondono a  $C_j$  sono invalidate. (+ inf)
- (b) La colonna e la riga che corrispondono a  $C_i$  sono utilizzate per  $C_{new}$ .

Nota: Se la matrice D è una matrice di similarità i dati nella matrice sono simmetrici.

### Complessità:

1. Caso "generico":  $O(n^3)$
2. Implementazione con mini-heap:  $O(n^2 \log(n))$
3. Usando single-linkage:  $O(n^2)$

### Vantaggi e Svantaggi

#### Vantaggi

1. Flessibile alla forma dei cluster

#### Svantaggi

1. Sensibile agli outliers

### 12.3.3 DHC - Divisive Hierarchical Clustering

#### Algoritmo

- ```

1. Inizialmente tutti i punti nel dataset appartengono ad un solo
   cluster
2. Partiziono il cluster in due cluster, i meno simili tra loro
3. Ad ogni iterazione il cluster pi eterogeno viene diviso in due
   cluster
4. Procede con le iterazioni fino a che tutti gli oggetti sono nel
   loro cluster

```

## 12.4 Clustering basato sulla Densità

Il clustering basato sulla densità è un approccio che mira a individuare concentrazioni di punti e aree vuote all'interno di uno spazio dati. I punti che non appartengono a nessun cluster vengono etichettati come rumore.

Per comprendere appieno il concetto di clustering basato sulla densità, è necessario introdurre alcuni concetti fondamentali:

### 12.4.1 Core Point " $p$ " e Vicinato di un Punto " $N_\epsilon(p)$ "

Un punto  $p$  è *core point* se soddisfa:

$$|N_\epsilon(p)| \geq \text{MinPts}$$

Dove:  $N_\epsilon(p) = \{q \in D | dist(q, p) \leq \epsilon\}$   
 $\epsilon$  e MinPts sono iperparametri da tunare.

### 12.4.2 Condizione per la Densità Reachability e Density Connectivity

Per stabilire quali punti dovranno appartenere allo stesso cluster di  $p$ , vengono introdotti i concetti di *Density Reachability* e *Density Connectivity*. Questi concetti delineano le relazioni di densità tra i punti e sono fondamentali per il clustering basato sulla densità.

## Density Reachability

### 1. Direttamente densamente raggiungibile

$q$  è direttamente densamente raggiungibile da  $p$  ( $p$  è core point) se:  $q \in N_\epsilon(p)$

### 2. Densamente raggiungibile

Il punto  $q$  è **densamente raggiungibile** da  $p$  se esiste una catena di punti  $p_1, p_2, \dots, p_n$  tale che  $p_1 = p$  e  $p_n = q$ , e vale la **condizione**:  $p_{i+1}$  è direttamente densamente raggiungibile da  $p_i$ .

## Density Connectivity

$s$  è **densamente connesso** a  $r$  se  $\exists$  un punto  $o$  tale che  $s$  è densamente raggiungibile da  $o$  e  $r$  è densamente raggiungibile da  $o$ .

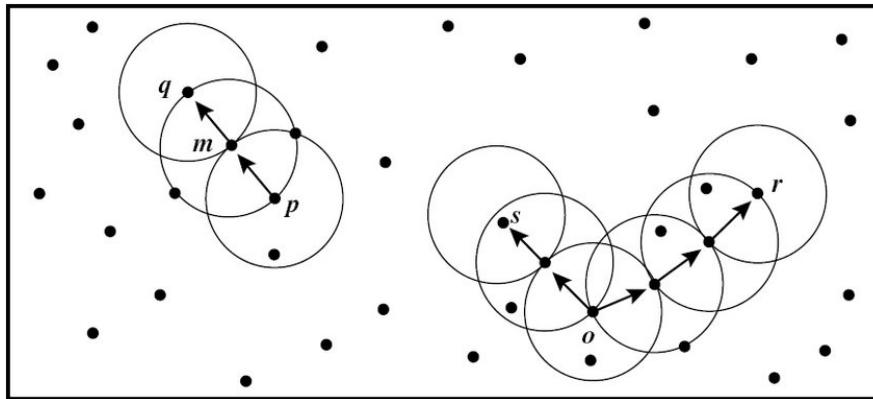


Figure 31: Density Reachability (left) and Density Connectivity(right). Crediti: Fonte

### 12.4.3 DB-Scan

Abbiamo visto come il clustering basato sulla densità si basa sulla rilevazione di regioni dense di punti, separando aree vuote e sparse. Attraverso i concetti di *Core Point*, *Neighbourhood*, *Density Reachability* e *Density Connectivity*, è possibile definire quali punti formano cluster significativi e identificare quelli che sono considerati rumore.

### Algoritmo

```

1: Label all points as core, border, or noise points.
2: Eliminate noise points.
3: Put an edge between all core points within a distance Eps of each
   other.
4: Make each group of connected core points into a separate cluster.
5: Assign each border point to one of the clusters of its associated
   core points.

```

### Complessità:

La complessità temporale di base dell'algoritmo DBSCAN è  $O(m \times$  tempo per trovare punti nel vicinato di  $Eps$ ), dove  $m$  è il numero di punti. Nel caso peggiore, questa complessità è  $O(m^2)$ . Tuttavia, in spazi a bassa dimensione (specialmente nello spazio 2D), strutture dati come gli alberi kd consentono il recupero efficiente di tutti i punti entro una determinata distanza da un punto specificato, e la complessità temporale può essere ridotta fino a  $O(m \log m)$  nel caso medio. Il

requisito di spazio di DBSCAN, anche per dati ad alta dimensionalità, è  $O(m)$  poiché è necessario conservare solo una piccola quantità di dati per ogni punto, ovvero l'etichetta del cluster e l'identificazione di ciascun punto come core, border o noise point.

### Vantaggi e Svantaggi

#### Vantaggi:

1. Gestisce il rumore in modo esplicito
2. Gestisce cluster di forma arbitraria
3. Non serve specificare il numero di cluster

#### Svantaggi:

1. Difficile trovare i giusti parametri per cluster con diversa densità

## 12.5 Model-Based Clustering

L'idea è quella di ipotizzare un modello per ogni cluster, e si cerca di trovare il miglior adattamento di quel modello tra loro.

### 12.5.1 Self-Organizing Map (SOM)

#### Obiettivo

L'obiettivo principale è quello di rappresentare i dati su una griglia bidimensionale in modo che i dati con caratteristiche simili siano vicini nella griglia.

#### Algoritmo

```
1: Inizializza Centroidi
2: repeat
3:   vai al prossimo oggetto e calcola il centroide pi vicino
4:   Aggiorna il centroide e tutti i centroidi -vicini-
5: until i centroidi sono stabili o una soglia stata superata
6: Assegna ad ogni oggetto il suo centroide pi vicino e ritorna
7: i centroidi e il cluster
```

L'algoritmo in questione rappresenta una generalizzazione avanzata del K-Means, arricchita da funzionalità di visualizzazione dei dati. In questa implementazione, ogni cella sulla mappa funge da neurone, ognuno associato a un vettore di pesi. L'obiettivo finale è ottenere un insieme di vettori di pesi in grado di assegnare a ogni oggetto il centroide che lo rappresenta al meglio.

Un elemento distintivo di questo approccio è la flessibilità topologica: le celle possono essere configurate non solo con i tradizionali lati quadrati, ma anche come esagoni. Ciò consente a ciascuna cella di avere più vicini immediati, ampliando le possibilità di analisi della distribuzione dei dati. Da sottolineare è l'applicabilità dell'algoritmo sia per hard clustering che per soft clustering, consentendo una vasta gamma di utilizzi in diversi contesti. La combinazione di precisione nell'assegnazione dei centroidi e la capacità di visualizzare i dati rende questo algoritmo una risorsa potente e versatile per l'analisi e la comprensione dei modelli nei dati complessi.

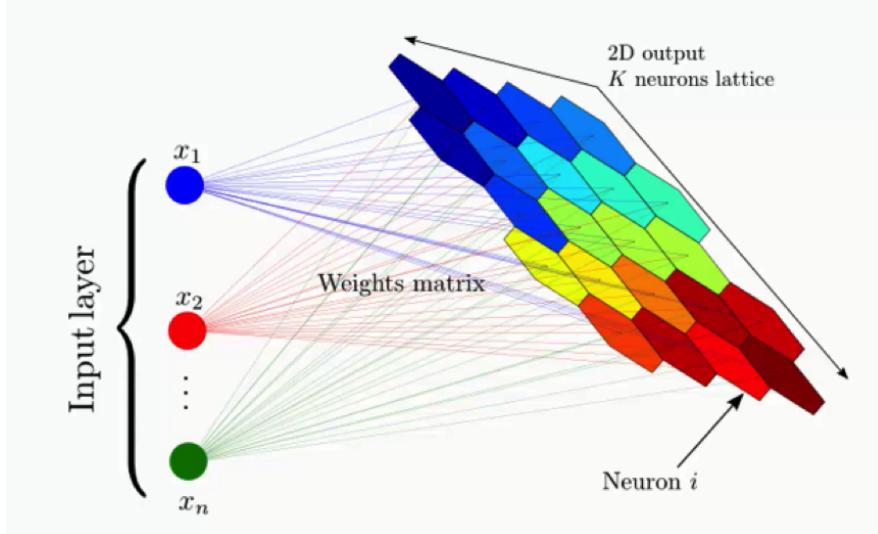


Figure 32: SOM. Crediti: Fonte

## 12.6 Clustering Evaluation

L'analisi e la valutazione delle prestazioni degli algoritmi di clustering rivestono un ruolo fondamentale nell'approfondire l'efficacia e la validità dei risultati ottenuti. Questa sezione si propone di esaminare approfonditamente le metriche impiegate per valutare la qualità dei clustering generati, offrendo una visione chiara sulle metodologie di valutazione utilizzate.

Intrinseca ed Estrinseca emergono come le due categorie principali di valutazione.

Il contesto della valutazione intrinseca, è i, caso tipico, dove non disponiamo di alcuna verità fondata o etichetta predefinita per i dati, ci concentriamo sull'analisi interna al processo di clustering.

Dall'altra parte, la valutazione estrinseca, si basa su verità fondate o etichettature preesistenti.

Questo può accadere quando esaminiamo dati sintetici, piccoli dataset etichettati a mano o dataset etichettati a posteriori. Questi approcci ci consentono di valutare la performance del clustering in contesti dove è disponibile una qualche forma di verità fondata.

La comprensione approfondita di tali criteri di valutazione è essenziale per un approccio informatico efficace nell'esplorare e interpretare modelli complessi nei dati.

### 12.6.1 Valutazione Intrinseca

#### L'idea

L'idea è un cluster è buono se gli oggetti dello stesso cluster sono vicini e se il cluster più vicino è distante.

Introduciamo i concetti di **intra-class similarity** e **inter-class dissimilarity**:

Per un oggetto  $o_i$  che appartiene al cluster  $C_h$ :

- **Intra-class similarity:** (mi aspetto un risultato piccolo)

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

- **Inter-class dissimilarity:** (mi aspetto un risultato grande)

Calcolo questo valore per tutti i k cluster ma ritornando il valore minimo trovato.

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

Combino queste due misure nel **silhouette coefficient**.

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

la funzione torna un output compreso tra -1 e 1, più  $s(o_i) \rightarrow 1$ , migliore è il cluster. Quindi così otterremo lo score di bontà di clustering per un singolo oggetto. **Eseguiamo la funzione su tutti gli oggetti, prendiamone la media, ed otterremo lo score di bontà di tutti i cluster.**

Nota: Se la misura di distanza non è adatta, il risultato non è attendibile

Potremmo utilizzare il silhouette coefficient, per trovare il giusto k in un k-means.

### 12.6.2 Valutazione Estrinseca

In questa situazione ci troviamo nel caso ideale, definiamo:

1.  $C(o_i)$  Il cluster assegnato dall'algoritmo all'oggetto  $o_i$
2.  $L(o_i)$  il cluster reale dell'oggetto  $o_i$

Formalmente:

1.  $TP = \{o_i, o_j : C(o_i) = C(o_j) \text{ and } L(o_i) = L(o_j)\}$
2.  $FP = \{o_i, o_j : C(o_i) = C(o_j) \text{ and } L(o_i) \neq L(o_j)\}$
3.  $TN = \{o_i, o_j : C(o_i) \neq C(o_j) \text{ and } L(o_i) \neq L(o_j)\}$
4.  $FN = \{o_i, o_j : C(o_i) \neq C(o_j) \text{ and } L(o_i) = L(o_j)\}$

Sulla base di queste definizioni vengono generalmente utilizzate due statistiche:

$$RandStatistic = \frac{TP + TN}{TP + TN + FP + FN}$$

$$JaccardCoeff = \frac{TP}{TP + FN + FP}$$

La Random Statistic ci dice: per quante coppie è stata effettuata una corretta decisione di clustering.

## 13 Associative Rules

Association Rule Mining è un metodo per l'identificazione di patterns frequenti, correlazioni, associazioni o strutture casuali in set di dati. Le associations rules sono delle regole che, dato un insieme di transazioni, predicono la presenza di un elemento basandosi sulla presenza o meno dell'elemento in altre transazioni. Si noti che queste regole sono regole di co-occorrenza e non regole di causalità. Una regola di associazione è normalmente rappresentata nella forma:  $\{X\} \Rightarrow \{Y\}$   $\{X\}$  è la parte antecedente, che si trova nei dati.  $\{Y\}$  è la parte conseguente, ovvero qualcosa che si trova in congiunzione con la parte antecedente Tabella di transazioni:

| 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 |

Table 5

### Example of Association Rules

1.  $\{\text{Diaper}\} -> \{\text{Beer}\}$ ,
2.  $\{\text{Milk}, \text{Bread}\} -> \{\text{Eggs}, \text{Coke}\}$ ,
3.  $\{\text{Beer}, \text{Bread}\} -> \{\text{Milk}\}$ ,

### 13.1 Definizioni

Adesso si forniscono alcune definizioni fondamentali per lavorare con le regole associative.

- **Itemset:** è una collezione di uno o più items (i.e:  $\{\text{Milk}, \text{Bread}\}$ )
  - Un k-itemset è un itemset composto da k elementi
- **Support count ( $\sigma$ ):** frequenza dell'itemset (i.e:  $\sigma(\text{Milk,Bread,Diaper}) = 2$ )

Per **valutare** una regola associativa “ $X \rightarrow Y$ ”, si usano i seguenti concetti:

- **Supporto:** Quante volte è frequente la transazione in T, dove T è l'insieme delle transazioni  $\frac{\sigma(X \cup Y)}{|T|}$
- **Confidenza:** Quante volte appare Y in transazioni che contengono X  $\frac{\sigma(X \cup Y)}{\sigma(Y)}$

**Nota:** Il valore del supporto per una transazione può anche essere relativamente basso, è però importante che la confidenza sia molto alta.

### 13.2 Obiettivo delle Associative Rules

L'obiettivo del association rule mining è quello di trovare, dato un set di transazioni  $T$ :

- supporto  $\geq \text{minsup}$
- confidenza  $\geq \text{minconf}$

Un itemset si dice "frequente" se il suo supporto è  $\geq \text{minsup}$

#### 13.2.1 Approccio Brute-Force

L'idea è banale, enumero tutte le possibili regole, per ognuna calcolo supporto e confidenza, e poi butto via quelle che non rispettano le soglie minsup e minconf. Questo approccio è **computazionalmente proibitivo**.

## Ottimizzazione

Possiamo fare meglio! L'idea: siccome tutte le regole generate dallo stesso itemset hanno lo stesso supporto, generiamo tutti gli items con supporto  $\geq \text{minsup}$ . Per ogni item frequente genero tutte le regole e ne calcolo la confidenza.

1. Frequent Itemset Generation

2. Rule Generation

Anche questo non è computazionalmente efficiente:  $O(NMw) \rightarrow M = 2^d$

L'unica strategia applicabile consiste in:

1. ridurre il numero di candidati (M)
2. ridurre il numero di transazioni (N)
3. ridurre il numero di comparazioni (NM)

Per applicare questa soluzione, si sfrutta la tecnica **Apriori**.

### 13.3 Apriori

#### 13.3.1 Apriori Principle

Se un itemset è frequente, allora anche tutti i suoi sottoinsiemi **devono** essere frequenti.

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

Questa è anche nota come: proprietà **anti-monotona** del supporto.

**Corollario:**

Se un itemset è poco frequente, allora anche tutti i suoi sovrainsiemi **devono** essere poco frequenti.

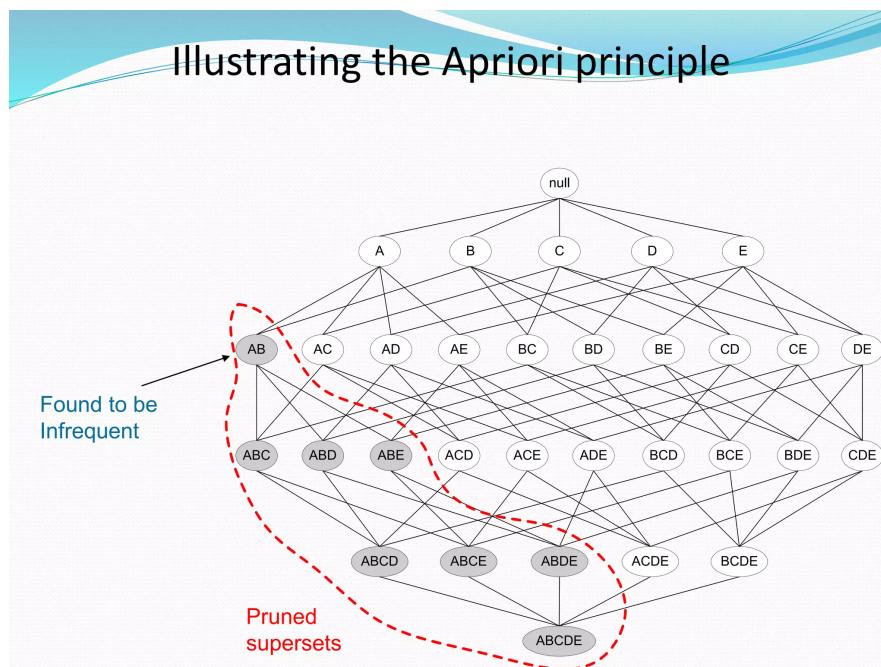


Figure 33: Crediti: Fonte

### 13.3.2 Strategie

Si tratta di un algoritmo “level wise”, ovvero esplora un livello alla volta (prima tutti quelli lunghi uno, poi quelli lunghi due, ...) Sfrutta due strategie:

1. **support-based pruning**, l’algoritmo sceglie solo le regole il cui supporto supera una soglia minima imposta.
2. **anti-monotone property**, il supporto per un set di elementi non supera mai il supporto dei suoi sottoinsiemi. In formula:  $\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$

### 13.3.3 Algoritmo

Siano:

$F_k$  : Frequent k-itemsets

$L_k$  : Candidate k-itemsets

---

#### Algorithm 3 Algoritmo Apriori

---

- 0:  $K \leftarrow 1$
  - 0: Genera  $F_1$  come insieme di 1-itemsets frequenti
  - 0: **repeat**
  - 0:   Genera  $L_{K+1}$  da  $F_K$  {Generazione Candidati}
  - 0:   Elimina i set candidati di  $L_{K+1}$  che contengono subset poco frequenti di lunghezza  $K$
  - 0:   Conta il numero di supporti ( $\sigma$ ) di ogni candidato in  $L_{K+1}$  analizzando il database {Support Counting}
  - 0:   Elimina i candidati in  $L_{K+1}$  che non sono frequenti, lasciando solo quelli la cui frequenza  $\geq F_{K+1}$  {Eliminazione Candidati}
  - 0:    $K \leftarrow K + 1$
  - 0: **until**  $F_K$  è vuoto =0
- 

### Generazione di candidati

1. **Brute-Force** si generano tutte le possibili combinazioni k-arie e da queste si fa il pruning.

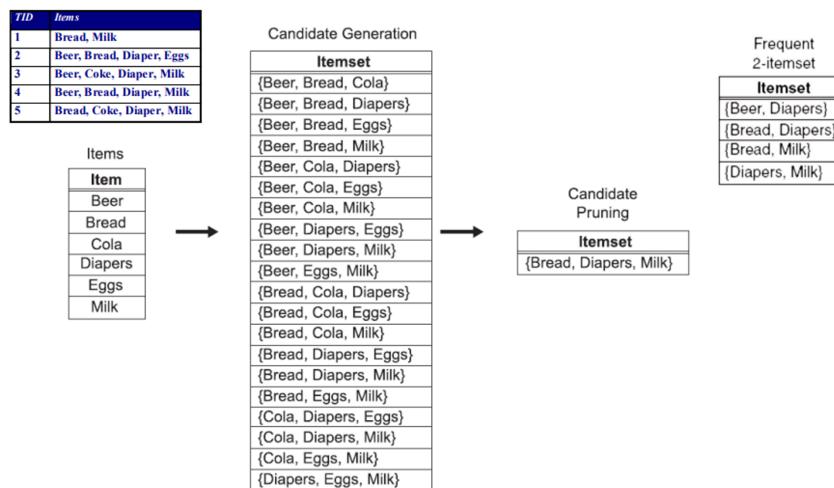
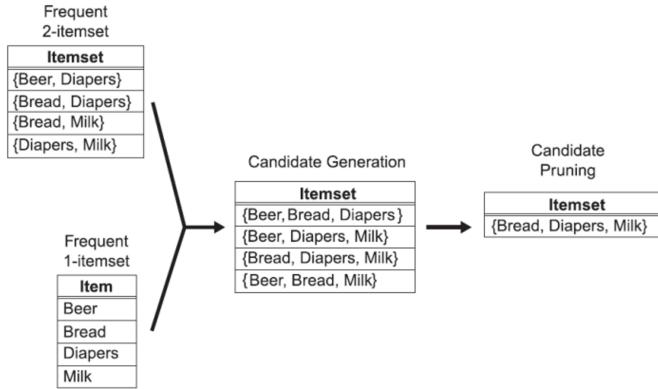


Figure 5.6. A brute-force method for generating candidate 3-itemsets.

Figure 34: Crediti: Slides prof. Lucchese

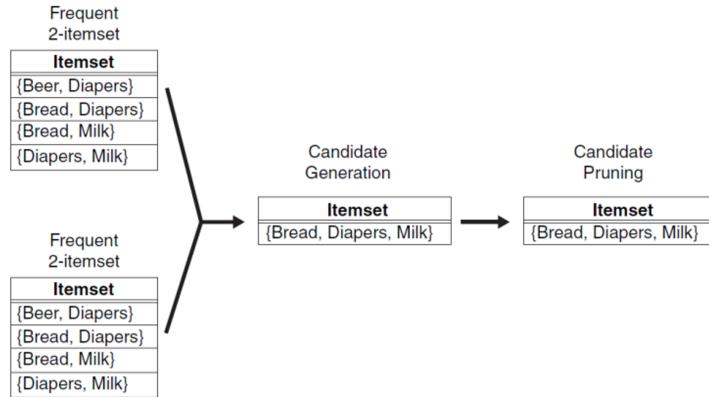
2. **Merge**  $F_{k-1} \times F_1$ : si generano le possibili combinazioni unendo gli item frequenti di lunghezza  $k - 1$  ( $F_{k-1}$ ) con gli item frequenti di lunghezza 1 ( $F_1$ ).



**Figure 5.7.** Generating and pruning candidate  $k$ -itemsets by merging a frequent  $(k-1)$ -itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

Figure 35: Crediti: Slides prof. Lucchese

3. **Merge  $F_{k-1} \times F_{k-1}$ :** si generano le possibili combinazioni unendo gli item frequenti di lunghezza  $k-1$  ( $F_{k-1}$ ) con gli stessi item frequenti di lunghezza  $k-1$  ( $F_{k-1}$ ). Ricordiamo che per effettuare il merge gli item devono condividere il prefisso per una lunghezza di  $k-2$ .



**Figure 5.8.** Generating and pruning candidate  $k$ -itemsets by merging pairs of frequent  $(k-1)$ -itemsets.

Figure 36: Crediti: Slides prof. Lucchese

### 13.4 FP-Growth

**Nota:** Il professore non ha caricato le slide per l'anno accademico 2023/2024.  
 Alla fine dell'algoritmo FP-growth, l'albero FP (Frequent Pattern Tree) risulterà essere una struttura gerarchica che rappresenta in modo compatto le informazioni sui pattern frequenti presenti nel dataset.  
 Questo albero ci permette di estrarre le informazioni sui pattern frequenti senza la necessità di scansionare ulteriormente il dataset.

### 13.4.1 Algoritmo

---

**Algorithm 4** FP-Growth

---

```
0: procedure FP-GROWTH(D, T)
0:    $T \leftarrow$  CostruisciFP-Tree( $D, T$ )
0:   for each  $i$  in  $D$  do
0:      $Di \leftarrow$  CostruisciConditionalPatternBase( $D, i$ )
0:      $Ti \leftarrow$  CostruisciPF-Tree( $Di, T$ )
0:     if  $Ti$  ha nodi then
0:       FP-GROWTH( $Di, T$ )
0:     end if
0:   end for
0:   return  $T$ 
0: end procedure=0
```

---

---

**Algorithm 5** FP-Growth

---

```
0: procedure COSTRUISCIFP-TREE(D, T)
0:   if  $T == \emptyset$  then
0:      $T \leftarrow \emptyset$ 
0:   end if
0:   for each  $t$  in  $D$  do
0:     Ordina elementi di  $t$  in base alla frequenza in  $D$ 
0:     Aggiungi gli elementi di  $t$  a  $T$  rispettando l'ordine
0:   end for
0:   return  $T$ 
0: end procedure=0
```

---

---

**Algorithm 6** FP-Growth

---

```
0: procedure COSTRUISCICONDITIONALPATTERNBASE(D, i)
0:    $res \leftarrow \emptyset$ 
0:   for each  $t$  in  $D$  do
0:     if  $i$  in  $t$  then
0:       Aggiungi  $t$  a  $res$ 
0:     end if
0:   end for
0:   return  $res$ 
0: end procedure=0
```

---

## 14 Recommender System

I recommender systems sono un sistema di **filtering delle informazioni** con lo scopo di fornire consigli/raccomandazioni.

La *similarità* è ciò su cui si fonda il problema.

Un recommender system si valuta secondo le seguenti misure di qualità:

1. efficienza nella *costruzione del modello*: costo del processare i dati e costruire il RS e analisi per generare i dati da utilizzare;
2. efficienza nella *generazione dei suggerimenti*: costo delle raccomandazioni a run-time;
3. *serendipity* delle raccomandazioni: esse devono essere nuove, non banali ed inaspettate aiutando gli utenti ad esplorare nuovi oggetti;
4. adattamento al problema della “partenza a freddo” (*Cold Start Problem*): come si comporta il modello quando ha a che fare con nuovi utenti e/o nuovi items?

### 14.1 Utenti, items e modellazione

Prima di tutto, denotando l’insieme degli utenti come  $U$  e l’insieme degli items come  $I$ , si formula il profilo dell’utente  $u \in U$  come:

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

Cioè la media degli items con cui ha interagito.

Si ha inoltre che il testo verrà modellato usando il vector space model:

- ogni item  $x \in I_u$  è un vettore di lunghezza  $N$ , con  $N$  corrispondente al numero di parole del lessico (stemming, lemming, rimozione delle stop-word vengono applicati qui).
- $x[t] = t \cdot f(t, x) \cdot i \cdot d \cdot f(t, I)$  dove:
  - $t \cdot f(t, x)$  è la frequenza del termine  $t$  nell’item  $x$
  - $i \cdot d \cdot f(t, I)$  è la frequenza inversa di  $t$  nell’insieme di items  $I$

Si identificano 3 tipi di recommender systems che utilizzano, ognuno, delle strategie diverse: Content Based, User-Based e Item-Based.

### 14.2 Content-Based

L’idea è quella di sfruttare la similarità tra gli items

#### 14.2.1 Algoritmo

---

##### Algorithm 7 Content-Based

---

```
0: procedure CALCOLASIMILARITÀ( $u, x, k$ )
0:   Similarità:  $\cos(u, x) = \frac{u \cdot x}{\|u\| \cdot \|x\|}$   $\{(1 \text{ per massima similarità, } -1 \text{ per massima dissimilarità})\}$ 
0:   return i migliori  $k$  items simili al profilo utente
0: end procedure=0
```

---

#### 14.2.2 Misure utilizzate

Il filtraggio basato sul contenuto utilizza le features degli items per consigliare altri items simili a quelli che piacciono all’utente.

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

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

Il sistema restituirà i  $k$ -items più simili al profilo dell’utente.

### 14.2.3 Valutazione

Tale strategia è:

1. **Facile** da costruire in quanto la costruzione di un vettore per ciascun item non è tanto costosa;
2. **Non è efficiente** nella generazione dei suggerimenti siccome deve cercare i vicini più "vicini" tra tutti i documenti;
3. **Poca serendipity** proprio perché è legato alla similarità del testo di un item;
4. **Cold-Start Problem "parziale"** in quanto un utente dovrebbe prima "toccare" almeno un item.

## 14.3 User-Based

Piuttosto che cercare items simili, cerchiamo utenti simili!

### 14.3.1 Algoritmo

---

#### Algorithm 8 User-Based

---

```

0: procedure RACCOMANDAZIONEUSERBASED( $u, N(u)$ )
0:   Cerchiamo  $N(u)$  il set di utenti simili ad  $u$  utilizzando Pearson correlation
0:   Sfruttiamo i rating degli utenti  $N(u)$  per costruire suggerimenti per  $u$ 
0: end procedure=0

```

---

### 14.3.2 Misure utilizzate

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.

Le tecniche di valutazione possono essere:

- esplicite, come le stelle assegnate ad un determinato film.
- implicite, come il tempo trascorso su una determinata pagina, o i click dell'utente su una pagina di risultati.

L'algoritmo consiste in:

1. Trovare un set di vicini dell'utente  $N(u)$ , utenti simili a  $u$ .
2. Sfrutta le valutazioni di questo set di utenti  $\mathbf{N}(u)$  per fornire raccomandazioni a  $u$ .

Per trovare i vicini dell'utente  $N(u)$  si sfrutta il coefficiente di correlazione di Pearson, il quale diventa più grande man mano che aumenta la similarità tra due utenti:

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

Ad ogni item  $i \in I$  viene assegnato uno score di similarità  $s_{ui}$  dove  $u \in U$  (score dell'item  $i$  per l'utente  $u$ ):

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

Nel caso in cui  $s_{ui}$  venisse usato per rankare ogni item di  $i$ , la formula si aggiorna:

$$r_{ui} = \bar{u} + s_{ui}$$

### 14.3.3 Valutazione

Tale strategia è:

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

## 14.4 Item-Based

Sfruttiamo sempre la similarità tra items, ma sfruttiamo anche lo score che darebbe un utente  $u$  ad un item  $i$ .

### 14.4.1 Algoritmo

---

#### Algorithm 9 Raccomandazione Item-Based

---

```
0: procedure RACCOMANDAZIONEITEMBASED( $u, i, N(i)$ )
0:    $N(i)$  è l'insieme di item simili a  $i$  e votati da  $u$  (usiamo l'adj cosine)
0:   Sfruttiamo quindi il rating che ha dato  $u$  agli item  $N(i)$  per computare il rating di  $i$ 
0: end procedure=0
```

---

### 14.4.2 Misure utilizzate

Con item-based si cambia di prospettiva.

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 da tenere in considerazione la scale di ratings che normalmente dà il singolo utente (Adjuster Cosine Similarity):

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

Da qui, risulta possibile calcolare il rating user-item come:

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

---

### 14.4.3 Valutazione

Tale strategia è:

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

## 14.5 Tabella di Confronto

Qui si illustra una tabella riassuntiva di valutazione dei 3 recommendere system visti.

Table 6: Confronto tra Content-Based, User-Based e Item-Based

| Parametro                                     | Content-Based     | User-Based         | Item-Based         |
|-----------------------------------------------|-------------------|--------------------|--------------------|
| Efficienza nella Costruzione del Modello      | Efficiente        | Costosa            | Efficiente         |
| Efficienza nella Generazione dei Suggerimenti | Costosa           | Efficiente         | Efficiente         |
| Serendipity                                   | Bassa             | Alta               | Media              |
| Cold Start Problem                            | Meno problematico | Molto problematico | Molto problematico |

## 15 Crediti

I contenuti di queste dispense sono stati tratti dalle seguenti fonti:

1. Libro di testo: Tan, P.-N., Steinbach, M., Kumar, V. (Year). Introduction to Data Mining (2nd ed.). Publisher
2. Libro di testo: Python Data Science Handbook: VanderPlas, J. (2017). Python Data Science Handbook. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. Prima edizione, Dicembre 2016.
3. Notebook del prof. Lucchese
4. <https://www.youtube.com/watch?v=IHZwWFHWa-w>
5. <https://medium.com/>

Si ringraziano gli autori originali per il loro contributo alla conoscenza e alla divulgazione dei concetti di Data and Web Mining.

© 2024 Domenico Sosta. Tutti i diritti riservati.